DRiFT: Differentiable Grid-Based Rigid-Fluid Coupling for Learning and Control

Anonymous authors
Paper under double-blind review

Abstract

Agents are often trained inside simulated environments to alleviate insufficient real-world data; and gradients from a differentiable simulation can help improve learning and control of complex dynamical systems. DRiFT presents an efficient discrete adjoint method for strongly two-way coupled Eulerian fluid–rigid-body dynamics, with the coupling enforced inside the pressure solve rather than treated as a loosely coupled post-process. In the forward pass, the solid-fluid boundary conditions are solved monolithically in a pressure linear system for stability. For backpropagation, we introduce a novel method of calculating and propagating gradients of combined fluid-solid state using adjoint method, which runs as fast as the forward solve. Our method, customized for coupling rigid bodies with inviscid fluids, is more suitable over general purpose methods like automatic differentiation, for use cases where performance is key for analyzing overall flow patterns and learning fluid properties. We demonstrate how our simulator can help to find an optimal force sequence to guide a pill through a complex fluid-filled blood vessel network, and how it can further generalize to arbitrary target states by helping to train a neural network controller. Additionally, we show the effectiveness of our differentiable simulator in isolation, in optimizing initial states which naturally evolve into desired target states. Finally, we analyze the accuracy, robustness and most importantly, the efficiency of our gradient computation method for learning.

1 Introduction

There is a rising trend of using ML methods to model physical laws and learn complex interactions in dynamical systems (Sanchez-Gonzalez et al., 2020; Li et al., 2021; Greydanus et al., 2019). They find applications in estimating physical parameters, guiding physical systems in control problems, learning system dynamics for fast inference or to model environment for training RL agents (Ha & Schmidhuber, 2018). For this purpose, increasingly sophisticated models (Hafner et al., 2024; Kang et al., 2025) have been developed, which rely on astronomical amounts of data for training, since many physical systems (e.g. fluids) are very high dimensional in nature. Specialised models like PINNs (Raissi et al., 2019) tend to perform better on scarce data, but their physically constrained loss terms are ultimately soft constraints, causing issues like violation of physical laws or limited numerical accuracy. Constantly increasing training times and resource consumption aside, if these models are to be used for prediction purposes, they would require novel data, especially covering rare phenomena, at a rate far exceeding that at which it is collected in reality.

Due to these challenges, models are often trained inside environments simulated using physical models, combined with accurate numerical methods, where new data can be generated on-the-fly (Banerjee et al., 2023). If the simulators are differentiable, they can provide gradients of objective functions grounded in the environment w.r.t. the physical state or control inputs. These gradients can serve as continuous reward signals for efficient training (Lutter et al., 2020; Heiden et al., 2020; Xian et al., 2023). Apart from improving sample efficiency, these gradients can help explore unintuitive trajectories, which are not obvious with unstructured rewards in traditional RL methods, which usually incorporate few agent behaviours like proximity to objective, contraint violation, etc. Differentiable simulators can also be effective in isolation, for derivative based optimization tasks. Consequently, there has been a growing interest in differentiable programming.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

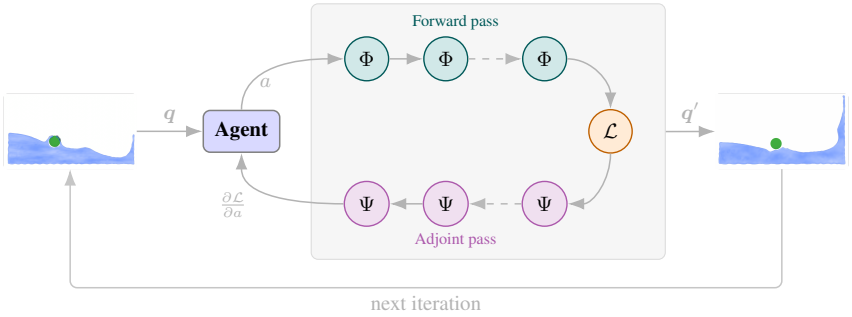


Figure 1: **Differentiable Simulator as an Environment Model.** An agent observes the env. state $\boldsymbol{q}$ and interacts through action a (e.g. a control force). The simulator's forward pass Φ rolls out the trajectory; the adjoint pass Ψ back-propagates the loss gradient $\frac{\partial \mathcal{L}}{\partial a}$, which serves as a training signal for the agent. See Figure 9 for the full simulator pipeline.

There have been significant efforts in developing differentiable simulators for different dynamical systems, including fluids, but relatively few focus on coupling fluids with dynamic obstacles. In fact, because of dynamic boundary conditions, even forward simulation is challenging, making the problem of finding gradients even more so. It is important to consider rigid-fluid coupling as it models agent-environment interaction. So, we present DRiFT, an efficient discrete adjoint method for strongly two-way coupled Eulerian fluid–rigid-body dynamics, with the coupling enforced inside the pressure solve rather than treated as a loosely coupled post-process. Its implementation is jointly parallelized on CPU and GPU for high efficiency. We employ a grid-based discretization using a MAC grid (Harlow & Welch, 1965) combined with an implicit pressure solve, which ensures incompressibility, rigid body constraints and stability in gradient computation. Our main contributions are summarized as follows:

1. We present DRiFT, an efficient discrete adjoint for strongly two-way coupled Eulerian fluid–rigid-body dynamics, with manually derived gradients, as detailed in Section 4.
2. Then, we test the gradients computed by our simulator with a variety of gradient-based optimization tasks. Specifically, in Section 6.1, we learn an optimal control sequence to guide a pill through a blood vessel network. Subsequently, we train a neural network to emit control signals for general target states. Then, in Section 6.2, we optimize the initial state of the system, so that it naturally evolves into a desired final state.
3. In Section 6.3, we provide validation experiments to demonstrate accuracy, robustness and efficiency of our gradient-computation method, and compare it with autodiff frameworks Enzyme Moses & Churavy (2020) and PyTorch Paszke et al. (2019).

2 Related Work

Differentiable simulators have been proposed for many dynamical systems, as reviewed comprehensively by Newbury et al. (2024). They offer exciting new learning-enabled applications like structure identification and discovery (Ingraham et al., 2019; Wang et al., 2020), policy and planning (Mora et al., 2021; Xu et al., 2022) as well as design and fabrication (Huang et al., 2021; Nava et al., 2022). Nzoyem Ngueguin et al. (2023) demonstrate how differentiable programming can solve optimal control under various constraints governed by PDEs, more effectively than PINNs. Many recent works employ differentiable environments for benchmarking reinforcement learning algorithms to solve a variety of tasks depending on system dynamics: FluidLab (Xian et al., 2023) for fluid manipulation, DaXBench (Chen* et al., 2023) and DLO-Lab Cao et al. (2026) for deformable object manipulation, Rewarped (Xing et al., 2025) for multiphysics planning, and FluidGym (Becktepe et al., 2026) for flow control. Out of these, the first four use a hybrid Lagrangian-Eulerian method MPM (Jiang et al., 2016) for simulation due to its versatility in handling different materials, while FluidGym uses the PICT solver (Franz et al., 2026). However, MPM is not suitable for two-way rigid-fluid coupling as it can neither enforce strict rigidity because of particle representation, nor can it enforce strict incompressibility on the liquid, because of explicit pressure updates, which additionally makes the method prone to stability issues.

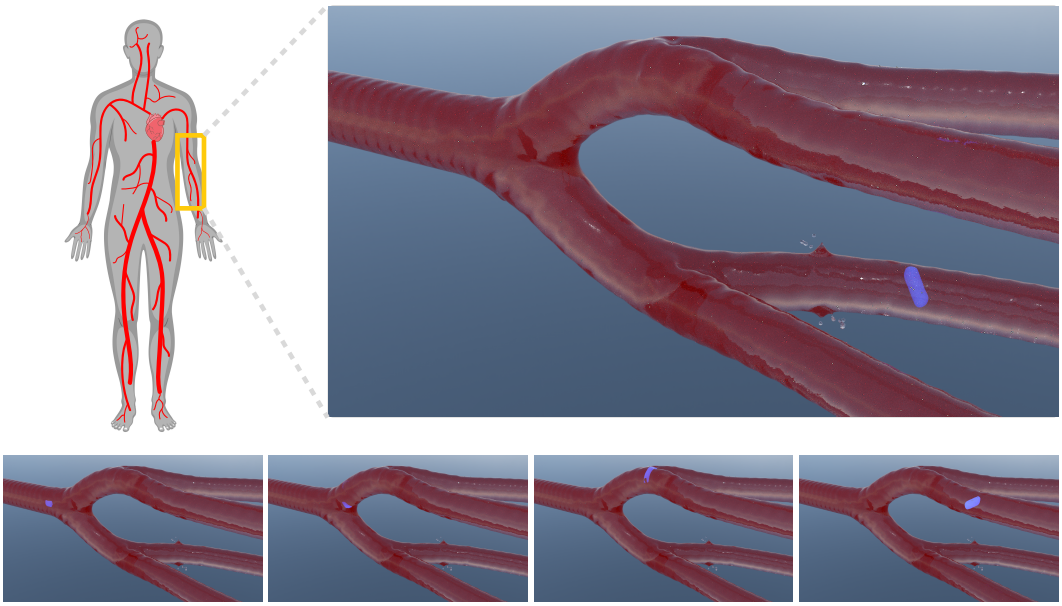


Figure 2: **Guided drug delivery via DRiFT. (a)** Navigating a micro-capsule through a complex blood vessel network requires solid-fluid coupling dynamics. **(b)** Our differentiable simulator constructs a realistic fluid-filled vessel with strong two-way rigid-fluid coupling. **(c)** The gradients from the simulator help to train control forces to be applied near bifurcations (frames 45 and 113) so that the capsule reaches the **front upper target branch**. Optimization trajectory is shown in Figure 4a

Increasing requirements of differentiability in general have spawned sophisticated automatic differentiation (AD) (Baydin et al., 2017) tools, which directly differentiate programs, e.g. by using source code transforms (Hu et al., 2020) or computational graphs (Paszke et al., 2019; Macklin, 2022). PhiFlow (Holl & Thuerey, 2024) and JAX-SPH (Toshev et al., 2024) are frameworks for solving grid-based and particle-based Navier Stokes equation respectively, both of which use AD to compute gradients. Ramos et al. (2022) used PhiFlow, augmented with tailored loss functions, to train a deep neural network controller for a two-way coupled rigid-fluid system. But AD tools are general-purpose and entail significant overhead (Kotary et al., 2023), which makes them slow as compared to manually derived gradients as we show in Section 6.3.

While many methods address fluid control without computing gradients of the objective (Pan & Manocha, 2017; Tang et al., 2021), we take inspiration from McNamara et al. (2004), who compute the simulation gradients of level-set based liquids. Li et al. (2025) propose a novel adjoint method on flow maps for fluid control. Takahashi et al. (2021) extend this to support one-way solid to liquid coupling. Li et al. (2024) further extend the pipeline with derivatives of boundary geometry. Two-way coupling additionally considers force transmitted to the solid from the fluid, which is very important for agents interacting with fluid-based environments. Li et al. (2023) designed a differentiable simulator for particle-based fluids coupled with rigid bodies, but they report instability in the computation of full analytic gradient. Lagrangian methods adopt a unified particle-based representation for fluids and solids, which simplifies treatment of different entities, but suffer from issues similar to MPM, i.e. violation of rigidity and incompressibility, high computation costs because of more degrees of freedom (DOF).

Comparison: Our main contribution is grid-based (Eulerian) a differentiable simulator for *fluids strongly coupled with rigid bodies, which employs far less DOFs*, thus promising *less resource consumption and better stability*. Aquarium (Lee et al., 2023) is the closest work to us in the literature, but there are three important distinctions: First, they use immersed boundary method(IBM) (Peskin,

2002) for fluid structure interaction without free surface handling, while **our work, inspired from variational method of Batty et al. (2007), accounts for partially occupied grid cells using volume fractions, naturally handling free surface**. Secondly, they compute per-step jacobians (i.e. $\frac{d(\text{output})}{d(\text{input})}$) for gradient computation (similar to forward mode auto-diff) which grow quadratically with the size of state. While **we use the adjoint method (Lions, 1971) to represent gradients as adjoint states**, which is a product of the step jacobian and the upstream gradient. Adjoint states have the same size as the system state, thereby using much less memory. This is similar to reverse mode auto diff, which is more efficient in tasks involving a scalar objective function. Third, Aquarium currently only supports 2D scenarios, while **our method supports both 2D and 3D scenarios**.

3 BACKGROUND

3.1 FORWARD SIMULATION

Fluid dynamics are governed by inviscid Navier Stokes equation with the incompressibility constraint:

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho}\boldsymbol{f} - \frac{1}{\rho}\nabla p, \quad \nabla \cdot \boldsymbol{u} = 0, \tag{1}$$

where $\boldsymbol{u}$: fluid velocity, ρ: fluid density, p: fluid pressure and $\boldsymbol{f}$: external force (e.g. gravity) per unit volume. The numerical solution is computed using operator splitting, where each term on the right-hand-side is solved sequentially in writing order. The fluid region, represented as a level-set ϕ (Osher & Fedkiw, 2001), is discretized on a MAC grid (Harlow & Welch, 1965). Rigid body motion is handled using semi-implicit time integration:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \Delta t\ \mathbf{M}_S^{-1}\boldsymbol{F}, \quad \boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \Delta t\ f_x(\boldsymbol{x}_t, \boldsymbol{v}_{t+1}), \tag{2}$$

where $\boldsymbol{v}$: solid velocity, $\boldsymbol{x}$: solid position, $\boldsymbol{F}$: external force (including control forces), and $\mathbf{M}_S$: mass matrix, at centre of mass. f_x is a per element function on vectors, such that $f_x(x, v) = v$, except for quaternions, where $f_x(q, \omega) = 0.5([0, \omega] \otimes q)$, $\otimes$ being quaternion multiplication. (Appendix D)

Let $\boldsymbol{q}_t = (\boldsymbol{u}_t, \phi_t, \boldsymbol{v}_t, \boldsymbol{x_t})$ represent the combined rigid-fluid state and $\boldsymbol{c}_t$ the control force through which the agent (rigid) interacts with the environment (fluid), at time t. If Φ represents the numerical simulation, then $\boldsymbol{q}_{t+1} = \Phi(\boldsymbol{q}_t, \boldsymbol{c}_t)$. Refer to Figure 9 for an overview of the simulation pipeline.

3.2 ADJOINT SIMULATION

Our objective is to find the optimal initial state $\widetilde{\boldsymbol{q}_0}$ and control forces $\widetilde{\boldsymbol{c}}$, collectively denoted as the control variables $\widetilde{\boldsymbol{cv}}$, so that the system results in a desired final state $\widetilde{\boldsymbol{q}_n}$ after n time steps. We pose this as an optimization problem: $\widetilde{\boldsymbol{cv}} = \arg\min_{\boldsymbol{cv}} \mathcal{L}(\boldsymbol{q}_n, \widetilde{\boldsymbol{q}_n})$, where $\mathcal{L}$ is the desired cost function and $\boldsymbol{q}_n = \Phi^n(\boldsymbol{cv})$. Unlike neural networks, we do not need to specialize the loss function or constrain the optimization to enforce boundary conditions, since the simulation based output $\boldsymbol{q}_n$ already enforces them. Because of this, we can use the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{cv}} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{c}}\right)$ to descend to the optimal value $\widetilde{\boldsymbol{cv}}$.

Starting from $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_n}$, we use reverse mode differentiation to obtain $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0}$. For state $\boldsymbol{q}_t$, consider its adjoint state $\boldsymbol{Q}_t = \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_t} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t}, \frac{\partial \mathcal{L}}{\partial \phi_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t}\right)$. From chain rule, $\boldsymbol{Q}_t = \left(\frac{\partial \Phi}{\partial \boldsymbol{q}}\right)^T (\boldsymbol{Q}_{t+1})$. From Eqn. 2, the control force derivative $\boldsymbol{C}_t = \frac{\partial \mathcal{L}}{\partial \boldsymbol{c}_t} = \Delta t \mathbf{M}_S^{-1}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}\right)$. Combining both gives us the adjoint operator Ψ, such that $\boldsymbol{Q}_t, \boldsymbol{C}_t = \Psi(\boldsymbol{Q_{t+1}})$. Composing across n time steps, $\boldsymbol{Q}_0, \boldsymbol{C}_0 = \Psi^n(\boldsymbol{Q}_n)$.

In short, the forward pass Φ advances an initial state $\boldsymbol{q}_0$ under control force sequence $\boldsymbol{c}$ to $\boldsymbol{q}_n$, while the adjoint pass Ψ backpropagates the adjoint state $\boldsymbol{Q}_n$ to the parameter derivatives $\boldsymbol{Q}_0$ and $\boldsymbol{C}$, closing one loop of optimization. In the following section, we provide details for solving each sub-operator of Φ and derive its corresponding adjoint sub-operator, which make up Ψ.

4 DIFFERENTIABLE TWO-WAY RIGID-FLUID COUPLING

The numerical solver for solid-fluid coupling Φ is split into three major steps: advection (Section 4.1), pressure solve (Section 4.2) and rigid body integration (Section 4.3), each of which require separate

numerical treatment and adjoint operators. In grid-based methods supporting free-surface liquids, an additional velocity correction step is also required. This step is discretization dependent and does not model any new dynamics, so we cover it in Appendix F. More details can be found in Figure 9.

4.1 ADVECTION

In grid-based methods, the movement of fluid against the stationary background grid is captured by the advection/convection term $\boldsymbol{u} \cdot \nabla s$, where s is the quantity advected (e.g. $\boldsymbol{u}, \phi$) by the velocity field $\boldsymbol{u}$. We use semi-Lagrangian discretization (Stam, 1999; Bridson et al., 2006) to solve it, according to which if $\mathcal{I}(\boldsymbol{g}, \boldsymbol{y})$ returns the multilinear interpolation of grid $\boldsymbol{g}$ at location $\boldsymbol{y}$ in space and $\boldsymbol{x}_g$ is a sample location, then $s_{t+1}(\boldsymbol{x}_g) = \mathcal{I}(\boldsymbol{s}_t, \boldsymbol{x}_g - \Delta t \mathcal{I}(\boldsymbol{u}_t, \boldsymbol{x}_g))$. Equivalently,

$$\boldsymbol{s}_{t+1} = \mathbf{W}(\boldsymbol{u}_t)\boldsymbol{s}_t, \quad \therefore \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_t} = \mathbf{W}^T \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{t+1}} \right), \quad \therefore \boldsymbol{u}_t^T \left(\frac{\partial \mathbf{W}}{\partial \boldsymbol{u}_t} \right)^T \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{s}_{t+1}} \right), \tag{3}$$

where $\mathbf{W}(\boldsymbol{u}_t)$ is interpolation weight matrix. Rows denote the weights corresponding to the back-traced location $\boldsymbol{x}_g - \Delta t \mathcal{I}(\boldsymbol{u}_t, \boldsymbol{x}_g)$ inside the grid $\boldsymbol{u}_t$, while column represents weights of a $\boldsymbol{s}_t$ sample. The same eqns 3 are used to advect both the velocity field $\boldsymbol{u}$ and the level set ϕ.

4.2 PRESSURE SOLVE

Pressure enforces incompressibility in fluid interior (Ω_F), solid-fluid impenetratibility constraint on solid-fluid boundary (Ω_{FS}) and ghost-fluid boundary condition (Gibou et al., 2002) on free surface (Ω_{FA}). For a rigid body (Ω_S), it behaves as an external force. Essentially, pressure p satisfies:

$$\begin{aligned} \boldsymbol{u}_{t+1}(\boldsymbol{y}) = \boldsymbol{u}_t(\boldsymbol{y}) - \frac{\Delta t}{\rho} \nabla p(\boldsymbol{y}) \text{ and } \nabla \cdot \boldsymbol{u}_{t+1}(\boldsymbol{y}) = 0, \quad \boldsymbol{y} \in \Omega_F \\ p(\boldsymbol{y}) = 0, \quad \boldsymbol{y} \in \Omega_{FA} \\ \boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \Delta t \mathbf{M}_S^{-1} \boldsymbol{F}_p \left(\boldsymbol{x}_t, p \right), \quad \boldsymbol{y} \in \Omega_S \\ \left(\boldsymbol{u}_{t+1}(\boldsymbol{y}) - \boldsymbol{v}_{t+1}(\boldsymbol{y}) \right) \cdot \hat{n} = 0, \quad \boldsymbol{y} \in \Omega_{FS}, \end{aligned} \tag{4}$$

where $\boldsymbol{q}_t = (\boldsymbol{u}_t, \phi_t, \boldsymbol{v}_t, \boldsymbol{x}_t)$: input state, $\boldsymbol{q}_{t+1} = (\boldsymbol{u}_{t+1}, \phi_t, \boldsymbol{v}_{t+1}, \boldsymbol{x}_t)$: output state and $\boldsymbol{F}_p$: external force on rigid body. We use the variational interpretation of pressure (Batty et al., 2007) to convert the equations 4 into a monolithic linear system:

$$\mathbf{A}\mathbf{p} = \frac{1}{\rho} \mathbf{G}^T \mathbf{M}_F \boldsymbol{u}_t - \mathbf{J}^T \boldsymbol{v}_t \text{ where } \mathbf{A} = \left[\frac{\Delta t}{\rho^2} \mathbf{G}^T \mathbf{M}_F \mathbf{G} + \Delta t \mathbf{J}^T \mathbf{M}_S^{-1} \mathbf{J} \right]. \tag{5}$$

where $\mathbf{p}$: pressure field, $\mathbf{G}$: discrete gradient operator, $\mathbf{J}$: linear map such that $\boldsymbol{F}_p = \mathbf{J}\mathbf{p}$ and $\mathbf{M}_F$: fluid mass matrix. The discrete velocity update can then be written as:

$$(\boldsymbol{u}_{t+1}, \boldsymbol{v}_{t+1}) = \left(\boldsymbol{u}_t - \frac{\Delta t}{\rho} \mathbf{G}\mathbf{p}, \; \boldsymbol{v}_t + \Delta t \mathbf{M}_S^{-1} \mathbf{J}\mathbf{p} \right), \tag{6}$$

The full derivation and proof can be found in Bridson et al. (2006). Given equations 5 and 6, we derive (in Appendix B) the corresponding adjoint update as:

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}} + \frac{1}{\rho} \mathbf{M}_F \mathbf{G}\mathbf{s}, \quad \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}} - \mathbf{J}\mathbf{s}, \\ &\frac{\partial \mathcal{L}}{\partial \phi_t} = \frac{\partial \mathcal{L}}{\partial \phi_{t+1}} - \frac{\Delta t}{\rho} \left[\left(\frac{\partial \mathbf{G}}{\partial \phi_t} \right) \mathbf{p} \right]^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} \\ &\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t} \mathrel{+}= \Delta t \; \mathbf{p}^T \left[\mathbf{M}_S^{-1} \left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t} \right) + \left(\frac{\partial \mathbf{M}_S^{-1}}{\partial \boldsymbol{x}_t} \right) \mathbf{J} \right]^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t} - \boldsymbol{v}_{t+1}^T \left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t} \right) \mathbf{s} + \frac{1}{\rho} \boldsymbol{u}_{t+1}^T \left(\frac{\partial \mathbf{M}_F}{\partial \boldsymbol{x}_t} \right) \mathbf{G}\mathbf{s} \end{aligned} \tag{7}$$

where adjoint pressure $\mathbf{s}$ is computed from the linear system:

$$\mathbf{A}\mathbf{s} = \left[\Delta t \mathbf{J}^T \mathbf{M}_S^{-1} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}} \right) - \frac{\Delta t}{\rho} \mathbf{G}^T \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}} \right) \right]. \tag{8}$$

4.3 RIGID BODY TIME INTEGRATION

This step updates the rigid body position, $\boldsymbol{x}_{t+1}$, using velocity $\boldsymbol{v}_{t+1}$: $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \Delta t\ f_x\left(\boldsymbol{x}_t, \boldsymbol{v}_{t+1}\right)$. For translational components $f_x(x, v) = v$ and $\left(\frac{\partial \mathcal{L}}{\partial x_t}, \frac{\partial \mathcal{L}}{\partial v_{t+1}}\right) = \left(\frac{\partial \mathcal{L}}{\partial x_{t+1}}, \Delta t\ \frac{\partial \mathcal{L}}{\partial x_{t+1}}\right)$.

For quaternions $f_x(q, \omega) = 0.5([0, \omega] \otimes q)$ and,

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial q_t} &= \left(I_{4\times 4} + \frac{\Delta t}{2} \frac{\partial\left([0, \omega_{t+1}] \otimes q_t\right)}{\partial q_t}\right) \frac{\partial \mathcal{L}}{\partial q_{t+1}} \\ \frac{\partial \mathcal{L}}{\partial \omega_{t+1}} &= \left(\frac{\Delta t}{2} \frac{\partial\left([0, \omega_{t+1}] \otimes q_t\right)}{\partial \omega_{t+1}}\right) \frac{\partial \mathcal{L}}{\partial q_{t+1}}. \end{aligned} \tag{9}$$

Check Appendix D for partial derivatives of $([0, \omega_{t+1}] \otimes q_t)$. Additionally, collisions with the external boundary are handled using a broad-phase collision resolution algorithm using spherical bounding volumes. The corresponding velocity update and its adjoint is detailed in Appendix E.

5 IMPLEMENTATION

Our simulator is written in C++ and parallelized using OpenMP and CUDA. Forward pass is adapted from *FluidRigidCoupling2D* library (Batty, 2016). We use Eigen (Guennebaud et al., 2010) for all linear algebra operations, libigl (Jacobson & Panozzo, 2017) and tetgen (Si, 2015) for all mesh related operations and *SplashSurf* library (Löschner et al., 2023) to recontruct liquid surface, before being rendered in Blender, or Mitsuba (Jakob et al., 2022). Our simulator is designed as a composition of differentiable operators on the combined solid-fluid state, and exposed to Python using pybind11 (Jakob et al., 2017), so they can be **easily integrated into ML libraries like PyTorch or JAX**.

Experiments in this paper are run on an Intel Xeon W-2123 CPU at 3.6 GHz with 8 threads, 32 GB memory and two Nvidia GeForce GTX 1080 Ti GPUs. Appendix A contains the algorithm pseudocode, with **full code in the supplementary**. Video results can be found on our website.

6 EXPERIMENTS AND EVALUATIONS

6.1 OPTIMAL CONTROL

In this experiment, we demonstrate how gradients from our simulator can help to optimize a control force sequence, which guides the rigid body to a desired final state after a fixed number of time steps.

Blood Vessel Experiment. We use the gradients from the simulator to optimize control forces, which guide a micro-capsule through a blood vessel network and reach at the upper front branch (refer to Figure 2). The domain is of size $5 \times 3.5 \times 3.5$ consisting of $100 \times 70 \times 70$ grid cells of size 0.05. We use Decroocq et al. to create the blood vessel boundary. The simulation is run for 200 frames, with time step size $0.01s$. DRiFT succesfully optimizes the control forces to guide the pill to the correct branch (Figure 2, bottom row).

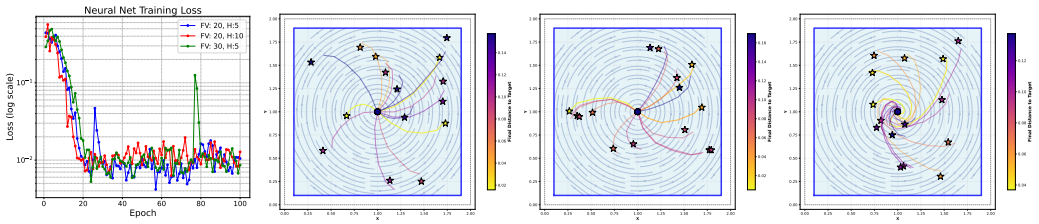
Generalization. We use the gradients from the simulator to train a neural network controller to guide a rigid body towards any given target state, while interacting with the surrounding liquid. The rigid body starts at $(1.0, 1.0)$ in a box domain of size 2×2 with 40×40 grid cells filled with water, which has an initial anticlockwise swirling motion with velocity FV. The neural net takes the initial state as input, and generates a control force sequence, which is applied to the rigid body over the next few time steps i.e. horizon H. We study the performance of controller for different values of FV and H. During training, convergence is reached in all cases (Figure 3a). During testing, the controller is applied to 15 target configurations. Figures 3b, 3c and 3d show the trajectories taken by the rigid body. Regardless of the environment state, the control forces help the rigid body move close to the target, since the gradients from simulator encode exactly that information.

6.2 INITIAL STATE ESTIMATION

In this experiment, we demonstrate how gradients from our differentiable simulator can help to optimize the initial state of the combined rigid-fluid system, which results into a desired final state after a fixed number of time steps.

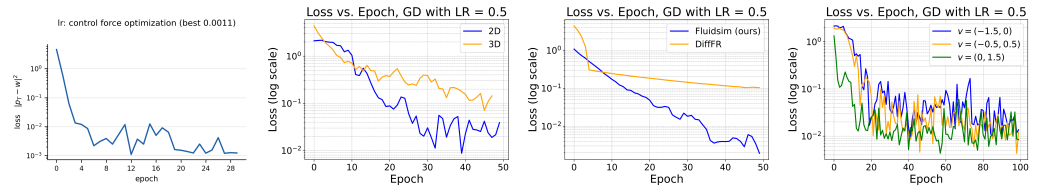
Dam Break. We perform this experiment in both 2D and 3D. In 3D, we set up a box domain of size $3.5 \times 2 \times 5$ containing $70 \times 40 \times 100$ grid cells of size 0.05. A ball of relative density 0.3 is thrown from $(1.7, 1.4, 1.5)$ against the wall of water at the start of the simulation. The objective is to learn the initial velocity so that it reaches $(2.4, 0.4, 3.5)$. In 2D, domain size is 3.5×2 with 70×40 grid cells of side length 0.05. The ball starts from $(1.7, 1.4)$ and the target is at $(1.75, 0.5)$. The simulations are performed for 100 frames ($\Delta t = 0.01$). We use gradient descent with fixed LR of 0.5 for this experiment. Figure 5 shows that our simulator learns the optimum initial velocity in both cases. Figure 4b shows the loss trends.

Block Drop. In this scenario, the objective is to optimize the initial velocity of a block so that it reaches a target position and orientation. The box domain has size $3.5 \times 1.5 \times 2$, discretized into $39 \times 15 \times 24$ grid cells. The block, with side length 2 times the grid cell size of 0.1 and density 2, is initially dropped with velocity $(1, -1, 1)$ onto the liquid covering half of the domain. The time step of the simulation is 5×10^{-3} and it is performed for 50 frames and 50 epochs. We use gradient



(a) Control Loss (b) FV = 20, H = 5 (c) FV = 30, H = 5 (d) FV = 20, H = 10

Figure 3: **Optimal Control. (a)** shows training loss v/s epoch, where FV is tangential fluid velocity and H is horizon length. **(b), (c)** and **(d)** show rigid body trajectory. Fluid is shown in light blue, with dark blue domain boundaries. Fluid motion is shown by light grey lines. Dark blue circle at the centre represents initial rigid body position. Target locations are denoted by stars, with color matching the corresponding trajectory. Colorbar indicates how close a trajectory reaches the target.



(a) Blood Vessel (b) Dam Break (c) Block Drop (d) Sensitivity Analysis

Figure 4: **Loss Curves.** We demonstrate that the gradients calculated by our solver are able to quickly decrease the loss in our experiments. We also compare our simulation against the particle based DiffFR and explore the sensitivity to different initial conditions. DRiFT remains robust and stable under different initial conditions. DRiFT also runs and converges much faster than DiffFR.

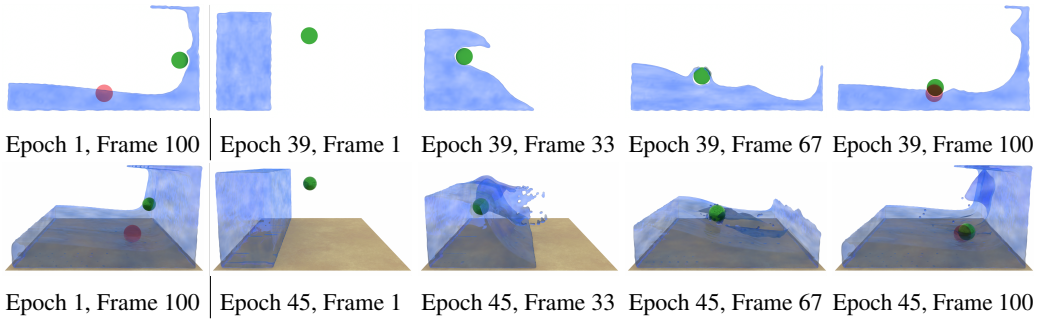


Figure 5: **Dam Break.** We optimize the initial velocity of ball(green) so that it reaches the target(red). Epoch 1 starts with a guess, by Epoch 39(2D) and Epoch 45(3D), the ball is quite close to target.

descent with a fixed LR of 0.5 for this experiment. Figure 6 shows that our simulator learns the optimum initial velocity. Figure 4c shows the loss trends.

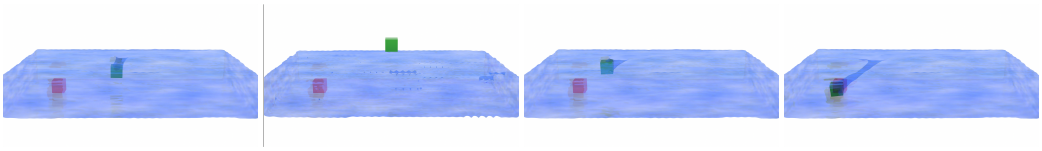


Figure 6: **Block Drop.** By Epoch 49, our simulator correctly optimizes the initial velocity which sends the rigid block (green) to reach the target position (red).

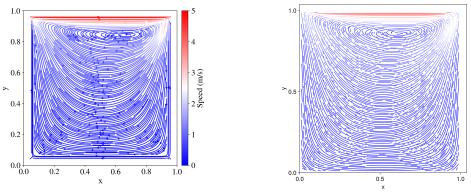
6.3 Analysis and Comparisons

Gradient Sensitivity Analysis. Our simulator is robust to the choice of initial state, and we test this using 2D dam break scenario set up earlier. The ball starts with three different initial velocities in separate runs. The simulations are allowed to run for 100 frames ($\Delta t = 0.01$) and 100 epochs. Figure 4d shows that despite the optimization trajectories being different, the simulator eventually reaches the optima, regardless of the initial velocity guess. Additionally we test the stability of our simulation in Appendix K and the stability of the optimization trajectory in Appendix J. Figure 4a shows the optimization trajectory.

Gradient Accuracy Analysis. We compare the accuracy of our computed gradient with three baselines: centered finite differences (FD), Enzyme AD and PyTorch AD on all demos of Section 6.2, as well as a few additional high-resolution variants of 2D Dam Break. To compute FD, we perturb the initial rigid-body velocity $\boldsymbol{v}_0$ by $\delta = 10^{-5}$, one component at a time, in positive and negative directions to obtain $\boldsymbol{v}_1$ and $\boldsymbol{v}_{-1}$, and obtain the quantity $(2\delta)^{-1}[\mathcal{L}(\Phi^n(\boldsymbol{v}_1)) - \mathcal{L}(\Phi^n(\boldsymbol{v}_{-1}))]$. The results are shown in Table 1. At lower resolutions, our computed gradient matches all the baselines exactly, with a maximum % relative error of $O(10^{-2})$. At higher resolutions, both Enzyme and PyTorch run out of allocated memory budget of 32 GB, while our method succesfully computes the gradient, which almost exactly matches the finite difference gradient.

Experiment	Grid Size	Accuracy Comparison of DRiFT		
		FD	Enzyme	PyTorch
Rigid Drop 2D	40×40	$O(10^{-3})$	$O(10^{-5})$	$O(10^{-5})$
Rigid Drop 3D	$39 \times 15 \times 24$	$O(10^{-2})$	$O(10^{-3})$	$O(10^{-3})$
Dam Break 2D	70×40	$O(10^{-6})$	$O(10^{-7})$	$O(10^{-7})$
	140×80	$O(10^{-7})$	$O(10^{-8})$	$O(10^{-7})$
	280×160	$O(10^{-3})$	-	-
	560×320	$O(10^{-6})$	-	-
Dam Break 3D	$70 \times 40 \times 100$	$O(10^{-6})$	-	-

Table 1: **Gradient Accuracy Analysis.** This table contains the % relative error of our computed gradient when compared with finite difference (FD), Enzyme AD and PyTorch AD in order notation. **For lower resolutions, our gradient matches almost exactly with all baselines**, while **for higher resolutions, our gradient matches the FD gradient exactly, while PyTorch and Enzyme run out of memory**.



DRiFT Aquarium

Figure 7: **Lid-driven Cavity.** We compare the velocity field of our method DRiFT (left) with Aquarium (right), at time $t = 1s$ (50 steps of size $0.02s$). The velocity field is shown as streamlines with arrows showing the direction, and color showing the magnitude. Our result not only obeys the boundary conditions ($u = 5m/s$ on North boundary, zero everywhere else) but matches with Aquarium.

Runtime Analysis: In this section we compare the runtimes of our adjoint based gradient computation method with that of Enzyme AD and PyTorch AD. The runtime numbers are in Table 2. Enzyme is written in C++, so it shares the forward pass with DRiFT, while on the other hand, we recreate a duplicate implementation of our simulator backend in Python for PyTorch to automatically differentiate, so there is some variance of forward pass runtimes between our method and PyTorch. At lower resolutions, our method maintains a significant speedup over Enzyme, while only a modest speedup over PyTorch, while at higher resolutions, Enzyme and PyTorch run out of the allocated memory budget of 32 GB, as mentioned earlier.

Experiment	Grid Size	Forward Pass (Φ) (ms)		Backward Pass (Ψ) (ms)				
		DRiFT	PyTorch	DRiFT	Enzyme	Speedup	PyTorch	Speedup
Rigid Drop 2D	40×40	12.49	19.6	5.31	133.9	25.2×	30.26	5.70×
Rigid Drop 3D	$39 \times 15 \times 24$	658.1	337.9	711.4	4605.6	6.53×	635.5	0.9×
Dam Break 2D	70×40	10.14	17.37	5.95	161.7	27.2×	17.52	2.96×
	140×80	62.41	34.16	19.81	1185.2	59.8×	31.87	1.61×
	280×160	437.7	-	71.87	-	-	-	-
	560×320	3467.1	-	319.36	-	-	-	-
Dam Break 3D	$70 \times 40 \times 100$	3423.5	-	8597.8	-	-	-	-

Table 2: **Runtime comparison between DRiFT and AD methods.** This table lists the per-frame runtimes of the demos in Section 6.2 (averaged across total frame count) for both forward and backward passes, and compares them with the corresponding runtimes of AD methods Enzyme and PyTorch. **At lower resolutions, when all methods execute succesfully, our method greatly surpasses Enzyme, while maintainting only a modest speedup over PyTorch. At higher resolutions, both Enzyme and PyTorch run out of memory and fail to execute beyond a few frames**.

Comparison with Aquarium: In this section, we validate our simulator by comparing its steady state output with Aquarium Lee et al. (2023) on benchmark scenarios:

1. **Lid-driven cavity:** The domain is a 1m square discretized into 50×50 grid cells of side length 0.02. The velocity field is constrained to have zero velocity on all boundaries, except the North boundary, which is fixed at +5m/s in +X direction. Figure 7 shows the results.
2. **Freestream cylinder:** The domain is a 4m square discretized into 100×100 grid cells of side length 0.04. The velocity field is constrained to have a fixed non-zero velocity u_∞ on all boundaries, its value dependent on the Reynolds number Re of the flow. The fluid passes through a static cylindrical obstacle of radius $0.1m$ at $(1m, 2m)$. Figure 8 shows the results.

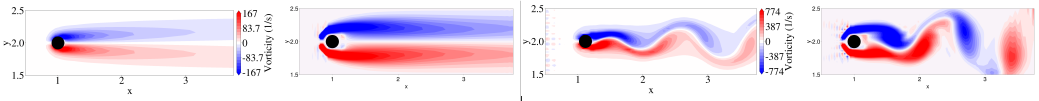


Figure 8: **Freestream cylinder.** We compare the vorticity field (curl of the velocity field i.e. $\nabla \times \boldsymbol{u}$) of our method DRiFT with Aquarium, at $t = 2.5s$ (2500 steps of size 0.001), at two different Reynolds numbers Re = 40 ($u_\infty = 20m/s$) and Re = 100 ($u_\infty = 50m/s$). The vorticity pattern given by our method is similar to that of Aquarium, especially the vortex-shedding at Re = 100.

7 Conclusions, Limitations and Future Work

The solutions to the equations that govern physical simulations are incredibly high dimensional and represent complex interactions. By incorporating gradient information through adjoint simulation, *accurate gradients through monolithically coupled fluid/rigid constraints enable substantially more sample-efficient optimization and control than treating the simulator as a black-box environment.* The derivation of the backward pass is non-trivial, and like the forward pass, is prone to numerical errors depending on the method of simulation used. In this work, we have proposed, designed and implemented an efficient discrete adjoint, DRiFT, for strongly two-way coupled Eulerian fluid–rigid-body dynamics, providing accurate and stable gradients efficiently.

However, our method has some limitations and it can be further improved. Our prototype simulation is grid-based, and accuracy is proportional to the resolution of the grid. If higher resolutions are required only at specific parts of the domain (e.g. near rigid bodies), using uniformly high resolution is inefficient. Differentiable hybrid methods, which can leverage adaptivity in form of multi-resolution grids or through particle-grid operations like MPM, can provide superior accuracy with potential trade-off in complexity, stability, or computation. Secondly, our method can be made more memory efficient by implementing checkpointing, which avoids storing the intermediate results of the entire forward pass, at the cost of runtime. Refer to Appendix G for details. Finally, we plan to provide a comprehensive benchmark with learning baselines and diverse solid-fluid coupling tasks.

AI USE STATEMENT

We used AI to generate the experiment scripts to obtain the runtime numbers of Tables 2 and **??**. All generated experiment scripts were thoroughly checked and validated. Everything else in the paper, including conceptualization, design and implementation of the simulator, experimentation and rendering, and finally writing of the paper was done by the authors without any AI assistance.

ETHICS STATEMENT

We acknowledge that we have read the ICLR Code of Ethics and are committed to adhere to it.

REPRODUCIBILITY STATEMENT

In this work, we have made every possible effort to make our claims and results reproducible. First of all, we provide a detailed derivation of our gradient computation method in Appendices B and D. Then, we provide an overview of our complete algorithm through an explanatory diagram (Figure 9) and pseudocode (Appendix A). Additionally, for all our experiments in Section 6.3, we provide settings and parameters required to replicate the results. Finally, we provide our code in the supplementary for direct reproducibility.

REFERENCES

C. Banerjee, Kien Nguyen, C. Fookes, and M. Raissi. A survey on physics informed reinforcement learning: Review and open problems. *Expert Syst. Appl.*, 287:128166, 2023. URL https://api.semanticscholar.org/CorpusId:261530044.

Christopher Batty. Fluidrigidcoupling2d. https://github.com/christopherbatty/FluidRigidCoupling2D, 2016.

Christopher Batty, Florence Bertails, and Robert Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph.*, 26(3):100–es, jul 2007.

Atılım Günes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *J. Mach. Learn. Res.*, 18(1):5595–5637, January 2017. ISSN 1532-4435.

Jannis Becktepe, Aleksandra Franz, Nils Thuerey, and Sebastian Peitz. Plug-and-play benchmarking of reinforcement learning algorithms for large-scale flow control. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ilEooI8fK1.

Robert Bridson, Ronald Fedkiw, and Matthias Müller-Fischer. Fluid simulation: Siggraph 2006 course notes (fedkiw and muller-fischer presenation videos are available from the citation page). In *ACM SIGGRAPH 2006 Courses*, 2006.

Junyi Cao, Yian Wang, Ziyan Xiong, Chunru Lin, Zhehuan Chen, and Chuang Gan. Dlo-lab: Benchmarking deformable linear object manipulations with differentiable physics. In *International Conference on Machine Learning (ICML)*, 2026.

Siwei Chen*, Yiqing Xu*, Cunjun Yu*, Linfeng Li, Xiao Ma, Zhongwen Xu, and David Hsu. Daxbench: Benchmarking deformable object manipulation with differentiable physics. In *ICLR*, 2023.

M. Decroocq, C. Frindel, P. Rougé, M. Ohta, and G. Lavoué. Modeling and hexahedral meshing of cerebral arterial networks from centerlines. *Medical image analysis, 89, 102912.*

Aleksandra Franz, Hao Wei, Luca Guastoni, and Nils Thuerey. Pict–a differentiable, gpu-accelerated multi-block piso solver for simulation-coupled learning tasks in fluid dynamics. *Journal of Computational Physics*, 544:114433, 2026. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2025.114433. URL https://www.sciencedirect.com/science/article/pii/S0021999125007156.

Frederic Gibou, Ronald P. Fedkiw, Li-Tien Cheng, and Myungjoo Kang. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002.

Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks, 2019. URL https://arxiv.org/abs/1906.01563.

Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

David Ha and Jürgen Schmidhuber. World models. 2018. doi: 10.5281/ZENODO.1207631. URL https://zenodo.org/record/1207631.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. URL https://arxiv.org/abs/2301.04104.

Francis H. Harlow and J. Eddie Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *The Physics of Fluids*, 8(12):2182–2189, 12 1965.

Eric Heiden, David Millard, Hejia Zhang, and Gaurav S. Sukhatme. Interactive differentiable simulation, 2020. URL https://arxiv.org/abs/1905.10706.

Philipp Holl and Nils Thuerey. Phiflow: Differentiable simulations for pytorch, tensorflow and jax. In *International Conference on Machine Learning*. PMLR, 2024.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation, 2020. URL https://arxiv.org/abs/1910.00935.

Zhiao Huang, Yuanming Hu, Tao Du, Siyuan Zhou, Hao Su, Joshua B. Tenenbaum, and Chuang Gan. PlasticineLab: A soft-body manipulation benchmark with differentiable physics. In *ICLR*, 2021.

John Ingraham, Adam Riesselman, Chris Sander, and Debora Marks. Learning protein structure with a differentiable simulator. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Byg3y3C9Km.

Alec Jacobson and Daniele Panozzo. libigl: prototyping geometry processing research in c++. In *SIGGRAPH Asia 2017 Courses*, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450354035. doi: 10.1145/3134472.3134497. URL https://doi.org/10.1145/3134472.3134497.

Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between c++11 and python, 2017. https://github.com/pybind/pybind11.

Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342896. doi: 10.1145/2897826.2927348. URL https://doi.org/10.1145/2897826.2927348.

Bingyi Kang, Yang Yue, Rui Lu, Zhijie Lin, Yang Zhao, Kaixin Wang, Gao Huang, and Jiashi Feng. How far is video generation from world model: A physical law perspective, 2025. URL https://arxiv.org/abs/2411.02385.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

James Kotary, Jacob K. Christopher, M. H. Dinh, and Ferdinando Fioretto. Analyzing and enhancing the backward-pass convergence of unrolled optimization. *ArXiv*, abs/2312.17394, 2023. URL https://api.semanticscholar.org/CorpusId:266691011.

Jeong Hun Lee, Mike Y. Michelis, Robert Katzschmann, and Zachary Manchester. Aquarium: A fully differentiable fluid-structure interaction solver for robotics applications, 2023. URL https://arxiv.org/abs/2301.07028.

Yifei Li, Yuchen Sun, Pingchuan Ma, Eftychios Sifakis, Tao Du, Bo Zhu, and Wojciech Matusik. Neuralfluid: Neural fluidic system design and control with differentiable simulation, 2024. URL https://arxiv.org/abs/2405.14903.

Zhehao Li, Qingyu Xu, Xiaohan Ye, Bo Ren, and Ligang Liu. Difffr: Differentiable sph-based fluid-rigid coupling for rigid body control. *ACM Trans. Graph.*, 42(6), dec 2023.

Zhiqi Li, Jinjin He, Barnabás Börcsök, Taiyuan Zhang, Duowen Chen, Tao Du, Ming C. Lin, Greg Turk, and Bo Zhu. An adjoint method for differentiable fluid simulation on flow maps. In *ACM SIGGRAPH Asia 2025 (Conference Track)*, 2025.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021. URL https://arxiv.org/abs/2010.08895.

J.L. Lions. *Optimal Control of Systems Governed by Partial Differential Equations:*. Springer Berlin, Heidelberg, 1971.

Michael Lutter, Johannes Silberbauer, Joe Watson, and Jan Peters. Differentiable physics models for real-world offline model-based reinforcement learning, 2020. URL https://arxiv.org/abs/2011.01734.

Fabian Löschner, Timna Böttcher, Stefan Rhys Jeske, and Jan Bender. Weighted Laplacian Smoothing for Surface Reconstruction of Particle-based Fluids. In *Vision, Modeling, and Visualization*. The Eurographics Association, 2023. doi: 10.2312/vmv.20231245.

Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC).

Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23(3):449–456, aug 2004.

Miguel Angel Zamora Mora, Momchil Peychev, Sehoon Ha, Martin Vechev, and Stelian Coros. Pods: Policy optimization via differentiable simulation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7805–7817, 18–24 Jul 2021.

William Moses and Valentin Churavy. Instead of rewriting foreign code for machine learning, automatically synthesize fast gradients. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 12472–12485. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/9332c513ef44b682e9347822c2e457ac-Paper.pdf.

Elvis Nava, John Z. Zhang, Mike Y. Michelis, Tao Du, Pingchuan Ma, Benjamin F. Grewe, Wojciech Matusik, and Robert K. Katzschmann. Fast aquatic swimmer optimization with differentiable projective dynamics and neural network hydrodynamic models, 2022.

Rhys Newbury, Jack Collins, Kerry He, Jiahe Pan, Ingmar Posner, David Howard, and Akansel Cosgun. A review of differentiable simulators, 2024. URL https://arxiv.org/abs/2407.05560.

Roussel Desmond Nzoyem Ngueguin, David A.W. Barton, and Tom Deakin. A comparison of mesh-free differentiable programming and data-driven strategies for optimal control under pde constraints. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*, pp. 21–28. ACM, November 2023. doi: 10.1145/3624062.3626078. URL http://dx.doi.org/10.1145/3624062.3626078.

Stanley Osher and Ronald P. Fedkiw. Level set methods: An overview and some recent results. *Journal of Computational Physics*, 169(2):463–502, 2001. ISSN 0021-9991. doi: https://doi.org/10.1006/jcph.2000.6636. URL https://www.sciencedirect.com/science/article/pii/S0021999100966361.

Zherong Pan and Dinesh Manocha. Efficient solver for spacetime control of smoke. *ACM Trans. Graph.*, 36(5), July 2017. ISSN 0730-0301. doi: 10.1145/3016963. URL https://doi.org/10.1145/3016963.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

Charles S. Peskin. The immersed boundary method. *Acta Numerica*, 11:479–517, 2002. doi: 10.1017/S0962492902000077.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Brener Ramos, Felix Trost, and Nils Thuerey. Control of two-way coupled fluid systems with differentiable solvers, 2022.

Magí Romanyà-Serrasolsas, Juan J. Casafranca, and Miguel A. Otaduy. Painless differentiable rotation dynamics. *ACM Trans. Graph.*, 44(4), July 2025. ISSN 0730-0301. doi: 10.1145/3730944. URL https://doi.org/10.1145/3730944.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. ICML'20. JMLR.org, 2020.

Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2), 2015. ISSN 0098-3500. doi: 10.1145/2629697. URL https://doi.org/10.1145/2629697.

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

Tetsuya Takahashi, Junbang Liang, Yi-Ling Qiao, and Ming C. Lin. Differentiable fluids with solid coupling for learning and control. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):6138–6146, May 2021.

Jingwei Tang, Vinicius C. Azevedo, Guillaume Cordonnier, and Barbara Solenthaler. Honey, i shrunk the domain: Frequency-aware force field reduction for efficient fluids optimization. *Computer Graphics Forum*, 40(2):339–353, 2021. doi: https://doi.org/10.1111/cgf.142637. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.142637.

Artur P. Toshev, Harish Ramachandran, Jonas A. Erbesdobler, Gianluca Galletti, Johannes Brandstetter, and Nikolaus A. Adams. Jax-sph: A differentiable smoothed particle hydrodynamics framework, 2024. URL https://arxiv.org/abs/2403.04750.

Kun Wang, Mridul Aanjaneya, and Kostas E. Bekris. A first principles approach for data-efficient system identification of spring-rod systems via differentiable physics engines. In *L4DC*, 2020.

Zhou Xian, Bo Zhu, Zhenjia Xu, Hsiao-Yu Tung, Antonio Torralba, Katerina Fragkiadaki, and Chuang Gan. Fluidlab: A differentiable environment for benchmarking complex fluid manipulation. In *International Conference on Learning Representations*, 2023.

Eliot Xing, Vernon Luk, and Jean Oh. Stabilizing reinforcement learning in differentiable multiphysics simulation, 2025. URL https://arxiv.org/abs/2412.12089.

Jie Xu, Viktor Makoviychuk, Yashraj Narang, Fabio Ramos, Wojciech Matusik, Animesh Garg, and Miles Macklin. Accelerated policy learning with parallel differentiable simulation, 2022.

A SIMULATOR PSEUDO CODE

Algorithm 1 Differentiable Simulation

Input: Initial state $\boldsymbol{q}_0 = (\boldsymbol{u}_0, \phi_0, \boldsymbol{v}_0, \boldsymbol{x}_0)$, Control Force Sequence $\boldsymbol{c}$, Number of frames f, Time step size Δt, Loss function $\mathcal{L}$

$\boldsymbol{q} \leftarrow \boldsymbol{q}_0$
for $t = 1$ **to** f **do**
  Forward Simulation: $\boldsymbol{q}, \widetilde{\boldsymbol{q}} \leftarrow \Phi(\boldsymbol{q}, \boldsymbol{c}[t], \Delta t)$ ▷ Algorithm 2
  Store results: $\widetilde{\boldsymbol{q}}$
end for
$\boldsymbol{Q} \leftarrow$ getLossDerivative($\mathcal{L}, \boldsymbol{q}$)
$\boldsymbol{C} \leftarrow$ Zeros$(f, 3)$
for $t = f$ **to** 1 **do**
  Adjoint Simulation: $\boldsymbol{Q}, \boldsymbol{C}[t] \leftarrow \Psi(\boldsymbol{Q}, \widetilde{\boldsymbol{q}}[t], \Delta t)$ ▷ Algorithm 3
end for

Output: Final adjoint state, $\boldsymbol{Q} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0}$

Algorithm 2 Forward Simulation (Φ), One Time Step

Input: State $\boldsymbol{q}_t$, Control Force $\boldsymbol{c}_t$, Time step size Δt

$\boldsymbol{q}_{t+\frac{1}{5}} \leftarrow$ advect$(\boldsymbol{q}_t, \Delta t)$ ▷ Eqn. 3
$\boldsymbol{q}_{t+\frac{2}{5}} \leftarrow$ externalForce$(\boldsymbol{q}_{t+\frac{1}{5}}, \Delta t)$
$\boldsymbol{q}_{t+\frac{3}{5}} \leftarrow$ solvePressure$(\boldsymbol{q}_{t+\frac{2}{5}}, \Delta t)$ ▷ Eqn. 6
$\boldsymbol{q}_{t+\frac{4}{5}} \leftarrow$ velocityCorrection$(\boldsymbol{q}_{t+\frac{3}{5}}, \Delta t)$ ▷ Eqn. 12
$\boldsymbol{q}_{t+1} \leftarrow$ updateRigidBody$(\boldsymbol{q}_{t+\frac{4}{5}}, \Delta t)$ ▷ Eqn. 2

Output: State $\boldsymbol{q}_{t+1}$

Algorithm 3 Adjoint Simulation (Ψ), One Time Step

Input: Adjoint state $\boldsymbol{Q}_{t+1}$, Storage state $\widetilde{\boldsymbol{q}}_t$, Time step size Δt

$\boldsymbol{Q}_{t+\frac{4}{5}}, \boldsymbol{C} \leftarrow$ adjointUpdateRigidBody$(\boldsymbol{Q}_{t+1}, \widetilde{\boldsymbol{q}}_t, \Delta t)$ ▷ Section 4.3
$\boldsymbol{Q}_{t+\frac{3}{5}} \leftarrow$ adjointVelocityCorrection$(\boldsymbol{Q}_{t+\frac{4}{5}}, \widetilde{\boldsymbol{q}}_t, \Delta t)$ ▷ Eqn. 12
$\boldsymbol{Q}_{t+\frac{2}{5}} \leftarrow$ adjointSolvePressure$(\boldsymbol{Q}_{t+\frac{3}{5}}, \widetilde{\boldsymbol{q}}_t, \Delta t)$ ▷ Eqn. 7
$\boldsymbol{Q}_t \leftarrow$ adjointAdvectVelocity$(\boldsymbol{Q}_{t+\frac{2}{5}}, \widetilde{\boldsymbol{q}}_t, \Delta t)$ ▷ Eqn. 3

Output: Adjoint state $\boldsymbol{Q}_t$, Control Force Derivative $\boldsymbol{C}$

The storage state $\widetilde{\boldsymbol{q}}$ is used for sharing data between the forward and adjoint passes to avoid recomputation. In our implementation, in addition to storing all the intermediate states q generated in all sub-stages of the forward pass, we also store SDFs of liquid (ϕ_F) and solid (ϕ_S), their volume fractions ($\mathbf{VF}_F$, $\mathbf{VF}_S$) and the computed pressure $\mathbf{p}$.

B DERIVATION: GRADIENTS OF PRESSURE SOLVE

The pressure solve or projection step transforms the combined input state $(\boldsymbol{u}_t, \phi_t, \boldsymbol{v}_t, \boldsymbol{x}_t)$ into a combined output velocity $(\boldsymbol{u}_{t+1}, \boldsymbol{v}_{t+1})$ that obeys the physical constraints highlighted in eqn. (2). Starting here, we wish to compute the downstream gradients $\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t}, \frac{\partial \mathcal{L}}{\partial \phi_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t}\right)$, using the upstream gradients $\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}}, \frac{\partial \mathcal{L}}{\partial \phi_{t+1}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{t+1}}\right)$. Let's say we require $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}$, where $\boldsymbol{q} = \boldsymbol{u}_t, \phi_t, \boldsymbol{v}_t, \boldsymbol{x}_t$.

Then, on differentiating the linear system of eqn. (2),

$$\begin{aligned}
&\mathbf{A}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{q}}\right)+\frac{\partial \mathbf{A}}{\partial \boldsymbol{q}}\mathbf{p}=\frac{\partial \mathbf{b}}{\partial \boldsymbol{q}}\\
\rightarrow &\mathbf{A}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{u}_t}\right)=\frac{1}{\rho}\mathbf{G}^T\mathbf{M}_F,\ \ \mathbf{A}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{v}_t}\right)=-\mathbf{J}^T\\
\rightarrow &\mathbf{A}\left(\frac{\partial \mathbf{p}}{\partial \phi_t}\right)=-\frac{\Delta t}{\rho^2}\mathbf{G}^T\mathbf{M}_F\left(\frac{\partial \mathbf{G}}{\partial \phi_t}\right)\mathbf{p}\\
\rightarrow &\mathbf{A}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{x}_t}\right)=\frac{1}{\rho}\mathbf{G}^T\left(\frac{\partial \mathbf{M}_F}{\partial \boldsymbol{x}_t}\right)\boldsymbol{u}_t-\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)^T\boldsymbol{v_t}-\Delta t\ \mathbf{J}^T\mathbf{M}_S^{-1}\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{p}\\
&\qquad\qquad-\Delta t\ \left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{M}_S^{-1}\mathbf{J}\mathbf{p}-\Delta t\ \mathbf{J}^T\left(\frac{\partial \mathbf{M}_S^{-1}}{\partial \boldsymbol{x}_t}\right)\mathbf{J}\mathbf{p}-\frac{\Delta t}{\rho^2}\mathbf{G}^T\left(\frac{\partial \mathbf{M}_F}{\partial \boldsymbol{x}_t}\right)\mathbf{G}\mathbf{p}
\end{aligned}$$

Now, differentiating the velocity updates of eqn. (2),

$$\begin{aligned}
\frac{\partial \boldsymbol{u}_{t+1}}{\partial \boldsymbol{q}}&=\frac{\partial \boldsymbol{u}_t}{\partial \boldsymbol{q}}-\frac{\Delta t}{\rho}\mathbf{G}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{q}}\right),\\
\frac{\partial \boldsymbol{v}_{t+1}}{\partial \boldsymbol{q}}&=\frac{\partial \boldsymbol{v}_t}{\partial \boldsymbol{q}}+\Delta t\mathbf{M}_S^{-1}\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{q}}\right)\mathbf{p}+\Delta t\mathbf{M}_S^{-1}\mathbf{J}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{q}}\right)+\Delta t\left(\frac{\partial \mathbf{M}_S^{-1}}{\partial \boldsymbol{q}}\right)\mathbf{J}\mathbf{p}
\end{aligned}$$

Now, we find $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}$ in terms of $\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{t+1}}\right)$, using the principle of total derivative,

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}=\left(\frac{\partial \boldsymbol{u}_{t+1}}{\partial \boldsymbol{q}}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}+\left(\frac{\partial \boldsymbol{v}_{t+1}}{\partial \boldsymbol{q}}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}+\left(\frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{q}}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{x_{t+1}}}\\
\rightarrow &\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t}=\left(\frac{\partial \boldsymbol{u}_{t+1}}{\partial \boldsymbol{u}_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}+\left(\frac{\partial \boldsymbol{v}_{t+1}}{\partial \boldsymbol{u}_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}+\cancelto{0}{\left(\frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{u}_t}\right)^T}\frac{\partial \mathcal{L}}{\partial \boldsymbol{x_{t+1}}}\\
&\qquad=\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}+\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{u}_t}\right)^T\left[\Delta t\mathbf{J}^T\mathbf{M}_S^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}-\frac{\Delta t}{\rho}\mathbf{G}^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}\right]\\
\rightarrow &\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t}=\left(\frac{\partial \boldsymbol{u}_{t+1}}{\partial \boldsymbol{v}_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}+\left(\frac{\partial \boldsymbol{v}_{t+1}}{\partial \boldsymbol{v}_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}+\cancelto{0}{\left(\frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{v}_t}\right)^T}\frac{\partial \mathcal{L}}{\partial \boldsymbol{x_{t+1}}}\\
&\qquad=\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}+\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{v}_t}\right)^T\left[\Delta t\mathbf{J}^T\mathbf{M}_S^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}-\frac{\Delta t}{\rho}\mathbf{G}^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}\right]\\
\rightarrow &\frac{\partial \mathcal{L}}{\partial \phi_t}=\left(\frac{\partial \boldsymbol{u}_{t+1}}{\partial \phi_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}+\left(\frac{\partial \boldsymbol{v}_{t+1}}{\partial \phi_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}+\cancelto{I}{\left(\frac{\partial \phi_{t+1}}{\partial \phi_t}\right)^T}\frac{\partial \mathcal{L}}{\partial \phi_{t+1}}\\
&\qquad=\frac{\partial \mathcal{L}}{\partial \phi_{t+1}}-\frac{\Delta t}{\rho}\left[\left(\frac{\partial \mathbf{G}}{\partial \phi_t}\right)\mathbf{p}\right]^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}}+\left(\frac{\partial \mathbf{p}}{\partial \phi_t}\right)^T\left[\Delta t\mathbf{J}^T\mathbf{M}_S^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}-\frac{\Delta t}{\rho}\mathbf{G}^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}\right]\\
\rightarrow &\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t}=\left(\frac{\partial \boldsymbol{u}_{t+1}}{\partial \boldsymbol{x}_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}+\left(\frac{\partial \boldsymbol{v}_{t+1}}{\partial \boldsymbol{x}_t}\right)^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}+\cancelto{I}{\left(\frac{\partial \boldsymbol{x}_{t+1}}{\partial \boldsymbol{x}_t}\right)^T}\frac{\partial \mathcal{L}}{\partial \boldsymbol{x_{t+1}}}\\
&\qquad=\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{t+1}}+\Delta t\ \left[\mathbf{M}_S^{-1}\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{p}\right]^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}+\Delta t\left[\left(\frac{\partial \mathbf{M}_S^{-1}}{\partial \boldsymbol{x}_t}\right)\mathbf{J}\mathbf{p}\right]^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}\\
&\qquad+\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{x}_t}\right)^T\left[\Delta t\mathbf{J}^T\mathbf{M}_S^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}}-\frac{\Delta t}{\rho}\mathbf{G}^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}\right]
\end{aligned}$$

For any vector $\mathbf{d}$, the quantity $\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{q}}\right)^T\mathbf{d}$ given $\mathbf{A}\left(\frac{\partial \mathbf{p}}{\partial \boldsymbol{q}}\right)=\mathbf{B}$ is equivalent to $\mathbf{B}^T\mathbf{s}$ given $\mathbf{A}^T\mathbf{s}=\mathbf{d}$. This is the adjoint method, first introduced for differentiable fluid simulation by McNamara et al.

(2004). Combined with the fact that $\mathbf{A}$ is symmetric:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}} + \frac{1}{\rho}\mathbf{M}_F\mathbf{G}\mathbf{s},$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}} - \mathbf{J}\mathbf{s}$$

$$\frac{\partial \mathcal{L}}{\partial \phi_t} = \frac{\partial \mathcal{L}}{\partial \phi_{t+1}} - \frac{\Delta t}{\rho}\left[\left(\frac{\partial \mathbf{G}}{\partial \phi_t}\right)\mathbf{p}\right]^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{t+1}} + \Delta t\left[\mathbf{M}_S^{-1}\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{p} + \left(\frac{\partial \mathbf{M}_S^{-1}}{\partial \boldsymbol{x}_t}\right)\mathbf{J}\mathbf{p}\right]^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t} - \boldsymbol{v}_{t+1}^T\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{s}$$
$$+ \frac{1}{\rho}\boldsymbol{u}_{t+1}^T\left(\frac{\partial \mathbf{M}_F}{\partial \boldsymbol{x}_t}\right)\mathbf{G}\mathbf{s}$$

$$\text{where } \mathbf{A}\mathbf{s} = \Delta t\mathbf{J}^T\mathbf{M}_S^{-1}\frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}} - \frac{\Delta t}{\rho}\mathbf{G}^T\frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}$$

C Detailed Simulator Flowchart

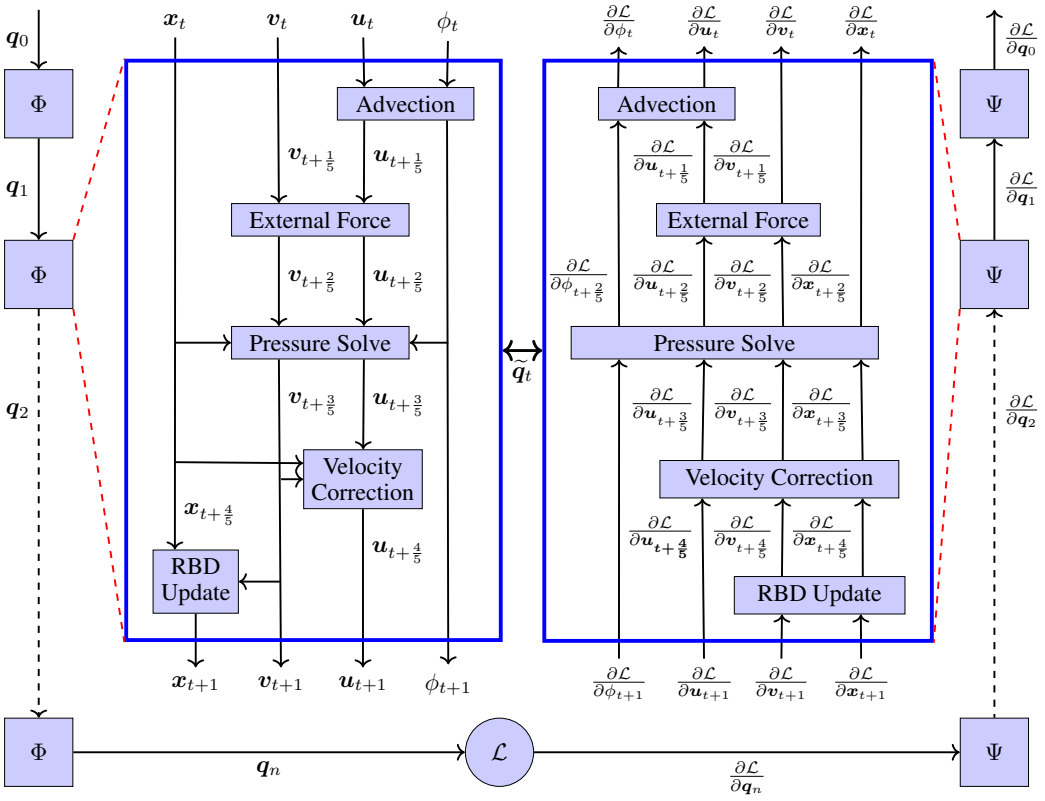


Figure 9: **Algorithm Flow Diagram** shows the forward (left) and backward pass (right) of our differentiable simulator. Each time step acts on the combined solid-fluid state $\boldsymbol{q}$ or the combined adjoint state $\frac{\partial \mathcal{L}}{\partial q}$. The zoomed in diagram shows the sub-stages in one time-step. There is some degree of sharing in each time-step (i.e. $\widetilde{\boldsymbol{q}}_t$) between the passes for efficiency. This framework is general and can solve a variety of optimization and learning problems with accurate fluid-solid coupling.

D Quaternions and their derivatives

We use quaternions for tracking the rotation of 3D rigid bodies. A quaternion $q = w + xi + yj + zk$ is a 4-dimensional imaginary number satisfying $i^2 = j^2 = k^2 = ijk = -1$; we write it compactly

as $q = [w, v]$ where $w \in \mathbb{R}$ is the scalar part and $v = (x, y, z) \in \mathbb{R}^3$ is the vector part. A *unit* quaternion ($\|q\| = 1$) encodes a 3D rotation: a rotation by angle θ about a unit axis $\hat{n}$ is represented as $q = [\cos(\theta/2),\ \hat{n}\sin(\theta/2)]$. Re-normalization after a time step amounts to dividing by $\|q\|$, which is differentiable everywhere. Note that q and $-q$ encode the same physical rotation; we track the sign consistently during integration to avoid spurious sign flips.

The product of two quaternions $p = [p_w, p_v]$ and $q = [q_w, q_v]$, denoted $p \otimes q$, is defined as

$$p \otimes q = [p_w q_w - p_v \cdot q_v,\ p_w q_v + q_w p_v + p_v \times q_v],$$

and represents the composition of the two encoded rotations. This product is associative but not commutative. The time-integration step in Section 4.3 uses the product $[0, \omega] \otimes q$ to map angular velocity ω and current orientation q into the quaternion rate of change, which is used to update the current quaternion. Given angular velocity $\omega = (\omega_x, \omega_y, \omega_z)$ about three principal axes, the quaternion update equation and the corresponding partial derivatives are:

$$\begin{aligned}
\tilde{q}_{t+1} &= q_t + \frac{\Delta t}{2}\left([0, \omega_{t+1}] \otimes q_t\right), \text{ time integration} \\
&= q_t + \frac{\Delta t}{2}[\omega_{t+1} \cdot (q_t)_v, (q_t)_w\ \omega_{t+1} + \omega_{t+1} \times (q_t)_v]
\end{aligned}$$

$$\therefore \frac{\partial \tilde{q}_{t+1}}{\partial q_t} = I_{4\times 4} + \frac{\Delta t}{2}\begin{bmatrix} 0 & -(\omega_{t+1})_x & -(\omega_{t+1})_y & -(\omega_{t+1})_z \\ (\omega_{t+1})_x & 0 & -(\omega_{t+1})_z & (\omega_{t+1})_y \\ (\omega_{t+1})_y & (\omega_{t+1})_z & 0 & -(\omega_{t+1})_x \\ (\omega_{t+1})_z & -(\omega_{t+1})_y & (\omega_{t+1})_x & 0 \end{bmatrix}$$

$$\therefore \frac{\partial \tilde{q}_{t+1}}{\partial \omega_{t+1}} = \frac{\Delta t}{2}\begin{bmatrix} -(q_t)_x & -(q_t)_y & -(q_t)_z \\ (q_t)_w & (q_t)_z & -(q_t)_y \\ -(q_t)_z & (q_t)_w & (q_t)_x \\ (q_t)_y & -(q_t)_x & (q_t)_w \end{bmatrix}$$

followed by the re-normalization step,

$$q_{t+1} = \frac{\tilde{q}_{t+1}}{||\tilde{q}_{t+1}||}, \text{ normalization}$$

$$\therefore \left(\frac{\partial q_{t+1}}{\partial \tilde{q}_{t+1}}\right)_{ij} = \begin{cases} \frac{1-(q_{t+1})_i^2}{||\tilde{q}_{t+1}||} & i = j \\ \frac{-(q_{t+1})_i (q_{t+1})_j}{||\tilde{q}_{t+1}||} & i \neq j \end{cases}$$

Note that while we account for the quaternion re-normalization after each time step in gradient computation, they must also be normalized after the optimizer step, which might cause an increase in objective function value. Strictly speaking, we need to look at constrained updates, which simultaneouly decrease the objective value. But this is out of the scope of our current work.

An alternative approach is to use a different representation of rotation derivatives Romanyà-Serrasolsas et al. (2025).

E Collision Handling for Rigid Bodies

We handle collisions between the rigid bodies and the external boundary using broad-phase collision detection on spherical bounding volumes encapsulating the rigid body, combined with linear velocity reflection for obtaining collision free velocity. If at any time t, a rigid body has a bounding volume $\mathcal{S}(\boldsymbol{x}_t, r)$, with centre $\boldsymbol{x}_t$ and radius r, then it is under collision with the boundary B if $\phi_B(\boldsymbol{x}_t) < r$, in which case, we approximate the contact point to be at $\boldsymbol{c} = \boldsymbol{x}_t - r\nabla\phi_B(\boldsymbol{x}_t)$, and contact normal $\boldsymbol{n} = \boldsymbol{c} - \boldsymbol{x}_t$. And if the pre-collision velocity is $\boldsymbol{v}_t$, then post-collision:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t - (1 + \mathcal{E})(\boldsymbol{v}_t \cdot \hat{\boldsymbol{n}})\hat{\boldsymbol{n}} \tag{10}$$

where $\mathcal{E}$ is the coefficient of restitution. The corresponding adjoint update is:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t} &= (I - (1 + \mathcal{E})\hat{\boldsymbol{n}}\hat{\boldsymbol{n}}^T)\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t} &= (1 + \mathcal{E})\left[\hat{\boldsymbol{n}}^T \boldsymbol{v}_t^T \left(\frac{\partial \hat{\boldsymbol{n}}}{\partial \boldsymbol{x}_t}\right) + \left(\frac{\partial \hat{\boldsymbol{n}}}{\partial \boldsymbol{x}_t}\right)^T (\boldsymbol{v}_t^T \hat{\boldsymbol{n}})\right]\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}
\end{aligned} \tag{11}$$

F FLUID VELOCITY CORRECTION

For efficieny, boundary conditions from equations 4 are not applied to grid cells with no fluid, but these samples may be used during advection in later time steps. These *invalid* velocities must be corrected to make sure that the advected velocity obeys the boundary conditions.

The method of correction depends on where the invalid velocity sample lies. If it lies in the air domain (Ω_A), it can be extrapolated from adjacent valid velocity samples, using Gauss-Seidel iterations. Larger time step means more iterations. In our experiments, 3 iterations were sufficient. If the velocity sample lies inside a rigid body (Ω_S), it is set to the appropriate component of the rigid body velocity at the sample point. In both cases, the corrected velocity field $\boldsymbol{u}_c$ is linear in the combined projected velocity $(\boldsymbol{u}_{t+1}, \boldsymbol{v}_{t+1})$:

$$\boldsymbol{u}_c = \mathbf{C} \begin{bmatrix} \boldsymbol{u}_{t+1} \\ \boldsymbol{v}_{t+1} \end{bmatrix} \implies \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}} \end{bmatrix}^T = \mathbf{C}^T \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_c} \right) \tag{12}$$

where C is the correction matrix.

G COMPUTATIONAL COMPLEXITY ANALYSIS

Suppose the number of grid cells is N, number of marker particles is P and there are M rigid bodies with maximum T triangles each. And let the number of frames in an epoch be F.

G.1 MEMORY

G.1.1 ANALYSIS

- Fluid State, $O(N+P)$: Velocity, $O(N)$ + Pressure, $O(N)$ + SDF (signed distance function), $O(N)$ + Marker Particles, $O(P)$
- Rigid Body State, $O(M+N)$: Velocity, $O(M)$ + Position, $O(M)$ + SDF, $O(N)$
- Combined State, $O(M+N+P)$
- Sparse Gradient Matrix $\mathbf{G}$, $O(N)$
- Solid-Fluid Coupling Matrix $\mathbf{J}$, $O(NM)$
- One iteration(frame) of forward pass, $O(MN+P)$
- One iteration(frame) of backward pass, $O(MN)$: No marker particles in backward pass
- One optimization epoch (forward + backward pass), $O(F \times (M+N))$: Because marker particles and temporary matrices are discarded after every frame

To avoid recomputation and save runtime, we store the entire forward pass, which significantly increases the memory overhead. This is a major limitation of our work. To put this into perspective, the peak memory usage, which occurs at the end of the forward pass, scales as $O(F \times (M+N))$. So, on 32 GB RAM, for a single rigid body with 6 DOFs, our method allows us to simulate about $\frac{1.33 \times 10^8}{F}$ grid cells. Because of this reason, we cannot simulate higher resolution scenarios beyond a certain point for meaningful lengths. For e.g. in Appendix **??**, we could simulate the 1080×640 scenario for only 200 frames.

G.1.2 IMPROVEMENTS

We can save some memory per frame by using sparser $\mathbf{J}$ to bring down its space to $O(N)$, but this is not significant in a longer simulation since it is discarded anyway at the end of the frame. A more effective startegy would be to not store any intermediate frames, but use **checkpointing** for backward pass. This brings down the memory requirements to $O(M+N)$ but increases the computation time of the backward pass by an equivalent factor.

G.2 TIME (UNPARALLELIZED)

G.2.1 ANALYSIS

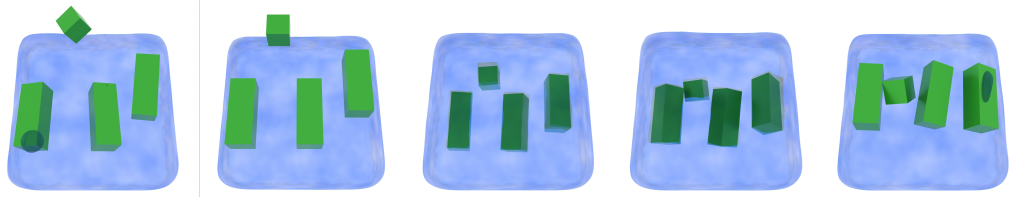
- Advection, $O(N)$
- Particle Advection + Fluid SDF Computation, $O(PMT)$
- Rigid Body Update, $O(M)$
- Rigid Body SDF Computation, $O(MT)$
- Pressure Solve, $O(NM + (N+M)^{1.5})$: Matrix constructions, $O(NM)$ + Linear solve using conjugate gradients, $O((N+M)^{1.5})$
- Velocity Correction, $O(N+M)$: sparse matrix-vector multiplication of size $M+N$
- One iteration(frame) of forward pass, $O(NM + (N+M)^{1.5} + PMT)$
- One iteration(frame) of backward pass, $O(NM + (N+M)^{1.5})$
- One optimization epoch (forward + backward pass), $O(F(NM + (N+M)^{1.5} + PMT))$

G.2.2 IMPROVEMENTS

For a fixed background grid, the above reported times are the best possible times (on a single thread/core) as far as we are aware. In practice, the particle advection (i.e. $O(PMT)$) takes the most time. So, one way is to optimize marker particle advection or employ a grid-based advection scheme like level-set advection. Another way this can be made much faster is if we use **adaptive grids**, which we are currently working on. Adpative grids have finer resolution in areas requiring higher accuracy (i.e. near solid-fluid boundary) and coarser resolution in other macroscopic areas (i.e. fluid regions). Such grids have the potential to be efficient both in space and time. But differentiability is a challenge in adaptive grids because of dynamic size of state variables.

H MORE RESULTS

Text Optimization. We demonstrate our solver's ability to jointly optimize multiple rigid bodies by spelling the word "HI". The simulation domain is $15 \times 15 \times 15$ with 0.1 spacing, and runs for 100 frames ($\Delta t = 0.01$) and 20 epochs. Adam Kingma & Ba (2017) optimizer is used with initial LR = 0.5. The fluid surface starts at height 0.6 meters from the bottom of the domain and all four blocks are dropped from a height of 0.8. Three long blocks measure $1.5 \times 1.5 \times 4.0$ grid cells; the fourth is 1.5^3. The loss function is the mean-squared distance between the observed and desired positions of the block. In the first epoch, where all blocks start with zero velocity, the blocks hit the water surface and bounce indiscriminately; however by the end of training, the blocks submerge themselves and bounce out to form the word. Results are shown in Fig 10 along with the smoothly decreasing loss curve in Fig 11a. Forward pass takes 0.76s per frame and the adjoint pass takes 2.76s per frame.

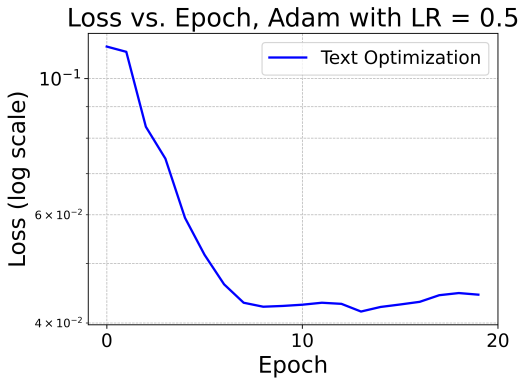


Epoch 1, Frame 100 | Epoch 19, Frame 1 | Epoch 19, Frame 33 | Epoch 19, Frame 66 | Epoch 19, Frame 100

Figure 10: **Text Optimization.** We optimize initial velocity of four blocks to spell the word "HI" after dropping and interacting with fluid.

I COMPARISON WITH DIFFFR

In this section, we compare our method with the SPH-based differentiable simulator DiffFR Li et al. (2023), on the block drop experiment. For our method, each forward pass was around 0.66 seconds,



(a) Text Optimization

Figure 11: **Loss Curve for Text Optimization Demo**

and backward pass about 0.67 seconds, totaling about 22 minutes for 20 epochs, while DiffFR takes about 2 hour, 51 minutes, which **makes our method about 8 times faster**. Figure 6 shows our learned trajectory, while Figure 12 shows that of DiffFR. Both methods converge to a reasonable solution in about 20 epochs as evident in the loss curve in Figure 4c, but our method greatly reduces the computational cost.

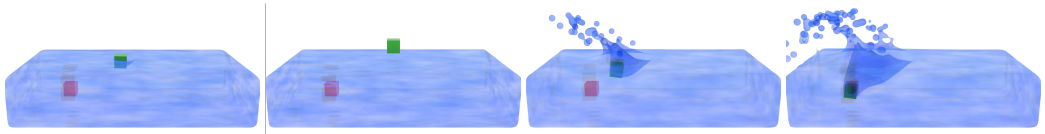


Figure 12: **Comparison with DiffFR.** We simulate the block drop scenario of Figure 6 but with the particle-based differentiable simulator DiffFR (Li et al., 2023). Similar to us, it manages to find an optimal velocity, but **at 8 times our runtime cost**.

J Optimization Stability

In this section, we analyze the stability of our optimization process by performing the experiments in the paper for longer, specifically 10 times the number of epochs in the main paper, for the demos in Section 6.2. The results are shown in Figure 13. For most of the cases, the optimization trajectory is stable. For Dam Break 2D, the optimization trajectory is oscillatory but the loss stays between 10^{-5} and 10^{-1}, with minimum value about 5.3×10^{-6}. For text optimization experiment, the loss decreases on average for the first 100 epochs, then increases slightly and decreases again to attain a minimum value of 5×10^{-3}. We believe the oscillations in the optimization trajectory are more due to the complicated loss landscape in these demos, and the fact that we use gradient descent with a fixed learning rate as optimizer, and less to do with the accuracy of our gradients, which we validated earlier in Section 6.3. We leave the exploration of this loss landscape and the application of better optimizers to future work.

K Simulation Stability

In this section, we test the stability of the our simulator to input perturbations. We take each of the demos of Section 6.2, perturb the initial rigid body position and velocity by varying amounts δ ranging from 10^{-7} to 10^{-3}, and study the relative change in final simulation state $\boldsymbol{q} = (\boldsymbol{u}, \boldsymbol{x}, \boldsymbol{v})$. Figure **??** contains the plots of the instability factor $\frac{\Delta \boldsymbol{q}}{\delta}$, for each of the components.

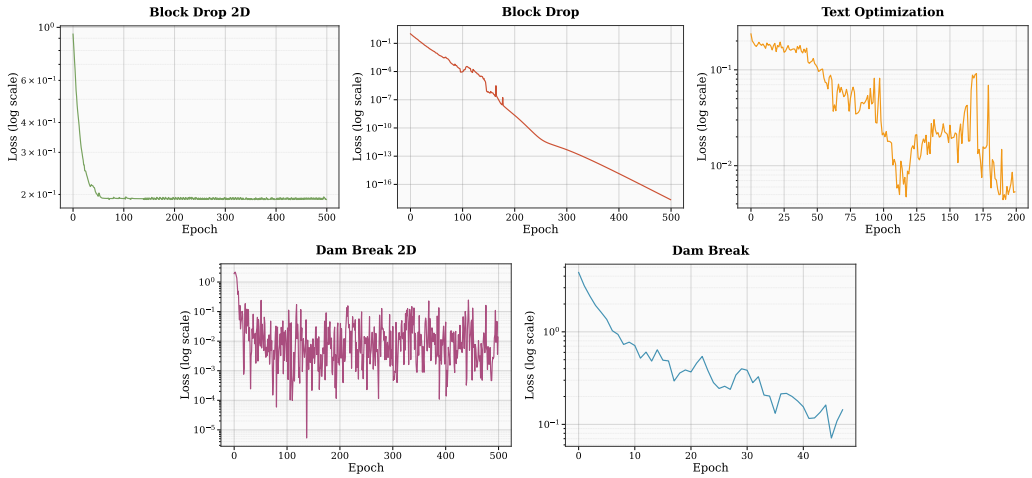


Figure 13: **Optimization Stability.** This experiment extends the demos of Section 6.2 to $10\times$ more epochs than in the main paper to analyze the optimization trajectory. For all the cases, the loss stays low on average. In some cases like Dam Break 2D and Text Optimization, we see oscillations, which we believe might be due to our fixed learning rate optimizer or the complex loss landscape, rather than the accuracy or stability of our solver.

The simplest demo, block drop, which contains a single rigid body interacting with static liquid, has the lowest instability factor which is fairly constant for any value of input perturbation δ. Dam Break scenario, which is more complicated and involves dynamic liquid has a higher instability. Finally, the text optimization scenario has large instability factor at extremely small perturbations. This may be because the perturbed rigid body might be clipping the external boundary or another rigid body, which is not handled by our simulator, atleast not differentiably, at the moment. **Note that these instability plots do not mean that our simulator is unstable, because the actual output perturbations are quite small in value, as shown in Table 3**. It's just that the relative values are large at small input perturbations, which might be because of the minimum inherent inaccuracy in grid-based discretization.

δ / Exp.	10^{-3}			10^{-4}			10^{-5}			10^{-6}			10^{-7}		
	$\Delta\boldsymbol{u}$	$\Delta\boldsymbol{x}$	$\Delta\boldsymbol{v}$	$\Delta\boldsymbol{u}$	$\Delta\boldsymbol{x}$	$\Delta\boldsymbol{v}$	$\Delta\boldsymbol{u}$	$\Delta\boldsymbol{x}$	$\Delta\boldsymbol{v}$	$\Delta\boldsymbol{u}$	$\Delta\boldsymbol{x}$	$\Delta\boldsymbol{v}$	$\Delta\boldsymbol{u}$	$\Delta\boldsymbol{x}$	$\Delta\boldsymbol{v}$
Dam Break 3D	$1.3e-3$	$3.4e-2$	$1.1e-1$	$8.6e-4$	$1.5e-2$	$1.3e-1$	$9.5e-4$	$1.7e-2$	$9.2e-2$	$7.2e-4$	$3.1e-3$	$6.2e-2$	$5.3e-4$	$2.3e-3$	$4.2e-2$
Dam Break 2D	$3.3e-3$	$1.3e-2$	$7.2e-2$	$2.5e-3$	$1.1e-2$	$5.8e-2$	$1.9e-4$	$1.5e-3$	$1.9e-3$	$9.6e-7$	$5.4e-5$	$1.3e-4$	$9.6e-8$	$5.4e-6$	$1.3e-5$
Text Optimization	$1.2e-2$	$8.6e-2$	$7.2e-1$	$9.1e-3$	$4.2e-2$	$4.4e-1$	$3.3e-3$	$6.1e-3$	$9.5e-2$	$1.9e-3$	$2.4e-3$	$4.7e-2$	$7.1e-4$	$9.2e-4$	$1.8e-2$
Block Drop 2D	$8.9e-3$	$4.2e-3$	$5.1e-2$	$5.1e-5$	$3.0e-4$	$3.1e-3$	$1.6e-6$	$4.7e-5$	$4.9e-4$	$1.6e-7$	$4.7e-3$	$4.9e-5$	$1.6e-8$	$4.7e-7$	$4.9e-6$
Block Drop 3D	$5.3e-4$	$3.0e-3$	$6.9e-2$	$9.0e-5$	$2.8e-4$	$9.2e-3$	$7.6e-6$	$2.0e-5$	$1.4e-3$	$5.7e-8$	$1.6e-6$	$3.3e-5$	$5.7e-9$	$1.6e-7$	$3.3e-6$

Table 3: **Output Perturbations:** This table shows the output perturbations $\Delta\boldsymbol{q}$ corresponding to the stability plots in Figure **??**. Note that the values decrease in general with decreasing input perturbations, but the rate of decrease, as summarized in Figure **??**, is lower for more complex scenarios, like text optimization and dam break