Differentiable Fluids with Solid Coupling for Learning and Control

Tetsuya Takahashi[1, 2], **Junbang Liang**[2], **Yi-Ling Qiao**[2] **and Ming C. Lin**[2]
[1]Adobe,
[2]University of Maryland at College Park

Abstract

We introduce an efficient differentiable fluid simulator that can be integrated with deep neural networks as a part of layers for learning dynamics and solving control problems. It offers the capability to handle one-way coupling of fluids with rigid objects using a variational principle that naturally enforces necessary boundary conditions at the fluid-solid interface with sub-grid details. This simulator utilizes the adjoint method to efficiently compute the gradient for multiple time steps of fluid simulation with user defined objective functions. We demonstrate the effectiveness of our method for solving inverse and control problems on fluids with one-way coupled solids. Our method outperforms the previous gradient computations, state-of-the-art derivative-free optimization, and model-free reinforcement learning techniques by at least one order of magnitude.

1 Introduction

Differentiable physics has been introduced recently as a powerful and effective approach to solving control and inverse problems due to the differentiability that makes it possible to compute the gradient with user-defined objective functions in a neural network. The computed gradient allows us to use gradient-based optimizers, such as gradient descent and quasi-Newton method, that are generally much more efficient than derivative-free optimization methods. Given the larger number of degrees of freedoms (DOFs) for complex dynamical systems (e.g., DOFs of fluids can be more than 10k) and their high computational cost for simulation over many time steps, it is indispensable to take advantage of the gradient information to efficiently minimize the objective functions.

Because of the effectiveness, many researchers investigated differentiable physics formulations and their applications, e.g., for rigid bodies (Belbute-Peres et al. 2018; Degrave et al. 2016), articulated rigid bodies (Geilinger et al. 2020; Sueda 2020), deformable solids (Hu et al. 2019; Weiss et al. 2020), thin shells and cloths (Liang, Lin, and Koltun 2019), coupling of rigid bodies and cloths (Qiao et al. 2020), smokes (Holl, Thuerey, and Koltun 2020; Um et al. 2020), and liquids (Sanchez-Gonzalez et al. 2020). While some differentiable formulations for fluid materials have been proposed, approaches to achieving differentiable physics differ from each other, typically due to differences in the underlying fluid simulation methodologies. In the literature, there are two types of commonly used fluid simulation methods characterized by distinct spatial discretization techniques: Eulerian grid-based approach and Lagrangian particle-based approach. While the Lagrangian particle-based approach has become popular due to its flexibility, it is known to be challenging to enforce the fluid incompressibility, which plays an important role for realistic fluid behaviors. By contrast, Eulerian grid-based approach has been proven to be efficient and effective in simulating incompressible fluids.

The differentiable approach for Eulerian fluid simulation has been proposed, e.g., with deep neural networks to predict fluid behaviors (Wiewel, Becher, and Thuerey 2019; Wiewel et al. 2020; Holl, Thuerey, and Koltun 2020; Wiewel et al. 2020). However, it is reported that predicting the complex fluid behaviors for a long term can be difficult (Wiewel, Becher, and Thuerey 2019; Wiewel et al. 2020). In addition, fluids involving solid objects can pose additional challenges because of the increased possible fluid and solid configurations, which further make it difficult to learn and predict the fluid dynamics. As such, for reliable predictions of fluid behaviors involving solid interactions, it is typically necessary to compute the fluid and solid dynamics using numerical simulation based on their governing equations.

In this paper, we propose a new differentiable fluid simulation method that can be integrated as a part of layers in deep neural networks. The integrated differentiable simulator enables end-to-end learning of the neural networks with back propagation of the gradient through both of the simulator and networks. The trained networks can generate control inputs for the simulator to handle fluid and solid control problems. Our key contributions are summarized as follows.

1. Our differentiable fluid simulator offers the capability of handling one-way coupled solids due to a variational principle enforcing the fluid incompressibility for realistic fluid behaviors, while appropriately applying the free-slip boundary condition at the sub-grid, fluid-solid interface.
2. To achieve the differentiability, we apply the adjoint method to the entire simulation steps of dynamical systems (unlike partial application of the adjoint method within one time step, e.g., as in (Belbute-Peres et al. 2018;

Liang, Lin, and Koltun 2019; Holl, Thuerey, and Koltun 2020)), thereby enabling efficient and scalable gradient computations for user defined objective functions.

3. Our differentiable fluid simulator computes the gradient based on automatic differentiation (AD) at the high-level, fluid simulation operations (e.g., advection and pressure projection), avoiding the expensive reverse mode AD at the low-level, primitive operations (e.g., addition and subtractions of each variable), commonly implemented in public libraries.

Our validations and experiments demonstrate that our differentiable simulator can achieve at least one order of magnitude performance improvement, compared to the state-of-the-art techniques in inverse and control problems.

2 Related Work

Differentiable physics has been a powerful approach, and various methods have been proposed. Therefore, we first review prior differentiable physics techniques in § 2.1. In §2.2, we discuss several works on machine learning for fluids since machine learning techniques can also be one approach to making physics simulations differentiable. For more machine learning applications on fluid mechanics, we refer to the paper (Brunton, Noack, and Koumoutsakos 2020).

2.1 Differentiable Physics

Differentiable physics allows us to compute the gradient over the sequence of the physics simulation, making it possible to use efficient gradient-based optimizers and thus to efficiently solve inverse and control problems. Degrave et al. (Degrave et al. 2016) proposed a differentiable rigid body simulation method by taking advantage of the automatic differentiation technique implemented in Theano (Theano Development Team 2016). Later, to improve the efficiency, Belbute-Peres et al. (Belbute-Peres et al. 2018) applied the adjoint method to rigid body contact problems at each simulation step in the gradient computation, combining the differentiable techniques developed for quadratic programs (Amos and Kolter 2017). Toussaint et al. (Toussaint et al. 2018) utilized differentiable physics to achieve complex manipulation tasks with a robot. Hu et al. (Hu et al. 2019) presented an algorithm that combines each of the operations in the material point method to achieve a differentiable soft body simulator. Liang et al. (Liang, Lin, and Koltun 2019) presented a differentiable cloth simulator by applying the adjoint method to handle contact problems, similar to the work of Belbute-Peres et al. (Belbute-Peres et al. 2018) while also relying on automatic differentiation implemented in PyTorch (Paszke et al. 2019). Recently, Qiao et al. (Qiao et al. 2020) extended this work to support two-way coupling of cloths and rigid bodies. Holl et al. (Holl, Thuerey, and Koltun 2020) proposed using deep neural networks to predict smoke behaviors and make the fluid simulation differentiable. They also employed the adjoint method to efficiently perform the gradient computation in solving pressure Poisson equations at each simulation step while their approach also utilized automatic differentiation provided by TensorFlow (Abadi et al. 2015). Schenck and Fox (Schenck and Fox 2018) proposed a differential liquid simulator based on Smoothed Particle Hydrodynamics (SPH) by implementing inter-particle force exchanges within neural networks. The differentiability of simulation methods also allow us to compute the gradient, with respect to structure or shapes, as demonstrated for protein (Ingraham et al. 2019), soft bodies (Hu et al. 2019), and plane wings (de Avila Belbute-Peres, Economon, and Kolter 2020).

Automatic Differentiation: In general, implementation of differentiable physics simulation methods is likely to be more complex than that of forward simulation methods. As such, several works utilized the automatic differentiation implemented in public libraries (Degrave et al. 2016; Belbute-Peres et al. 2018; Liang, Lin, and Koltun 2019). However, their low-level automatic differentiation is known to be slow and expensive in memory usage, in exchange for their generality (see e.g., (Hu et al. 2019, 2020)), and specialized approaches (e.g., the adjoint method for high-level automatic differentiation) which take advantage of the underlying system structures are likely to be more efficient. Similar to our work, the adjoint method has been employed to efficiently compute the gradient of objective functions for fluids (McNamara et al. 2004), particle systems (Wojtan, Mucha, and Turk 2006), deformable objects (Weiss et al. 2020), articulated rigid bodies (Geilinger et al. 2020; Sueda 2020). For the gradient computation using the adjoint method for fluid control, we refer to the review paper (Kim and Bewley 2007). As one approach to helping practitioners, Hu et al. (Hu et al. 2020) proposed a programming language, which computes gradient of physics simulation (e.g., for rigid bodies and fluids) based on source code transformations.

Differentiable Fluids: Among differentiable physics approaches, while our work shares a similar goal with a concurrent work, PhiFlow (Holl, Thuerey, and Koltun 2020), there are notable differences. In their work, forward simulation has been replaced with predictions using neural networks for efficiency while we rely on numerical simulation to accurately handle interactions between fluids and rigid objects. In addition, their work combines low-level automatic differentiation and adjoint method to make simulation steps differentiable, whereas we apply the adjoint method over the entire simulation at every time step to make the entire simulator *fully differentiable* on its own without relying on low-level automatic differentiation.

Adjoint Methods: Our differentiable fluid simulator is inspired by (McNamara et al. 2004). However, as a significant departure from their approach, we introduce one-way coupled solid objects to achieve boundary-driven fluid controls and control of rigid bodies in the fluid environment, enforcing appropriate solid boundary conditions with sub-grid details. To achieve the sub-grid accuracy, we formulate the one-way coupling based on a variational principle with volume fractions, which require augmenting the adjoint method, unlike their voxelized simulator. Additionally, to integrate the gradient computation based on the adjoint method with neural networks, we interweave forward simulation, backward gradient computation, control input predictions, and back propagation through neural networks. Such a tight integration has not yet been considered in their work.

2.2 Machine Learning for Fluids

Applying machine learning to fluid simulation has become popular recently. One of the earliest work presented by Ladický et al. (Ladický et al. 2015) demonstrated that the dynamics of SPH fluids can be learned with hand crafted features to predict the behaviors of simulation particles using regression forests. To avoid using hand crafted features, Schenck and Fox (Schenck and Fox 2018) presented a deep-learning-based approach to predicting the behaviors of SPH fluids. This approach is extended to improve the efficiency by using continuous convolution with an auxiliary grid (Ummenhofer et al. 2020) and to support different materials, such as rigid bodies and deformable solids, by dynamically building graph structures (Li et al. 2019). Recently, Sanchez-Gonzalez et al. (Sanchez-Gonzalez et al. 2020) proposed a general framework for learning simulation from data, exploiting graph networks (Battaglia et al. 2018) and demonstrated that their approach is robust to hyperparameter choices across various settings.

In the Eulerian setting, Wiewel et al. (Wiewel, Becher, and Thuerey 2019) proposed a Long Short-Term Memory (LSTM)-based approach to predicting the behaviors of grid-based fluids, and this approach was later extended to robustly predict long-term sequences by dividing the latent space (Wiewel et al. 2020). Morton et al. (Morton et al. 2018) also presented a deep learning framework to predict time evolution of fluid flows based on Koopman theory. Deep learning methods for predicting the dynamics of Reynolds-Averaged Navier-Stokes simulations have been also reported in (Ling, Kurzawski, and Templeton 2016; Thuerey et al. 2020). Unlike these sequence prediction approaches, Tompson et al. (Tompson et al. 2017) presented a Convolutional Neural Network (CNN)-based method for predicting a pressure field at each time step to accelerate solving pressure Poisson equations. To improve visual details of smoke, super resolutions methods have been proposed using CNN (Chu and Thuerey 2017) and Generative Adversarial Networks (GAN) (Xie et al. 2018). To improve the visual details of liquid droplets, Um et al. (Um, Hu, and Thuerey 2018) presented a method for augmenting liquid splashes using neural networks. Prantl et al. (Prantl, Bonev, and Thuerey 2019) presented a generative model for interpolating space-time configurations of fluids represented by signed distance functions, and this approach was later extended by Kim et al. (Kim et al. 2019b) to interpolate more general classes of fluids based on parameters, such as viscosity values. Machine learning techniques for fluids have been also used to achieve interactive car design (Guo, Li, and Iorio 2016; Umetani and Bickel 2018), to control rigid bodies (Ma et al. 2018), and to transfer visual appearances (Kim et al. 2019a, 2020).

Similar to some of previous works for fluids (e.g., (Wiewel, Becher, and Thuerey 2019; Wiewel et al. 2020; de Avila Belbute-Peres, Economon, and Kolter 2020)), our method also uses neural networks. However, these works are complementary to ours, as we use neural networks to learn fluid dynamics and generate control forces for fluids and/or rigid bodies to achieve the desired motions, while their methods use neural networks to predict fluid behaviors.

3 Overview and Preliminaries

The overview of our framework is illustrated in Figure 1. Our differentiable fluid simulator can be naturally integrated with deep neural networks to efficiently perform end-to-end training and to handle control problems. For the forward pass, given the observation from the simulation, the neural network can generate control inputs, which are combined with the current state to perform the forward simulation generating the next state. While the resulting states are used to further forward the simulation states, we can also consider the resulting states as new observations for the input to the neural network. For the backward pass, each of the simulation results can be used to evaluate the objective (loss) function, and the computed loss can be repeatedly back propagated through our differentiable fluid simulator and then deep neural network with the backpropagation techniques to finally give the gradient.

Our differentiable fluid simulator is established on top of the fluid simulation with one-way coupled rigid bodies and the adjoint method for dynamical physics systems. Thus, we first explain the fundamental formulations of fluid simulation with one-way coupled rigid bodies in § 3.1. Then, we briefly review the adjoint method that is specifically designed to efficiently compute the gradient for arbitrary objective functions over multiple simulation steps, unlike adjoint methods applied at an instance (e.g., (Belbute-Peres et al. 2018; Liang, Lin, and Koltun 2019)) in §3.2. The details of our differentiable fluid simulator are given in §4.

3.1 Fluid Simulation with One-Way Coupled Solids

The governing equations for incompressible fluids can be described by the incompressible Euler equations written as

$$\frac{D\mathbf{u}}{Dt} = -\frac{1}{\rho}\nabla \mathbf{p} + \frac{1}{\rho}\mathbf{f}_v, \text{ and } \nabla \cdot \mathbf{u} = 0, \tag{1}$$

where t denotes time, $\frac{D}{Dt}$ material derivative, $\mathbf{u}$ fluid velocity, ρ fluid density, $\mathbf{p}$ fluid pressure, and $\mathbf{f}_v$ external force (e.g., gravity and control forces) for fluid. To advance the simulation step, we use the operator splitting approach; we first address the advection (due to the material derivative) using the semi-Lagrangian method (Stam 1999), apply external forces, and then handle the pressure term, which projects the fluid velocities onto the divergence-free manifold. This step is known as pressure projection and is essential to enforce the fluid incompressibility while satisfying free-slip boundary conditions at the fluid-solid interface. Specifically, fluid velocities can be updated to enforce the incompressibility with pressure forces by $\mathbf{u}^{t+\Delta t} = \mathbf{u}^{**} - \frac{\Delta t}{\rho}\nabla \mathbf{p}^{t+\Delta t}$, where $\mathbf{u}^{**}$ denotes the intermediate fluid velocity after advection and external force steps, and Δt time step size. The pressure $\mathbf{p}$ can be computed, e.g, by solving a pressure Poisson equation derived from the incompressibility constraint $\nabla \cdot \mathbf{u}^{t+\Delta t} = 0$.

Assuming fluid pressure forces are applied to the surface of rigid bodies, the velocity of the rigid bodies can be considered to be updated with $\mathbf{v}^{t+\Delta t} = \mathbf{v}^{**} + \Delta t \mathbf{M}_r^{-1}\mathbf{J}\mathbf{p}^{t+\Delta t}$, where $\mathbf{v}^{t+\Delta t}$ and $\mathbf{v}^{**}$ denote the rigid body velocity after

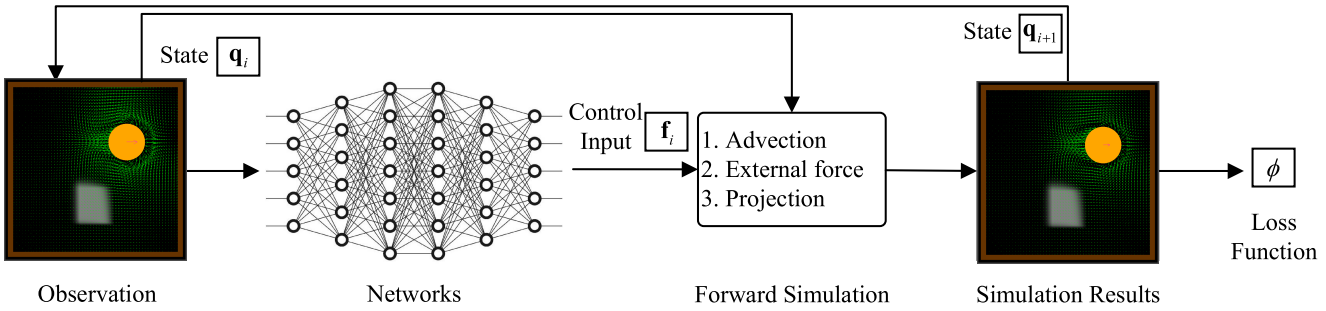

Figure 1: Overview of our framework. Our differentiable fluid simulator is integrated with deep neural networks. Given the observation, the neural network produces control inputs, which are used in the forward simulation to generate the next state (new observation). The next state is used to compute the loss function, and the gradient computation is performed backward through the differentiable simulator and then deep neural networks. Our framework is end-to-end differentiable, which makes it possible to efficiently handle learning and control problems.

and before application of pressure forces, respectively, $\mathbf{M}_r$ the mass of rigid bodies, and $\mathbf{J}$ is a linear operator which integrates the pressures over the surface of the rigid bodies to determine the net pressure forces. Due to a variational principle (Batty, Bertails, and Bridson 2007), one can formulate a discrete minimization problem for pressure to enforce the fluid incompressibility while applying free-slip boundary conditions for solid boundaries as

$$\mathbf{p} = \arg\min_{\mathbf{p}} \frac{1}{2}\Big(\left\|\mathbf{u}^{**} - \Delta t \mathbf{P}^{-1}\mathbf{G}\mathbf{p}\right\|^2_{\mathbf{M}_f} + \left\|\mathbf{v}^{**} + \Delta t \mathbf{M}_r^{-1}\mathbf{J}\mathbf{p}\right\|^2_{\mathbf{M}_r}\Big), \quad (2)$$

where $\mathbf{P}$ denotes a fluid density matrix, $\mathbf{G}$ a discrete gradient operator, and $\mathbf{M}_f$ a diagonal matrix for fluid mass at each cell, and matrix-weighted vector norm $\|\mathbf{u}\|_{\mathbf{W}} = \sqrt{\mathbf{u}^T\mathbf{W}\mathbf{u}}$. Here, to account for the sub-grid details, we introduce volume fractions that represent how much volume is occupied by a material in each cell, and we use diagonal matrices $\mathbf{W}^u_F$ and $\mathbf{W}^u_S$ (where $\mathbf{W}^u_F + \mathbf{W}^u_S = \mathbf{I}$ with the identity matrix $\mathbf{I}$) to denote the volume fractions of fluids and solids in the cell at the fluid velocity samples, respectively. Given $\mathbf{W}^u_F$ and $\mathbf{W}^u_S$, we can consistently define $\mathbf{M}_f = \mathbf{P}\mathbf{W}^u_F$ and $\mathbf{J} = -\mathbf{Q}\mathbf{W}^u_S\mathbf{G}$ with the aggregation matrix $\mathbf{Q}$, which accumulates contributions at the fluid velocity samples to apply to each solid body. Considering one-way coupled solids with infinite mass, i.e., by setting $\mathbf{M}_r^{-1} = 0$, we can obtain the pressure by solving the minimization problem. Then, we update fluid velocity by $\mathbf{u}^{t+\Delta t} = \mathbf{u}^{**} - \Delta t \mathbf{P}^{-1}\mathbf{G}\mathbf{p}$ to enforce the incompressibility.

3.2 Adjoint Method for Dynamical System

The goal of the adjoint method is to efficiently compute the gradient for a user-defined objective function. Given a dynamical system, we start with an initial state $\mathbf{q}_0$ and iteratively apply forward simulation operation $\mathbf{F}_i$ to compute the next state with control input $\mathbf{f}_i$ by $\mathbf{q}_{i+1} = \mathbf{F}_i(\mathbf{q}_i, \mathbf{f}_i)$, evolving the state until we obtain $\mathbf{q}_n$, where n denotes the index for the last state. By concatenating each state and control input into one state vector $\mathbf{q} = (\mathbf{q}_0^T, \ldots, \mathbf{q}_n^T)^T$ and control input vector $\mathbf{f} = (\mathbf{f}_0^T, \ldots, \mathbf{f}_n^T)^T$, we can define our objective function as $\phi(\mathbf{q}, \mathbf{f})$. Given the objective function, we aim to minimize it following the constraint on states due to the forward simulation. This can be formally written as $\mathbf{f} = \arg\min_{\mathbf{f}} \phi(\mathbf{q}, \mathbf{f})$ s.t. $\mathbf{q} = \mathbf{F}(\mathbf{q}, \mathbf{f})$, where $\mathbf{F}$ is the concatenation of $\mathbf{F}_i$ involving consecutive states $\mathbf{q}_i$ and $\mathbf{q}_{i+1}$ only. To use efficient gradient-based optimizers, it is necessary to compute the gradient of the objective function with respect to the control inputs, and the gradient can be written as $\frac{d\phi}{d\mathbf{f}} = \frac{\partial\phi}{\partial\mathbf{q}}\frac{d\mathbf{q}}{d\mathbf{f}} + \frac{\partial\phi}{\partial\mathbf{f}}$. However, the direct computation of $\frac{d\mathbf{q}}{d\mathbf{f}}$ is extremely costly since this term requires computing the gradient for each state with respect to each control input. This expensive computation can be avoided using the adjoint method.

By differentiating $\mathbf{q} = \mathbf{F}(\mathbf{q}, \mathbf{f})$ with respect to $\mathbf{f}$, we obtain $\frac{\partial\mathbf{q}}{\partial\mathbf{f}} = \frac{\partial\mathbf{F}}{\partial\mathbf{q}}\frac{\partial\mathbf{q}}{\partial\mathbf{f}} + \frac{\partial\mathbf{F}}{\partial\mathbf{f}}$, i.e., $(\mathbf{I} - \frac{\partial\mathbf{F}}{\partial\mathbf{q}})\frac{\partial\mathbf{q}}{\partial\mathbf{f}} = \frac{\partial\mathbf{F}}{\partial\mathbf{f}}$. Under this constraint, the computation of $\frac{\partial\phi}{\partial\mathbf{q}}\frac{d\mathbf{q}}{d\mathbf{f}}$ can be efficiently performed using the adjoint method (see e.g., (McNamara et al. 2004)) as $\frac{\partial\phi}{\partial\mathbf{q}}\frac{d\mathbf{q}}{d\mathbf{f}} = \mathbf{r}^T\frac{\partial\mathbf{F}}{\partial\mathbf{f}}$ s.t. $\left(\mathbf{I} - \frac{\partial\mathbf{F}}{\partial\mathbf{q}}\right)^T\mathbf{r} = \left(\frac{\partial\phi}{\partial\mathbf{q}}\right)^T$, where $\mathbf{r}$ is known as the adjoint state. With the adjoint state, we can compute the objective function by $\frac{d\phi}{d\mathbf{f}} = \mathbf{r}^T\frac{\partial\mathbf{F}}{\partial\mathbf{f}} + \frac{\partial\phi}{\partial\mathbf{f}}$ while we have the recursive relation of $\mathbf{r}$ as $\mathbf{r} = \left(\frac{\partial\mathbf{F}}{\partial\mathbf{q}}\right)^T\mathbf{r} + \left(\frac{\partial\phi}{\partial\mathbf{q}}\right)^T$. Due to the structure of the constraint, i.e., forward simulation which involves only two consecutive simulation steps (i and $i+1$), the adjoint state is considered as $\mathbf{r} = (\mathbf{r}_0^T, \ldots, \mathbf{r}_n^T)^T$ and can be efficiently computed via backward computation as $\mathbf{r}_i = \left(\frac{\partial\mathbf{F}_i}{\partial\mathbf{q}_i}\right)^T\mathbf{r}_{i+1} + \left(\frac{\partial\phi}{\partial\mathbf{q}_i}\right)^T$, where $\mathbf{r}_n = \left(\frac{\partial\phi}{\partial\mathbf{q}_n}\right)^T$. We note that it is typically necessary to store $\mathbf{q}$ (and some intermediate results for efficiency) in the forward pass since $\frac{\partial\mathbf{F}_i}{\partial\mathbf{q}_i}$ in the backward adjoint state update depends on $\mathbf{q}_i$. An algorithm for the adjoint method is

summarized in Algorithm 1.

Algorithm 1 Gradient computation with the adjoint method

Store $\mathbf{q}_0$
for $i = 0, \dots, n-1$ **do**
  Forward simulation: $\mathbf{q}_{i+1} = \mathbf{F}_i(\mathbf{q}_i, \mathbf{f}_i)$
  Store $\mathbf{q}_i$ and intermediate results
$\mathbf{r}_n = \left(\frac{\partial \phi}{\partial \mathbf{q}_n}\right)^T$
for $i = n-1, \dots, 0$ **do**
  Backward update: $\mathbf{r}_i = \left(\frac{\partial \mathbf{F}_i}{\partial \mathbf{q}_i}\right)^T \mathbf{r}_{i+1} + \left(\frac{\partial \phi}{\partial \mathbf{q}_i}\right)^T$
  Compute gradient: $\frac{d\phi}{d\mathbf{f}_i} = \mathbf{r}_i^T \frac{\partial \mathbf{F}_i}{\partial \mathbf{f}_i} + \frac{\partial \phi}{\partial \mathbf{f}_i}$

4 Differentiable Fluids with One-Way Fluid-Solid Coupling

In our framework, we consider velocities of fluids and solids as states of the system (i.e., $\mathbf{q} = (\mathbf{u}^T, \mathbf{v}^T)^T$), and if necessary, we can also extend the state to include, e.g., smoke density fields (McNamara et al. 2004; Holl, Thuerey, and Koltun 2020), position and (parameterized) shapes of rigid bodies. To compute the gradient of an objective function with respect to control inputs $\frac{d\phi}{d\mathbf{f}}$ using the adjoint method, in addition to the forward simulation, it is necessary to compute the adjoint state $\mathbf{r}$ (with $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ and $\frac{\partial \phi}{\partial \mathbf{q}}$), $\frac{\partial \phi}{\partial \mathbf{f}}$, and $\frac{\partial \mathbf{F}}{\partial \mathbf{f}}$ in the backward pass.

To achieve specific tasks, for convenience, we formulate our objective function as $\phi(\mathbf{q}, \mathbf{f}) = E_{\text{state}}(\mathbf{q}) + E_{\text{control}}(\mathbf{f})$, where E_{state} evaluates how the state $\mathbf{q}$ is far from our desired state (e.g., key frame), and E_{control} penalizes the use of the control input $\mathbf{f}$. We define these two terms as quadratic functions: $E_{\text{state}}(\mathbf{q}) = \frac{1}{2}(\mathbf{q} - \mathbf{q}^\dagger)^T \mathbf{K}(\mathbf{q} - \mathbf{q}^\dagger)$, and $E_{\text{control}}(\mathbf{f}) = \frac{1}{2}\mathbf{f}^T \mathbf{S}^T \mathbf{C}\mathbf{S}\mathbf{f}$, where $\mathbf{q}^\dagger$ denotes our key frames (desired states), $\mathbf{K}$ a diagonal matrix to weight key frames, $\mathbf{S}$ a selection matrix to specify active control inputs, and $\mathbf{C}$ a diagonal matrix to adjust the magnitude of penalty for using control inputs. Due to the quadratic nature of these objective functions, $\frac{\partial \phi}{\partial \mathbf{f}}$ and $\frac{\partial \phi}{\partial \mathbf{q}}$ can be easily computed by $\left(\frac{\partial \phi}{\partial \mathbf{f}}\right)^T = \mathbf{S}^T \mathbf{C}\mathbf{S}\mathbf{f}$ and $\left(\frac{\partial \phi}{\partial \mathbf{q}}\right)^T = \mathbf{K}(\mathbf{q} - \mathbf{q}^\dagger)$.

Advection. To address the advection in the forward simulation, we use the semi-Lagrangian method (Stam 1999), which updates physical variables with interpolated values at a back-traced location. Considering the fact that this advection operation does not depend on solid velocities, we can write this as $\mathbf{u}^* = \mathbf{F}_{\text{adv}}(\mathbf{u}^t) = L(\mathbf{x} - \Delta t \mathbf{u}^t)\mathbf{u}^t$, where $\mathbf{u}^*$ denotes fluid velocity after the advection step, L an interpolation operator, and thus we obtain $\left(\frac{\partial \mathbf{F}_{\text{adv}}}{\partial \mathbf{u}^t}\right)^T = L(\mathbf{x} - \Delta t \mathbf{u}^t)^T$.

External force. Due to (1) and the operator splitting, the integration of the external force can be formulated as $\mathbf{q}^{**} = \mathbf{F}_{\text{control}}(\mathbf{q}^*, \mathbf{f}) = \mathbf{q}^* + \mathbf{S}\mathbf{f}$, including similarly defined external forces to rigid bodies. From this formulation, we obtain $\left(\frac{\partial \mathbf{F}_{\text{control}}}{\partial \mathbf{q}^*}\right)^T = \mathbf{I}^T$, and $\left(\frac{\partial \mathbf{F}_{\text{control}}}{\partial \mathbf{f}}\right)^T = \mathbf{S}^T$.

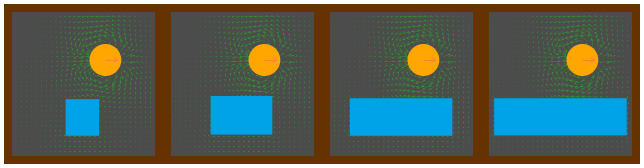

Figure 2: Example frame for FD comparison with different sizes of control areas (cyan) with zero target velocities.

Projection. Due to the quadratic nature of (2), the minimization problem can be handled by solving a linear system. Thus, by combining the linear solve and the fluid velocity update $\mathbf{u}^{t+\Delta t} = \mathbf{u}^{**} - \Delta t \mathbf{P}^{-1}\mathbf{G}\mathbf{p}$, the pressure projection step ($\mathbf{q}^{t+\Delta t} = \mathbf{F}_{\text{proj}}(\mathbf{q}^{**})$) can be written as

$$\begin{pmatrix} \mathbf{u}^{t+\Delta t} \\ \mathbf{v}^{t+\Delta t} \end{pmatrix} = \begin{pmatrix} \mathbf{B} & \Delta t \mathbf{P}^{-1}\mathbf{G}\mathbf{A}^{-1}\mathbf{J}^T \\ \mathbf{O} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{u}^{**} \\ \mathbf{v}^{**} \end{pmatrix}, \tag{3}$$

where $\mathbf{B} = \mathbf{I} - \Delta t \mathbf{P}^{-1}\mathbf{G}\mathbf{A}^{-1}\mathbf{G}^T\mathbf{W}_F^u$, $\mathbf{A} = \Delta t \mathbf{G}^T \mathbf{W}_F^u \mathbf{G}$, and $\mathbf{O}$ is the zero matrix. We note that, as a solution of the minimization problem (2), pressure $\mathbf{p}$ is defined as $\mathbf{p} = \mathbf{A}^{-1}(\mathbf{G}^T\mathbf{W}_F^u\mathbf{u}^{**} - \mathbf{J}^T\mathbf{v}^{**})$, which we solve with Modified Incomplete Cholesky Conjugate Gradient (MICCG). The computed pressure can be substituted to (3) for efficiency. Since the adjoint state involves the transpose of the system matrix, in the backward pass, $\left(\frac{\partial \mathbf{F}_{\text{proj}}}{\partial \mathbf{q}^{**}}\right)^T$ is given by

$$\left(\frac{\partial \mathbf{F}_{\text{proj}}}{\partial \mathbf{q}^{**}}\right)^T = \begin{pmatrix} \mathbf{I} - \Delta t \mathbf{W}_F^u \mathbf{G}\mathbf{A}^{-1}\mathbf{G}^T\mathbf{P}^{-1} & \mathbf{O} \\ \Delta t \mathbf{J}\mathbf{A}^{-1}\mathbf{G}^T\mathbf{P}^{-1} & \mathbf{I} \end{pmatrix}. \tag{4}$$

By performing forward and backward computations (see Algorithm 1), we can efficiently compute the gradient. We note that in practice to minimize the memory usage for the adjoint state, we implicitly merge operations for the advection, external force, and projection steps. To integrate the gradient computation with neural networks, we evaluate neural networks with the current state to generate control input before each forward simulation step, and perform back propagation through the neural network after each of backward gradient computations.

5 Experiments

We implemented our differentiable fluid simulator in C++ and integrated it with deep neural networks implemented in PyTorch 1.5 (Paszke et al. 2019). To evaluate the effectiveness of our method, we conduct three types of experiments on an Intel Core i5-7200U with 8GB RAM. First, we perform an ablation study to evaluate the performance gain compared to other gradient computation techniques in §5.1. Then, we apply our method to optimization problems in §5.2 and control problems §5.3. More details and results can be found in the supplementary material.

5.1 Ablation Study

Comparison to numerical differentiation. To demonstrate the efficiency and scalability of our method in the gradient computation, we first compare our adjoint-based method

DOFs	Ours (ms)	FD (ms)	Speed-up
4,200	28	7,232	258.3
8,400	48	13,533	281.9
14,000	43	22,905	532.7
18,200	33	29,170	883.9

Table 1: Performance results for gradient computations. Ours is 2-3 orders of magnitude faster than FD.

with numerical differentiation using the central finite difference (FD). We use four different numbers of control inputs and evaluate computational time using the scene shown in Figure 2. Performance numbers are summarized in Table 1. In this experiment, relative errors are **less than 1.0%** on average, and both gradient computation methods worked equally well generating comparable results.

Our gradient computation is significantly faster than FD since the adjoint method computes the gradient with two passes (forward and backward computations) whose total cost is approximately double of the forward simulation cost. In addition, the computational cost is mostly irrelevant to the number of control inputs (DOFs), and the computational time is similar over different number of control DOFs, suggesting the linear scalability of our method with respect to the simulation resolution and length. By contrast, FD needs to perform two forward simulations for each control DOF. As such, the computational cost becomes much higher than our method and increases linearly with the number of control DOFs. Although FD can be advantageous in terms of memory usage, the performance gain shown here clearly suggests the advantage of our method in practical use.

Comparison to the voxelized method of (McNamara et al. 2004). Due to the volume fractions (for any non-grid-aligned objects including box-shaped solids), our method can achieve *sub-grid accuracy*, in stead of voxel resolution (McNamara et al. 2004), as also reported in the forward simulation (Batty, Bertails, and Bridson 2007). These volume fractions also need to be taken into account in the adjoint update via (4), and neglecting the volume fractions in the backward pass can negatively affect the accuracy of the gradient, as shown in Figure 3. On average, the relative error for the gradient with and without volume fractions was 1.8% and 6.4%, respectively, with almost the same speed.

We also note that our method fully integrates the differentiable fluid simulator with neural networks, interweaving forward and backward computations within our simulator and the neural networks, thereby addressing the control problems shown (Figures 6 and 7), while their work did not consider such integration.

Comparison to PhiFlow (Holl, Thuerey, and Koltun 2020). While this concurrent work suggested to replace the forward simulation with the neural network prediction, their released source code, written in python, supports differentiable fluid simulation without using neural networks. This library uses the low-level AD implemented in TensorFlow, and the adjoint method is only used *within one simulation step* to avoid unrolling of long chains in the pressure projection. In contrast, our method based on the adjoint method is fully differentiable on its own, with our C++ implementation achieving a speed-up of one to two orders of magnitude in performance for gradient computations over 30 frames. Observing these differences, we show comparisons in Table 2.

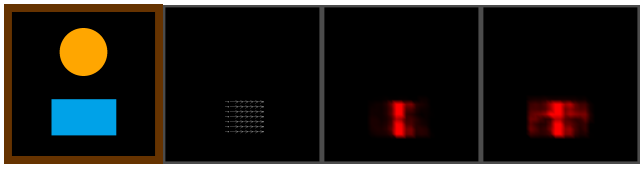

Figure 3: Gradient computation comparison with and without volume fractions. From left to right, setup, initial forces to induce non-zero gradients, relative errors of the gradient with and without volume fractions. Neglecting the volume fractions for the orange sphere and brown wall leads to larger relative errors in the gradient shown in red.

Resolution	Ours (s)	PhiFlow (s)	Speed-up
32×32	0.17	23.08	135.8
64×64	0.88	46.86	53.3
128×128	5.42	175.99	32.5
256×256	42.56	1,032.03	24.2

Table 2: Performance comparison with PhiFlow. Ours is about 1-2 orders of magnitude faster than PhiFlow.

Comparison to low-level AD. We also compare our high-level AD with low-level AD, which we implemented using C++ AD in PyTorch 1.5 (Paszke et al. 2019), and the result is summarized in Table 3. Due to the large overhead of the low-level AD, ours was significantly faster. A similar observation has also been reported in (Hu et al. 2019).

5.2 Space-Time Optimization

Space-time optimization for fluid control. Our differentiable simulator makes it possible to use efficient gradient-based optimizers. To demonstrate the efficiency, we compare our method using a gradient descent method with a state-of-the-art derivative-free optimizer, CMA-ES (Hansen and Kern 2004), in a space-time optimization setup with the grid resolution of 32×32 over 5 frames, as shown in Figure 4. We use 4 samples (simulations) in each CMA-ES iteration, and thus one iteration for the gradient descent is generally faster. The goal is to generate control forces to keep fluid velocities zero at specific areas (cyan box in the left most image of Figure 4) while an orange rigid rotating bar perturbs the fluid velocities. The number of active control DOFs is 840.

Resolution	Ours (s)	Low-level AD (s)	Speed-up
32×32	0.17	12.58	74.0
64×64	0.86	54.10	62.8
128×128	5.34	179.10	33.5
256×256	41.27	842.00	20.4

Table 3: Comparison with low-level AD. Ours is about 1-2 orders of magnitude faster than low-level AD.

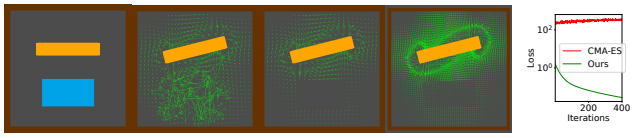

Figure 4: A rotating orange bar perturbing fluids. From left to right, setup, result (32×32) with CMA-ES and our method, result (64×64) with ours, and a convergence profile for results (32×32) with CMA-ES and ours. While CMA-ES fails to converge generating spurious velocities, our method converges quickly achieving zero velocities at the areas even with higher resolutions and longer sequences.

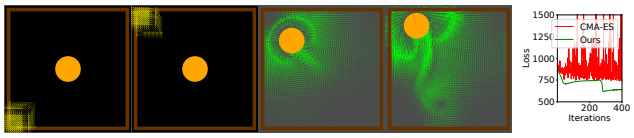

Figure 5: A controlled orange ball to generate our desirable velocities. From left to right, setup with target velocities (yellow arrows) before and after 0.5s, result with CMA-ES and our method, and a convergence profile for results with CMA-ES and ours. While CMA-ES slowly converges, our method converges much faster generating velocities closer to our desired one.

Due to the large number of DOFs, CMA-ES failed to converge and generated velocity fields far from our desired ones, even though their velocity fields are divergence-free. On the other hand, our method quickly converged and generated our desired velocity fields. In addition, our method was able to generate plausible results even with more complex scenarios, where the grid resolution is 64×64 and the number of frames is 50 leading to 67,600 active control DOFs.

Space-time optimization for solid control. To further demonstrate the effectiveness of our method, we compare our method with CMA-ES (same setting as above) using a solid control scenario, as shown in Figure 5. In this problem with the grid resolution of 64×64 over 50 frames, we control an orange rigid ball to generate some velocity fields at specific regions and times. The number of active control DOFs is 150. While CMA-ES converges slowly, our method converges at least one order of magnitude faster, generating velocity fields closer to our desired velocity fields.

5.3 Learning and Control

The differentiability of our method makes it possible to integrate our simulator with neural networks. To demonstrate the effectiveness, we experiment with a control scenario with the grid resolution of 32×32 (Figure 6). Our goal is to control a rigid orange box to generate desired velocities. We note that unlike previous experiments using the space-time optimization, we use our trained neural networks to generate control inputs at each simulation step. We use the simulation state and time as an input for the neural network, 30 and 20 nodes in the first and second layers, respectively, with the ReLU activation, and finally generate three dimensional control forces for controlling the orange box. For this experiment, we compare our method with a state-of-the-art model-free reinforcement learning method, PPO (Schulman et al. 2017). For PPO, we set the simulation state and time as the state space, control outputs as the action space, and defined the reward function as a negated objective function.

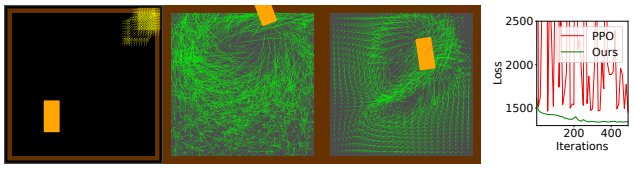

Figure 6: An orange box moving due to the control input from our trained neural networks to generate our desirable velocities. From left to right, setup with target velocities, result with PPO and ours, and a convergence profile for each PPO iteration (sampling) and gradient descent iteration with ours. Our method converges much faster than PPO and generates velocities closer to our desired one.

Figure 7: A cyan box controlled with our trained neural networks to generate our desired velocities to disperse smoke within an (invisible) 3D domain.

Due to the derivative-free nature of PPO, it typically requires more simulation samples to decrease the loss. By contrast, our method can achieve at least one order of magnitude faster convergence by taking advantage of the gradient information. This efficiency enables us to simulate a solid body control even in more costly 3D scenarios (see Figure 7).

6 Conclusions

We proposed a new differentiable fluid simulator that can be integrated with deep neural networks. Our method formulates one-way coupling with rigid bodies as a minimization problem based on a variational principle while enforcing the free-slip boundary conditions at the fluid-solid interface with sub-grid accuracy. We utilize the adjoint method to make the fluid simulation fully differentiable by implementing the backward computation for efficiency, without relying on the low-level AD. We demonstrated effectiveness of our method, achieving significant performance gain *at least one order of magnitude* over the previous gradient computations with FD and low-level AD, state-of-the-art derivative-free optimization (CMA-ES), and model-free reinforcement learning techniques (PPO).

Given the effectiveness and generality of our framework, we plan to explore interactions of various materials, such as viscous fluids and deformable objects. As another future direction, it would also be interesting to investigate how our method works in real-world environments.

Acknowledgments

We would like to thank the anonymous reviewers for their valuable suggestions and comments. This work was supported in part by National Science Foundation and Elizabeth Stevinson Iribe Chair Professorship.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. URL https://www.tensorflow.org/. (Accessed on 02/19/2021).

Amos, B.; and Kolter, J. Z. 2017. OptNet: Differentiable Optimization as a Layer in Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 136–145.

Battaglia, P.; Hamrick, J. B. C.; Bapst, V.; Sanchez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; Gulcehre, C.; Song, F.; Ballard, A.; Gilmer, J.; Dahl, G. E.; Vaswani, A.; Allen, K.; Nash, C.; Langston, V. J.; Dyer, C.; Heess, N.; Wierstra, D.; Kohli, P.; Botvinick, M.; Vinyals, O.; Li, Y.; and Pascanu, R. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv* URL https://arxiv.org/pdf/1806.01261.pdf.

Batty, C.; Bertails, F.; and Bridson, R. 2007. A Fast Variational Framework for Accurate Solid-fluid Coupling. *ACM Trans. Graph.* 26(3).

Belbute-Peres, F. d. A.; Smith, K. A.; Allen, K. R.; Tenenbaum, J. B.; and Kolter, J. Z. 2018. End-to-End Differentiable Physics for Learning and Control. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 7178–7189.

Brunton, S.; Noack, B.; and Koumoutsakos, P. 2020. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics* 52.

Chu, M.; and Thuerey, N. 2017. Data-Driven Synthesis of Smoke Flows with CNN-Based Feature Descriptors. *ACM Trans. Graph.* 36(4).

de Avila Belbute-Peres, F.; Economon, T. D.; and Kolter, J. Z. 2020. Combining Differentiable PDE Solvers and Graph Neural Networks for Fluid Flow Prediction. In *Proceedings of the 37th International Conference on Machine Learning*, 119:2402–2411.

Degrave, J.; Hermans, M.; Dambre, J.; and wyffels, F. 2016. A Differentiable Physics Engine for Deep Learning in Robotics. *Frontiers in Neurorobotics* 13.

Geilinger, M.; Hahn, D.; Zehnder, J.; Bächer, M.; Thomaszewski, B.; and Coros, S. 2020. ADD: Analytically Differentiable Dynamics for Multi-Body Systems with Frictional Contact. *ACM Trans. Graph.* 39(6).

Guo, X.; Li, W.; and Iorio, F. 2016. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 481–490.

Hansen, N.; and Kern, S. 2004. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *PPSN*.

Holl, P.; Thuerey, N.; and Koltun, V. 2020. Learning to Control PDEs with Differentiable Physics. In *International Conference on Learning Representations*.

Hu, Y.; Anderson, L.; Li, T.-M.; Sun, Q.; Carr, N.; Ragan-Kelley, J.; and Durand, F. 2020. DiffTaichi: Differentiable Programming for Physical Simulation. In *International Conference on Learning Representations*.

Hu, Y.; Liu, J.; Spielberg, A.; Tenenbaum, J. B.; Freeman, W. T.; Wu, J.; Rus, D.; and Matusik, W. 2019. ChainQueen: A Real-Time Differentiable Physical Simulator for Soft Robotics. *IEEE International Conference on Robotics and Automation (ICRA)* .

Ingraham, J.; Riesselman, A.; Sander, C.; and Marks, D. 2019. Learning Protein Structure with a Differentiable Simulator. In *International Conference on Learning Representations*.

Kim, B.; Azevedo, V. C.; Gross, M.; and Solenthaler, B. 2019a. Transport-Based Neural Style Transfer for Smoke Simulations. *ACM Trans. Graph.* 38(6).

Kim, B.; Azevedo, V. C.; Gross, M.; and Solenthaler, B. 2020. Lagrangian Neural Style Transfer for Fluids. *ACM Trans. Graph.* 39(4).

Kim, B.; Azevedo, V. C.; Thuerey, N.; Kim, T.; Gross, M.; and Solenthaler, B. 2019b. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum* 38(2): 59–70.

Kim, J.; and Bewley, T. R. 2007. A Linear Systems Approach to Flow Control. *Annual Review of Fluid Mechanics* 39(1): 383–417.

Ladický, L.; Jeong, S.; Solenthaler, B.; Pollefeys, M.; and Gross, M. 2015. Data-Driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.* 34(6). ISSN 0730-0301. doi:10.1145/2816795.2818129. URL https://doi.org/10.1145/2816795.2818129.

Li, Y.; Wu, J.; Tedrake, R.; Tenenbaum, J. B.; and Torralba, A. 2019. Learning Particle Dynamics for Manipulating Rigid Bodies, Deformable Objects, and Fluids. In *International Conference on Learning Representations*. URL https://openreview.net/forum?id=rJgbSn09Ym.

Liang, J.; Lin, M.; and Koltun, V. 2019. Differentiable Cloth Simulation for Inverse Problems. In *Advances in Neural Information Processing Systems 32*, 772–781.

Ling, J.; Kurzawski, A.; and Templeton, J. 2016. Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *Journal of Fluid Mechanics* 807: 155–166.

Ma, P.; Tian, Y.; Pan, Z.; Ren, B.; and Manocha, D. 2018. Fluid Directed Rigid Body Control Using Deep Reinforcement Learning. *ACM Trans. Graph.* 37(4).

McNamara, A.; Treuille, A.; Popović, Z.; and Stam, J. 2004. Fluid Control Using the Adjoint Method. *ACM Trans. Graph.* 23(3): 449–456.

Morton, J.; Jameson, A.; Kochenderfer, M. J.; and Witherden, F. 2018. Deep Dynamical Modeling and Control of Unsteady Fluid Flows. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 31*, 9258–9268.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; dAlché Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 32*, 8024–8035. Curran Associates, Inc. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Prantl, L.; Bonev, B.; and Thuerey, N. 2019. Generating Liquid Simulations with Deformation-aware Neural Networks. In *International Conference on Learning Representations*.

Qiao, Y.; Liang, J.; Koltun, V.; and Lin, M. C. 2020. Scalable Differentiable Physics for Learning and Control. In *International Conference on Machine Learning*.

Sanchez-Gonzalez, A.; Godwin, J.; Pfaff, T.; Ying, R.; Leskovec, J.; and Battaglia, P. W. 2020. Learning to Simulate Complex Physics with Graph Networks. In *International Conference on Machine Learning*.

Schenck, C.; and Fox, D. 2018. SPNets: Differentiable Fluid Dynamics for Deep Neural Networks. In Billard, A.; Dragan, A.; Peters, J.; and Morimoto, J., eds., *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, 317–335. PMLR.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. URL https://arxiv.org/pdf/1806.01261.pdf.

Stam, J. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, 121–128.

Sueda, S. 2020. Analytically Differentiable Articulated Rigid Body Dynamics. URL https://github.com/sueda/redmax/blob/master/notes.pdf. (Accessed on 02/19/2021).

Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* URL http://arxiv.org/abs/1605.02688.

Thuerey, N.; Weißenow, K.; Prantl, L.; and Hu, X. 2020. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA Journal* 58(1): 25–36.

Tompson, J.; Schlachter, K.; Sprechmann, P.; and Perlin, K. 2017. Accelerating Eulerian Fluid Simulation with Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, 3424–3433. JMLR.org.

Toussaint, M.; Allen, K. R.; Smith, K. A.; and Tenenbaum, J. B. 2018. Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. In *Proc. of Robotics: Science and Systems (RSS 2018)*.

Um, K.; Brand, R.; Fei, Y.; Holl, P.; and Thuerey, N. 2020. Solver-in-the-Loop: Learning from Differentiable Physics to Interact with Iterative PDE-Solvers. *Advances in Neural Information Processing Systems* .

Um, K.; Hu, X.; and Thuerey, N. 2018. Liquid Splash Modeling with Neural Networks. *Computer Graphics Forum* 37(8): 171–182.

Umetani, N.; and Bickel, B. 2018. Learning Three-Dimensional Flow for Interactive Aerodynamic Design. *ACM Trans. Graph.* 37(4).

Ummenhofer, B.; Prantl, L.; Thuerey, N.; and Koltun, V. 2020. Lagrangian Fluid Simulation with Continuous Convolutions. In *International Conference on Learning Representations*.

Weiss, S.; Maier, R.; Cremers, D.; Westermann, R.; and Thuerey, N. 2020. Correspondence-Free Material Reconstruction using Sparse Surface Constraints. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Wiewel, S.; Becher, M.; and Thuerey, N. 2019. Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. *Computer Graphics Forum* 38(2): 71–82.

Wiewel, S.; Kim, B.; Azevedo, V. C.; Solenthaler, B.; and Thuerey, N. 2020. Latent Space Subdivision: Stable and Controllable Time Predictions for Fluid Flow. In *Symposium on Computer Animation*.

Wojtan, C.; Mucha, P. J.; and Turk, G. 2006. Keyframe control of complex particle systems using the adjoint method. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 15–23.

Xie, Y.; Franz, E.; Chu, M.; and Thuerey, N. 2018. TempoGAN: A Temporally Coherent, Volumetric GAN for Super-Resolution Fluid Flow. *ACM Trans. Graph.* 37(4).