Differentiable Two-Way Eulerian Fluid-Solid Coupling

Shrey Patel [1] **Samuel Audia** [1] **Rahul Narain** [2] **Ming Lin** [1]

Abstract

Gradients of physical quantities play an important role in derivative-based optimization and learning. Although much work has been done in differentiable physics for calculating gradients numerically by leveraging physical models, relatively little has been focused on two-way coupling of fluids with rigid bodies. Given the multitude of flow-based scenarios in the day-to-day tasks, we present an end-to-end differentiable Eulerian (grid-based) simulation with strong two-way coupling with rigid bodies. For the forward simulation, the solid-fluid boundary conditions are converted to a monolithic linear pressure solve using a variational method. For the backpropagation, we introduce a novel method of calculating and propagating gradients for the combined fluid-solid state using the adjoint method, which runs as fast as the forward solver. Our continuum-based formulation makes it more suitable over Lagrangian (particle-based) methods, for use cases where performance is key for analyzing overall flow patterns and learning fluid properties.

1. Introduction

Physical models, combined with accurate numerical methods, have been widely used to create predictive simulations crucial for studying physical properties of dynamical systems. Interactions between multiple types of objects, such as between fluids and solid bodies, further introduce computational complexity and interesting insights to their evolution in time. Recently, machine learning methods have been used for solving problems related to physics and other fundamental sciences. Such methods find applications in estimating physical parameters, guiding physical systems in control problems, or learning system dynamics for fast inference. However, most interesting physical systems, notably fluids, tend to be very high-dimensional in nature which, for most general purpose ML models, results in long training and/or inference times. Specialized models tailored for specific use cases tend to do better on this front, but often suffer from issues, like violation of physical constraints or limited numerical accuracy.

Consequently, there has been a growing interest in differentiable physics to derive gradients directly from a physically based simulation and enable a guided backpropagation to contract the massive search space typically encountered in physics problems. These gradients, as a physical prior, can then be combined with machine learning models to tackle the aforementioned problems – and many more.

There have been many significant advances in developing differentiable simulators for fluids, but they rarely account for interaction of fluids with dynamic obstacles like rigid bodies. In fact, because of moving boundary conditions, even the forward simulation is challenging, making the problem of finding gradients even more intractable for coupling scenarios. Additionally, the numerical methods employed depend upon the discretization scheme in the domain, each of which have their own pros and cons with respect to calculating gradients. Lagrangian methods assume a unified particle-based representation for fluids and solids, which simplifies treatment of different entities, and thus in gradient calculation. But particle-based methods often fail to preserve necessary constraints like incompressibility and rigid body shape, in addition to suffering from high computational cost of gradient computation, owing to large number of degrees of freedom. This representation translates to stability issues in the gradient calculation, as investigated by Li et al. (2023). Hybrid methods such as Material Point Methods (MPM) (Jiang et al., 2016), which combines Eulerian and Lagrangian representations, greatly complicate gradient computations, because information needs to be transferred between these two representations.

In this paper, we focus on differentiating an Eulerian (grid-based) method where the computational domain is a regular grid onto which entities like rigid bodies and fluids are discretized. Although this method does not provide arbitrary accuracy, the limited degrees of freedom ensure that the incompressibility and rigid body constraints hold in the continuum. Additionally, we show that the gradient computation is as efficient as the forward solve, which makes this approach highly scalable. Our main contributions are summarized as:

1. We analytically derive the gradients of the entire differentiable physics simulation pipeline, consisting primarily of fluid velocity advection (Section 4.1), monolithic

pressure solve for solid-fluid boundary conditions (Section 4.2) and velocity correction (Section 4.3). The derivation uses the adjoint method(McNamara et al., 2004).

2. We demonstrate the application of our computed gradients in an optimization process aimed at estimating an initial state of the rigid body which results in a desired final state both on a static surface and in a dynamic dam break scenario.

3. Finally, we compare our results with a particle-based differentiable simulator DiffFR(Li et al., 2023), showing that our Eulerian approach uses **3** times less memory and runs **27.8** times faster in a rigid body drop experiment.

2. Related Work

2.1. Differentiable Physics

Differentiable physics simulators have been proposed for numerous dynamical systems, including rigid bodies (Qiao et al., 2020; Freeman et al., 2021; de Avila Belbute-Peres et al., 2018), articulated bodies (Qiao et al., 2021; Degrave et al., 2016), soft bodies (Qiao et al., 2022; Du et al., 2021; Hu et al., 2018), cloth (Liang et al., 2019; Li et al., 2022), traffic (Son et al., 2022; Andelfinger, 2021), quantum computing (Leng et al., 2022), and coupled multi-body systems (Geilinger et al., 2020; Werling et al., 2021).

Differentiable physics offers many exciting new learning-enabled applications like structure identification and discovery (Jatavallabhula et al., 2021; Ingraham et al., 2019; Wang et al., 2020; Song & Boularias, 2020), policy and planning (Toussaint et al., 2018; Xu et al., 2022; Mora et al., 2021) as well as design and fabrication (Xu et al., 2021; Ma et al., 2021; Nava et al., 2022; Spielberg et al., 2019). (Huang et al., 2021).

Indeed, this method has become so versatile that many generalized differentiable programming paradigms have been proposed. Hu et al. (2019) introduced a new differentiable programming environment DiffTaichi to compute the gradients of physics simulations, and Heiden et al. (2021) provide a templatized simulation framework leveraging existing auto-differentiation tools like CppAD (Bell et al., 2018), Ceres (Agarwal et al., 2010), and PyTorch (Paszke et al., 2019). Nvidia Warp (Macklin, 2022) is a more recent differentiable framework with a focus on high-performance graphics and simulation.

2.2. Numerical Methods for Solid-Fluid Coupling

Grid-based numerical techniques for physics-based fluid simulation involve spatially discretizing the fluid dynamics (Navier-Stokes equation) on a regular staggered grid (Harlow & Welch, 1965), splitting the dynamics into individual operators (one for each term) and integrating the fluid variables like velocity through each operator while obeying boundary conditions. (Foster & Metaxas, 1996) pioneered this approach, using a finite difference based explicit solve. (Stam, 1999) proposed an implicit method with unconditional stability to extend the use of fluid simulation to real-time applications, where large timesteps are preferred. Due to these and many other subsequent works, most grid-based fluid simulators solve advection using higher order methods like the Semi-Lagrangian scheme (Robert et al., 1985) and projection using a pressure Poisson solve. For boundary conditions, free surfaces are addressed using ghost-fluid conditions (Gibou et al., 2002), while a non-coupled static solid boundary is handled using level sets (Foster & Fedkiw, 2001).

Moving boundaries, a central part of solid-fluid coupling, are more challenging. The standard technique is to integrate fluid pressure onto the solid as external force and provide solid velocity as a boundary condition for the fluid. Partitioned approaches (Carlson et al., 2004; Banks et al., 2018; Akbay et al., 2018) have separate solvers for fluid and solid dynamics with some degree of inter-communication, called sequentially, solving for a constraint might violate a previously enforced one. Further, the use of iterative procedures to achieve coupling leads to convergence and stability issues. Many of these problems are alleviated by monolithic approaches (Klingner et al., 2006; Robinson-Mosher et al., 2011; Aanjaneya, 2018; Gibou & Min, 2012; Zarifi & Batty, 2017), which aim to combine the solid-fluid dynamics into a single implicit system, though the methods to arrive at such a coupled system differ dramatically.

Our work is inspired by (Batty et al., 2007), who used the variational method to convert complex solid-fluid interactions into a kinetic energy minimization problem. The minimization can then be simplified into a symmetric positive semi-definite linear system on pressure. This method has been quite successful and adapted by several, including (Larionov et al., 2017; Takahashi & Lin, 2019; Takahashi & Batty, 2020) for viscous fluids, (Takahashi & Batty, 2021) for granular flow, and (Takahashi & Batty, 2022) for elastic-rigid coupling.

2.3. Differentiable Fluid Simulation

To control smoke through keyframe matching, (Treuille et al., 2003) formulated a nonlinear optimization problem which uses derivatives of the simulator. Later, (McNamara et al., 2004) extended this approach to level-set based liquid boundaries and adapted the adjoint method (Lions, 1971) to accelerate gradient calculation. (Takahashi et al., 2021) combined the adjoint method with the variational method of solid-fluid coupling (Batty et al., 2007) to obtain a one-way

coupled differentiable simulator. (Li et al., 2024) extended the differentiable pipeline with derivatives for the boundary geometry, unlocking optimization based design applications. Recently, (Li et al., 2023) managed to design a differentiable simulator for two-way rigid-fluid coupling using a unified particle representation of fluids and solids. They use the implicit DFSPH method (Bender & Koschier, 2015) for fluid simulation and the method of (Akinci et al., 2012) for fluid-rigid coupling. Even though they propose a localized gradient computation scheme to decrease the computational cost typical of particle-based approaches, grid-based fluid simulation methods employ far fewer degrees of freedom and are therefore expected to have lower computational cost, which is the impetus for our method.

There is a growing trend of using data-driven approaches for super-resolution, inferencing frames, or imposing control in fluid simulations, using established deep learning models like feed forward neural networks (Prantl et al., 2019; Um et al., 2018), convolutional neural networks (CNNs) (Tompson et al., 2022), generative adversarial networks (GANs) (Xie et al., 2018; Kim et al., 2019), and long short term memory (LSTM) networks (Wiewel et al., 2019). Because of the high dimensionality of physical systems, it is very challenging to ensure that physical constraints hold in the dynamics learned by these models. Often this limitation leads to poor performance on novel samples. (Raissi et al., 2019) introduced physics informed neural networks (PINNs) by augmenting existing networks with physics-based loss functions to ensure that physical constraints are respected.

Differentiable physics simulators go one step further by explicitly providing derivatives with respect to the state of the physical system. These derivatives can then either be used directly to demonstrate simple control experiments or fed to a deep learning model for sophisticated learning-based applications. (Holl et al., 2020) train a hierarchical predictor-corrector model that learns to understand and control complex physical systems represented by PDEs, using gradients from their differentiable simulator PhiFlow, leveraging low level auto-differentiation with adjoint method. (Ramos et al., 2022) explore whether a neural network trained using PhiFlow, and augmented with specialized physical loss functions, can act as a controller for a two-way coupled fluid-rigid system. But, they use weak coupling, solving the fluid and solid dynamics one after the other, instead of using a monolithic solve, thereby leaving the method more likely to be prone to numerical stability issues.

3. Problem Setup

We setup the relevant mathematical equation at a high level and motivate the differentiable simulator.

3.1. Background

The fluid dynamics are governed by the inviscid Navier Stokes equation, paired with the incompressibility constraint:

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

where $\boldsymbol{u}$: fluid velocity, ρ: fluid density, p: fluid pressure and $\boldsymbol{f}$: external force (i.e. gravity, wind forces, etc.) 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. We detail each of these stages in Section 4, along with their derivatives. The fluid domain is discretized as a regular MAC grid ((Harlow & Welch, 1965)), with velocity at face centers and pressure at cell centers. From here on, $\boldsymbol{u}$, $\mathbf{p}$ represent the vectors of fluid velocity and pressure samples respectively.

We address rigid body dynamics using semi-implicit time-stepping:

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

where $\boldsymbol{v}$: solid velocity, $\boldsymbol{x}$: solid position, $\boldsymbol{F}$: external force, and $\mathbf{M}_S$: mass matrix of solid. Each of these pertain to the centre of mass and include both translation and rotation components. f_x is a per element function on vectors, such that $f_x(x, v) = v$, except for quaternions, in which case $f_x(q, \omega) = 0.5([0, \omega] \otimes q)$, $\otimes$ being quaternion multiplication.

Pressure, serving as an external force for the rigid body, and as a quantity enforcing incompressibility and rigid-fluid boundary conditions on the fluid velocity field, acts as the coupling mechanism between the two domains.

At any point in the simulation, say after t time steps, let $\boldsymbol{q}_t = (\boldsymbol{u}_t, \boldsymbol{v}_t, \boldsymbol{x_t})$ represent the state of the combined rigid-fluid system. Each of the stages in the numerical solution acts as a function ϕ between states. Assuming there are k such stages ($k = 5$ in our case), ϕ_k's compose to form one simulation step $\Phi = \phi_k \circ \dots \phi_2 \circ \phi_1$, so that $\boldsymbol{q}_{t+1} = \Phi(\boldsymbol{q}_t)$. We use fractional time steps e.g. $\boldsymbol{q}_{t+\frac{1}{k}} = \phi_1(\boldsymbol{q}_t)$, only to denote each stage of the simulation, otherwise each of the stages are applied for the full time-step. Refer to Figure 1 as an illustration.

3.2. Problem Specification

Our general objective is to find an initial state $\widetilde{\boldsymbol{q}}_0$, which through n simulation steps, results in a final desired state $\widetilde{\boldsymbol{q}}_n$. We pose this as an optimization problem. Suppose a convex loss function $\mathcal{L}$ measures the distance between two states and $\boldsymbol{q}_n = \Phi^n(\boldsymbol{q}_0)$, then $\widetilde{\boldsymbol{q}}_0 = \arg\min_{\boldsymbol{q}_0} \mathcal{L}(\boldsymbol{q}_n, \widetilde{\boldsymbol{q}}_n)$.

Unlike neural networks, we do not need to specialize the loss function or constrain the optimization to avoid violating any boundary conditions, since the simulation based output $\boldsymbol{q}_n$ already enforces them.

Starting from an arbitrary initial state $\boldsymbol{q}_0$, the derivative $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0}$ points to a direction of decreasing value of $\mathcal{L}$ (Boyd & Vandenberghe (2004)). Indeed, this is the central idea behind gradient descent algorithms, employed in a majority of learning based problems. Our contribution is the derivation and calculation of $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0}$.

3.3. Adjoint Simulation

The quantity $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_n}$ can be computed directly from $\mathcal{L}$. We use reverse mode differentiation to obtain $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0}$ from this point. For state $\boldsymbol{q}_t$, consider its adjoint state $\boldsymbol{Q}_t = \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_t} = \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t}\right)$. Then, from chain rule, $\boldsymbol{Q}_t = \Psi(\boldsymbol{Q}_{t+1})$, where $\Psi = \frac{\partial \Phi}{\partial \boldsymbol{q}}$. Additionally, the sequential structure of Φ allows the composition: $\Psi = \psi_1 \circ \psi_2 \cdots \circ \psi_k$, $\psi_i = \frac{\partial \phi_i}{\partial \boldsymbol{q}}$. As a result, we have that $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_0} = \boldsymbol{Q}_0 = \Psi^n(\boldsymbol{Q}_n) = \Psi^n\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}_n}\right)$.

Effectively, in the same way that the simulation Φ transforms an initial state $\boldsymbol{q}_0$ to $\boldsymbol{q}_n$, the adjoint simulation Ψ transforms the adjoint state $\boldsymbol{Q}_n$ to $\boldsymbol{Q}_0$, closing one loop of optimization. Again, refer to the flow diagram in Figure 1.

4. Differentiable 2-Way Solid-Fluid Coupling

In this section, we elaborate different stages of simulation ϕ_i, and their corresponding adjoint ψ_i. Not all stages require and/or modify all the variables $\boldsymbol{u}$, $\boldsymbol{v}$, and $\boldsymbol{x}$ of combined state $\boldsymbol{q}$. For instance, in the forward direction, advection (ϕ_1) only requires and affects the fluid velocity field $\boldsymbol{u}$, while the pressure solve (ϕ_3) requires the entire state $\boldsymbol{q}$, but affects only the velocities $\boldsymbol{u}$, $\boldsymbol{v}$. This saves significant derivative computation, like in the case of advection (ψ_1), where we do not need to compute $\frac{\partial \phi_1}{\partial \boldsymbol{x}}$ or $\frac{\partial \phi_1}{\partial \boldsymbol{v}}$. Going forward, it is implied that if a stage ϕ_i does not require a variable $\boldsymbol{y}$, $\boldsymbol{y}_{t_1} = \boldsymbol{y}_{t_2}$ and if it also does not modify $\boldsymbol{y}$, $\boldsymbol{Y}_{t_1} = \boldsymbol{Y}_{t_2}$; $t_1 = t + \frac{i-1}{5}$, $t_2 = t + \frac{i}{5}$, $\boldsymbol{Y} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}$. Dependencies are shown in Figure 1.

Constant external forces like gravity, control forces, etc. only add an offset to the velocity, so there is no change in the adjoint state. On applying force $\boldsymbol{f} = (\boldsymbol{f}_u, \boldsymbol{f}_v)$,

$$\begin{aligned}\left(\boldsymbol{u}_{t+\frac{2}{5}}, \boldsymbol{v}_{t+\frac{2}{5}}\right) &= \phi_2\left(\boldsymbol{u}_{t+\frac{1}{5}}, \boldsymbol{v}_{t+\frac{1}{5}}\right) \\ &= \left(\boldsymbol{u}_{t+\frac{1}{5}} + \frac{\Delta t}{\rho}\boldsymbol{f}_u, \boldsymbol{v}_{t+\frac{1}{5}} + \Delta t\ \mathbf{M}^{-1}\boldsymbol{f}_v\right) \qquad (3) \\ \therefore \boldsymbol{Q}_{t+\frac{1}{5}} &= \psi_2\left(\boldsymbol{Q}_{t+\frac{2}{5}}\right) = \boldsymbol{Q}_{t+\frac{2}{5}}.\end{aligned}$$

We treat other stages in their separate subsections.

4.1. Advection

In grid-based methods, it is crucial to account for the movement of fluid while tracking changes in fluid velocity, which remains stationary in space with the background grid, unlike the fluid itself. The advection term $\boldsymbol{u} \cdot \nabla \boldsymbol{u}$ in Equation 1 captures this behavior. To solve the term, we use semi-Lagrangian discretization (Stam, 1999) in space for its stability, and forward Euler discretization in time, for its relatively simple derivative calculation. Bridson et al. (2006) elucidates this concept.

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 $u_{t+\frac{1}{5}}(\boldsymbol{x}_g) = \mathcal{I}(\boldsymbol{u}_t, \boldsymbol{x_g} - \Delta t \mathcal{I}(\boldsymbol{u}_t, \boldsymbol{x}_g))$. Collecting them into a vector:

$$\begin{aligned}\boldsymbol{u}_{t+\frac{1}{5}} &= \phi_1(\boldsymbol{u}_t) \\ &= \mathbf{W}\boldsymbol{u}_t \\ \therefore \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} &= \psi_1\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{1}{5}}}\right) \qquad (4) \\ &= \left(\mathbf{W} + \frac{\partial \mathbf{W}}{\partial \boldsymbol{u}}\boldsymbol{u}_t\right)^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{1}{5}}},\end{aligned}$$

where $\mathbf{W}(\boldsymbol{u}_t)$ is the matrix of interpolation weights. A row signifies weights corresponding to a backtracked location $\boldsymbol{x_g} - \Delta t \mathcal{I}(\boldsymbol{u}_t, \boldsymbol{x}_g)$ inside the grid $\boldsymbol{u}_t$, while a column represents weights of a particular sample of $\boldsymbol{u}_t$.

4.2. Pressure Solve

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

$$\begin{aligned}\boldsymbol{u}_{t+\frac{3}{5}}(\boldsymbol{y}) = \boldsymbol{u}_{t+\frac{2}{5}}(\boldsymbol{y}) - \frac{\Delta t}{\rho}\nabla p(\boldsymbol{y}), &\quad \boldsymbol{y} \in \Omega_F \\ \nabla \cdot \boldsymbol{u}_{t+\frac{3}{5}}(\boldsymbol{y}) = 0, &\quad \boldsymbol{y} \in \Omega_F \\ p(\boldsymbol{y}) = 0, &\quad \boldsymbol{y} \in \Omega_{FA} \qquad (5) \\ \boldsymbol{v}_{t+\frac{3}{5}} = \boldsymbol{v}_{t+\frac{2}{5}} + \Delta t \mathbf{M}_S^{-1}\boldsymbol{F}_p&\left(\boldsymbol{x}_{t+\frac{2}{5}}, p\right) \\ \left(\boldsymbol{u}_{t+\frac{3}{5}}(\boldsymbol{y}) - \boldsymbol{v}_{t+\frac{3}{5}}(\boldsymbol{y})\right) \cdot \hat{n} = 0, &\quad \boldsymbol{y} \in \Omega_{FS},\end{aligned}$$

where $\boldsymbol{q}_{t+\frac{2}{5}} = \left(\boldsymbol{u}_{t+\frac{2}{5}}, \boldsymbol{v}_{t+\frac{2}{5}}, \boldsymbol{x_{t+\frac{2}{5}}}\right)$: input state, $\boldsymbol{q}_{t+\frac{3}{5}} = \left(\boldsymbol{u}_{t+\frac{3}{5}}, \boldsymbol{v}_{t+\frac{3}{5}}\right)$: output state, $\boldsymbol{F}_p$: external force on rigid body, and $\boldsymbol{y}_{CM}$: centre of mass of rigid body. We use the variational interpretation of pressure, inspired from Batty et al. (2007), to convert the set of Equations 5 into a monolithic linear system. We rewrite the pressure field as vector

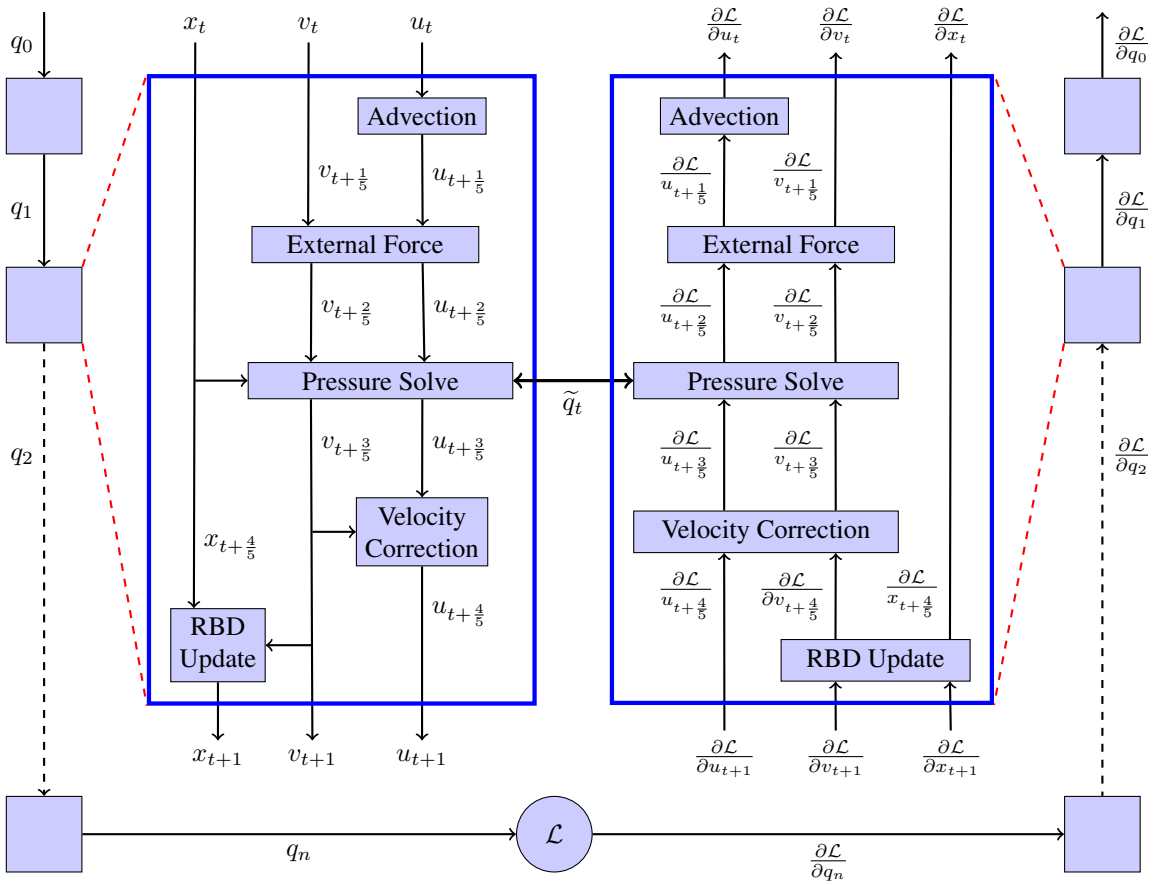


Figure 1: **Algorithm Flow Diagram.** We show the forward pass (left) and backward pass (right) of our differentiable fluid simulator. Each time step acts on the combined solid-fluid state q or the combined adjoint state $\frac{\partial \mathcal{L}}{\partial q}$. The zoomed in diagram shows each of the sub-stages in one time step. The fractional subscripts indicate intermediate values. There is some degree of sharing between the forward and backward pass for efficiency. This framework is general and can solve a variety of optimization and learning problems with accurate fluid-solid coupling.

$\mathbf{p}$ and the gradient operator as matrix $\mathbf{G}$. Pressure force being linear in pressure, we write $\boldsymbol{F}_p$ as the vector $\mathbf{Jp}$, where matrix $\mathbf{J}$ depends on the location $\boldsymbol{x}_{t+\frac{2}{5}}$ of the rigid body. Treating each velocity sample as a binary decision of presence/absence of liquid in each cell causes artifacts on non-grid aligned boundaries, as investigated previously by Batty et al. (2007); Takahashi & Lin (2019); Larionov et al. (2017). So, we use a fluid mass matrix $\mathbf{M}_F$ to associate a mass value with each fluid velocity sample, depending on the amount of fluid in the corresponding grid cell. It allows us to properly enforce mass conservation on the fluid. Leaving out the full derivation, which can be found in Batty et al. (2007), Bridson et al. (2006), the matrix version of Equations 5 resolves to:

$$\begin{aligned}\left(\boldsymbol{u}_{t+\frac{3}{5}}, \boldsymbol{v}_{t+\frac{3}{5}}\right) &= \phi_3\left(\boldsymbol{u}_{t+\frac{2}{5}}, \boldsymbol{v}_{t+\frac{2}{5}}, \boldsymbol{x}_{t+\frac{2}{5}}\right) \\ &= \left(\boldsymbol{u}_{t+\frac{2}{5}} - \frac{\Delta t}{\rho}\mathbf{Gp},\ \boldsymbol{v}_{t+\frac{2}{5}} + \Delta t \mathbf{M}_S^{-1}\mathbf{Jp}\right)\end{aligned} \tag{6}$$

where pressure $\mathbf{p}$ is obtained from the linear system:

$$\begin{aligned}\mathbf{Ap} &= \frac{1}{\rho}\mathbf{G}^T\mathbf{M}_F\boldsymbol{u}_{t+\frac{2}{5}} - \mathbf{J}^T\boldsymbol{v}_{t+\frac{2}{5}} \\ \text{where } \mathbf{A} &= \left[\frac{\Delta t}{\rho^2}\mathbf{G}^T\mathbf{M}_F\mathbf{G} + \Delta t \mathbf{J}^T\mathbf{M}_S^{-1}\mathbf{J}\right].\end{aligned} \tag{7}$$

The full version of the gradients, derived in Appendix 1, features a term $\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}}$, which is a non-zero tensor; however, we assume that for a small enough time step, the matrix $\mathbf{J}$ does not change much. The value of $\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}}$ is then close to zero. Consequently, the derivative $\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}}$ remains mostly unchanged. The rest of the adjoint update is:

$$\begin{aligned}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{2}{5}}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+\frac{2}{5}}}\right) &= \psi_3\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{3}{5}}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+\frac{3}{5}}}\right) \\ &= \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{3}{5}}} + \frac{1}{\rho}\mathbf{M}_F\mathbf{Gs}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+\frac{3}{5}}} - \mathbf{Js}\right),\end{aligned} \tag{8}$$

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

$$\mathbf{As} = \left[\Delta t \mathbf{J}^T \mathbf{M}_S^{-1} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+\frac{3}{5}}}\right) - \frac{\Delta t}{\rho} \mathbf{G}^T \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{3}{5}}}\right)\right]. \tag{9}$$

4.3. Fluid Velocity Correction

Because of the choice of discretization, the boundary conditions from Equations 5 are only applied to velocity samples adjacent to grid cells containing fluid. But, in every stage, the fluid velocity is advected even where samples are not affected by pressure. These *invalid* velocities must be corrected to make sure that the advected velocity obeys the boundary conditions.

The method of corrections 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 type iterations. A larger time step requires larger number of iterations. In our experiments, 3 such iterations work sufficiently well. If on the other hand, the velocity sample lies in the interior of 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 $\boldsymbol{u}_{t+\frac{4}{5}}$ is linear in the combined velocity $\left(\boldsymbol{u}_{t+\frac{3}{5}}, \boldsymbol{v}_{t+\frac{3}{5}}\right)$:

$$\begin{aligned} \boldsymbol{u}_{t+\frac{4}{5}} &= \mathbf{C} \begin{bmatrix} \boldsymbol{u}_{t+\frac{3}{5}} \\ \boldsymbol{v}_{t+\frac{3}{5}} \end{bmatrix} \\ \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{3}{5}}} & \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+\frac{3}{5}}} \end{bmatrix}^T &= \mathbf{C}^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+\frac{4}{5}}}, \end{aligned} \tag{10}$$

where $C\left(\boldsymbol{u}_{t+\frac{3}{5}}, \boldsymbol{v}_{t+\frac{3}{5}}\right)$ is the correction matrix as a function of input combined velocity.

4.4. Rigid Body Time Integration

This step updates the position of a rigid body, $\boldsymbol{x}_{t+1}$, using velocity $\boldsymbol{v}_{t+\frac{4}{5}}$: $\boldsymbol{x}_{t+1} = \phi_5\left(\boldsymbol{x}_{t+\frac{4}{5}}, \boldsymbol{v}_{t+\frac{4}{5}}\right) = \boldsymbol{x}_{t+\frac{4}{5}} + \Delta t\ f_x\left(\boldsymbol{x}_{t+\frac{4}{5}}, \boldsymbol{v}_{t+\frac{4}{5}}\right)$. For translational components $f_x(x, v) = v$ and $\left(\frac{\partial \mathcal{L}}{\partial x_{t+\frac{4}{5}}}, \frac{\partial \mathcal{L}}{\partial v_{t+\frac{4}{5}}}\right) = \left(\frac{\partial \mathcal{L}}{\partial x_{t+1}}, \Delta t\ \frac{\partial \mathcal{L}}{\partial x_{t+1}}\right)$.

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

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial q_{t+\frac{4}{5}}} &= \left(I_{4\times 4} + \frac{\Delta t}{2} \frac{\partial\left(\left[0, \omega_{t+\frac{4}{5}}\right] \otimes q_{t+\frac{4}{5}}\right)}{\partial q_{t+\frac{4}{5}}}\right) \frac{\partial \mathcal{L}}{\partial q_{t+1}} \\ \frac{\partial \mathcal{L}}{\partial \omega_{t+\frac{4}{5}}} &= \left(\frac{\Delta t}{2} \frac{\partial\left(\left[0, \omega_{t+\frac{4}{5}}\right] \otimes q_{t+\frac{4}{5}}\right)}{\partial \omega_{t+\frac{4}{5}}}\right) \frac{\partial \mathcal{L}}{\partial q_{t+1}}. \end{aligned} \tag{11}$$

Partial derivatives of $\left(\left[0, \omega_{t+\frac{4}{5}}\right] \otimes q_{t+\frac{4}{5}}\right)$ can be computed easily from the definition of quaternion multiplication $\otimes$.

5. Implementation

We have implemented our differentiable simulator in C++. OpenMP was used to parallelize all steps of the code in the forward and backward pass. The Eigen (Guennebaud et al., 2010) library was used to solve for the pressure using the Simplicial LDLT Cholesky factorization. The forward simulation is adapted from the *FluidRigidCoupling2D* library (Batty, 2016). We use a uniform MAC grid with an offset of 2 cells along each boundary to account for boundary conditions against a solid walled tank. Marker particles are used to track the free surface and for rendering. These particles are updated using semi-Lagrangian advection in each time step. For rendering, we first generate a reconstructed liquid surface from marker particles using *SplashSurf* library (Löschner et al., 2023), and then perform the final renderings in Blender (Community, 2018). Appendix B includes pseudo code for the simulation.

6. Experiments

We evaluate our simulation on a variety of optimization problems. We also include a comparison against DiffFR (Li et al., 2023) to show the improved performance of our grid based method over their particle based method. All experiments are run on an AMD Ryzen 3700X with 16 threads at 4.1 GHz and 32 GB of system memory. For optimization, we use Gradient Descent (Kiefer & Wolfowitz, 1952) with a learning rate of 1.0.

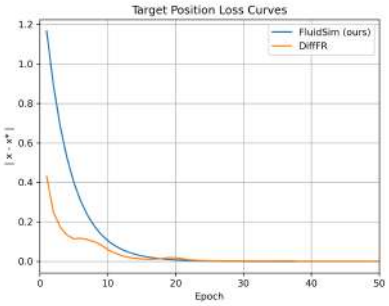


Figure 2: We plot the loss curves for our method and DiffFR on the rigid body velocity initialization experiment. Both methods quickly reach the target by the 20th epoch. Our method provides a smoother, exponentially decreasing loss curve. The SPH based method is not monotonically decreasing, which may be due to the complex interactions between particles.

Block Drop. As a first test for our simulator, we seek to optimize the initial velocity of a block dropping into water, such that it reaches the correct final position and orientation. We ran a fluid domain 3.5 meters wide by 1.5 meters tall by 2 meters deep. The 0.2 meter square cube of density 2000

kg/m^3 was dropped from 0.25 meters above the center of the water surface. Before the first gradient descent step, the cube had an initial velocity of $[-1, -1, 1]$ along x, y, and z. The body was given no initial angular velocity, but the optimizer was allowed to update this value during training. The grid spacing was 0.1 meters with a time step of 0.005 seconds. The simulation was stepped by 50 frames with 50 training epochs. Each forward pass was around 6 seconds, while the backward pass was about 1.5 seconds. Figure 4 shows the results at the start and end of training. At the beginning, the block is only partially submerged in the liquid after 50 time steps. After 50 gradient updates, the block quickly reaches its final position near the bottom corner of the tank. We see that our simulator quickly finds the initial velocities as shown by the loss curve in Figure 2.

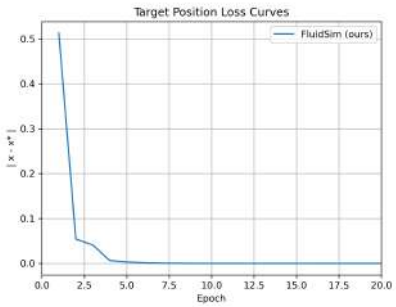


Figure 3: We plot the loss curve the the dam break experiment and find that the loss quickly reduces to a small value. This shows that our method is able to quickly optimize parameters is highly dynamic environments.

Dam Break. We now demonstrate our simulator's performance on a more dynamic scenario. We double the number of grid cells along the x direction to 40 and keep the same grid spacing of 0.1 meters. The time stepping was increased to 0.05 seconds per frame. These parameters results in 48000 grid cell values. We start with a block of fluid along the right wall of the tank. This block is 0.4 meters wide, 0.7 meters tall, and 1 meter deep. A ball of radius 0.2 meters is placed at $[1.6, 1.4, 1.0]$ in x, y, and z. The ball has no initial velocity and is allowed to free fall into the dam break produced by the fluid. The goal of the optimizer is again to update the initial velocity of the ball, such that it reaches the target position of $[2.0, 0.6, 0.6]$ after 50 time steps. We found that setting the learning rate to 0.1 gave the best performance in this experiment. We run the simulation for 20 epochs and plot results in Figure 5. Each forward pass took around 3 minutes with the backward taking 2 seconds. This discrepency is most likely due to an unoptimized collision detection in the marker particle update. We see that our simulation is again able to recover the correct initial velocity to match the target even though there are complex dynamics in the scene. Figure 3 shows that the loss quickly converges to bring the ball to the correct position. This experiment demonstrates the robustness of our method to different fluid environments.

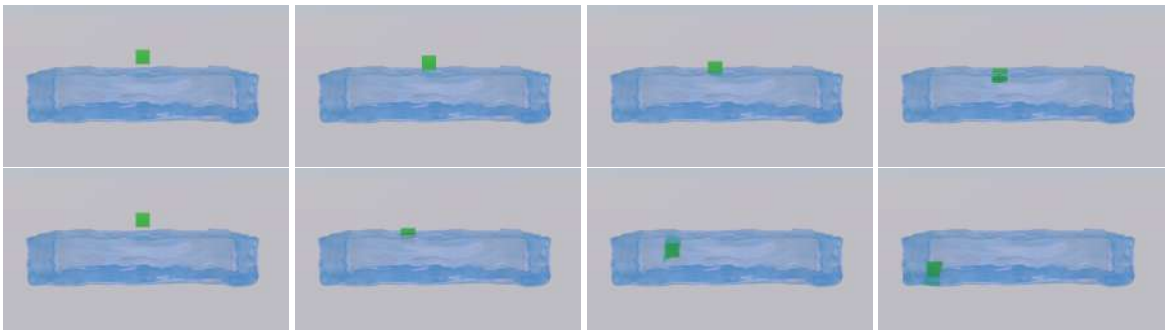
Method	Memory (MB)	Time (h:m)
FluidSim (ours)	180	0:6.18
Diff-FR (SPH)	510	2:51.83
Improvement	**2.83x**	**27.80x**

Table 1: This table shows memory and runtime comparisons between our method and DiffFR on a rigid body drop over an initially static surface. We find that our solver uses a fraction of the memory, while reducing a 3 hour runtime to under 10 minutes. These differences come from our use of an Eulerian grid greatly reducing the number of computations necessary in the forward and backward differentiable fluid-solid coupling simulations.

DiffFR Comparison. We compare the rigid body drop experiement between our simulation and DiffFR (Li et al., 2023). The problem setup is the same between the two simulations. Using the default settings from DiffFR, they use a total of 237699 particles, while our method requires only 53352 grid cells on the same domain. The order of magnitude difference is the key benefit of our method as it greatly reduces the computational time needed to perform both the forward and backward simulations. For these tests, no rendering was performed in the loop, so we did not advect marker particles in our simulation. This meant that large changes in fluid surface would not be tracked, but the movement of the block was not enough create a large perturbation.

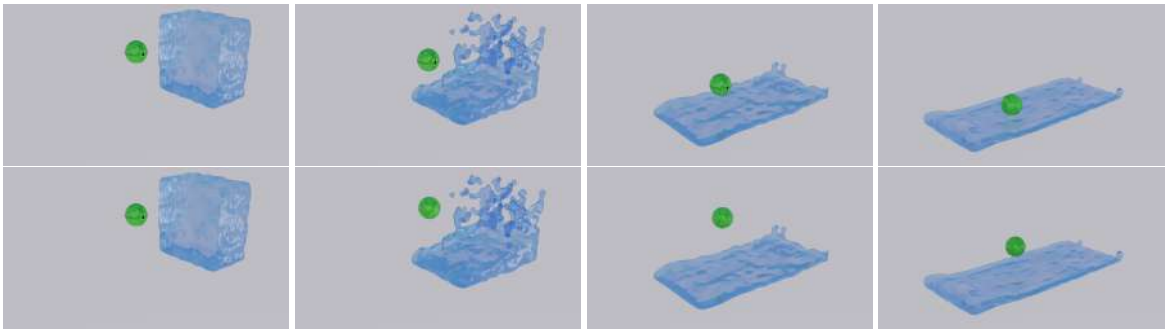
The runtimes of the two simulations are reported in Table 1. Our simulation runs 27.80 times faster reducing a nearly 3 hour run to under 10 minutes. At the same time, our method uses almost 3 times less memory due to the differences between the Eulerian and Lagrangian discretizations. Figure 2 shows the loss curves for the two simulations. Our simulation exponentially decreases the loss and quickly converges to the target. DiffFR does not monotonically decrease in loss; however, both method reach reasonable answer within 20 iterations. Our method greatly reduces the computational cost of differentiable solid-fluid coupling while achieving similar accuracy as the state of the art.

Similar to DiffFR, we found that the gradients of the backward pass explode due to instability. For shorter time scales this does not affect the results, but for longer simulations, this will need to be addressed. Naive clipping of the gradients does not immediately fix the issue. The local gradient method used by DiffFR does not directly apply to our work; however, we do seek to only optimize the initial body velocity as the adjoint fluid velocities appeared to be the most unstable. We hope to address these limitations in future work.



Frame 0 Frame 16 Frame 33 Frame 50

Figure 4: **Block Motion Control in Fluid Flows.** We optimize the initial linear and angular velocity of the red block so that it reaches the bottom left corner of the fluid tank. The top row shows the block trajectory over the simulation before training. The bottom row shows the trajectory after training the 50th epoch. While the initial velocity barely submerges the block in the liquid, the learned velocity quickly finds the target block position.



Frame 0 Frame 16 Frame 33 Frame 50

Figure 5: **Dam Break Control.** We again optimize the linear and angular velocity of the rigid body as it is dropped into a fluid. We compute a loss based on the ball's final position relative to a target location. Our method quickly optimizes the velocities to meet this goal in dynamic environment with large fluid motions.

7. Summary and Conclusion

Physical simulation is important in engineering and modeling our world, whether it is for science or computer graphics. The solutions to the equations that govern the simulations are incredibly high dimensional and represent complex interactions. Black box or hand-tuned optimization of parameters is inefficient and may not even converge to the desired results. By incorporating gradient information and backwards simulation, first-order optimization methods become available, which greatly improves convergence. The derivation of this backward pass is non-trivial, and just like the forward simulation, there are many different techniques for doing so. Our work provides an Eulerian approach to differentiable two-way solid-fluid coupling, which is *orders of magnitude* faster than the state-of-the-art (SOTA) method based on a Lagrangian discretization. We show that our method can handle both static and dynamic scenes by considering a rigid-body drop and the dam-break problem. In the rigid-body experiment, we found that our simulation ran 28 times faster than the SOTA.

However, there is still room for further improvement. First, robust numerical treatment is required to enable gradient computation for a large number of time steps, in order to handle longer and more meaningful simulation. Second, modeling solids that are smaller than a grid cell is difficult and can result in incorrect torques in the forward simulation. We also do not consider viscosity in our work for simplicity. Future work can seek to address these problems and further improve the performance of this method. For example, fluid simulations often use adaptive grids for their discretization and multi-grid methods to speed up their pressure solver. Graphics processing units can also greatly improve performance by computing across all grid cells in parallel. Our method greatly improves on the speed and memory footprint of existing two-way coupling fluid-solid simulations and opens up a broad research path for continually improving differentiable fluid simulation.

Impact Statement

The main impact of our work is that it facilitates the broader scientific community in adopting learning-based methods for problem-solving in physics-based simulations. We do not foresee any direct harmful impacts of our contribution on any individual, society, or the environment.

References

Aanjaneya, M. An efficient solver for two-way coupling rigid bodies with incompressible flow. *Computer Graphics Forum*, 37(8):59–68, July 2018.

Agarwal, S., Mierle, K., et al. Ceres solver. http://ceres-solver.org, 2010.

Akbay, M., Nobles, N., Zordan, V., and Shinar, T. An extended partitioned method for conservative solid-fluid coupling. *ACM Trans. Graph.*, 37(4), July 2018.

Akinci, N., Ihmsen, M., Akinci, G., Solenthaler, B., and Teschner, M. Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph.*, July 2012.

Andelfinger, P. Differentiable agent-based simulation for gradient-guided simulation-based optimization, 2021.

Banks, J., Henshaw, W., Schwendeman, D., and Tang, Q. A stable partitioned fsi algorithm for rigid bodies and incompressible flow in three dimensions. *Journal of Computational Physics*, 373:455–492, 2018.

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

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

Bell, B. M. et al. CppAD: C++ algorithmic differentiation. https://projects.coin-or.org/CppAD, 2018.

Bender, J. and Koschier, D. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pp. 147–155, 2015.

Boyd, S. and Vandenberghe, L. *Convex optimization*. Cambridge university press, 2004.

Bridson, R., Fedkiw, R., and Müller-Fischer, M. Fluid simulation: Siggraph 2006 course notes (fedkiw and muller-fischer presenation videos are available from the citation page). In *ACM SIGGRAPH 2006 Courses*, pp. 1–87, 2006.

Carlson, M., Mucha, P. J., and Turk, G. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Trans. Graph.*, 23(3):377–384, August 2004.

Community, B. O. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

de Avila Belbute-Peres, F., Smith, K., Allen, K., Tenenbaum, J., and Kolter, J. Z. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Degrave, J., Hermans, M., Dambre, J., and Wyffels, F. A differentiable physics engine for deep learning in robotics. *CoRR*, abs/1611.01652, 2016.

Du, T., Wu, K., Ma, P., Wah, S., Spielberg, A., Rus, D., and Matusik, W. Diffpd: Differentiable projective dynamics. *ACM Trans. Graph.*, 41(2), nov 2021.

Foster, N. and Fedkiw, R. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 23–30, 2001.

Foster, N. and Metaxas, D. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. Brax – a differentiable physics engine for large scale rigid body simulation, 2021.

Geilinger, M., Hahn, D., Zehnder, J., Bächer, M., Thomaszewski, B., and Coros, S. Add: Analytically differentiable dynamics for multi-body systems with frictional contact, 2020.

Gibou, F. and Min, C. Efficient symmetric positive definite second-order accurate monolithic solver for fluid/solid interactions. *Journal of Computational Physics*, 231(8): 3246–3263, 2012.

Gibou, F., Fedkiw, R. P., Cheng, L.-T., and Kang, M. A second-order-accurate symmetric discretization of the poisson equation on irregular domains. *Journal of Computational Physics*, 176(1):205–227, 2002.

Guennebaud, G., Jacob, B., et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

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

Heiden, E., Millard, D., Coumans, E., Sheng, Y., and Sukhatme, G. S. NeuralSim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://github.com/google-research/tiny-differentiable-simulator.

Holl, P., Koltun, V., and Thuerey, N. Learning to control pdes with differentiable physics, 2020.

Hu, Y., Liu, J., Spielberg, A., Tenenbaum, J. B., Freeman, W. T., Wu, J., Rus, D., and Matusik, W. Chainqueen: A real-time differentiable physical simulator for soft robotics, 2018.

Hu, Y., Li, T.-M., Anderson, L., Ragan-Kelley, J., and Durand, F. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

Huang, Z., Hu, Y., Du, T., Zhou, S., Su, H., Tenenbaum, J. B., and Gan, C. PlasticineLab: A soft-body manipulation benchmark with differentiable physics. In *ICLR*, 2021.

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

Jatavallabhula, K. M., Macklin, M., Golemo, F., Voleti, V., Petrini, L., Weiss, M., Considine, B., Parent-Levesque, J., Xie, K., Erleben, K., Paull, L., Shkurti, F., Nowrouzezahrai, D., and Fidler, S. gradsim: Differentiable simulation for system identification and visuomotor control, 2021.

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., and Selle, A. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16. Association for Computing Machinery, 2016.

Kiefer, J. and Wolfowitz, J. Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462–466, 3 1952. ISSN 00034851. URL http://www.jstor.org/stable/2236690.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B. Deep fluids: A generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, May 2019.

Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F. Fluid animation with dynamic meshes. *ACM Trans. Graph.*, 25(3):820–825, July 2006.

Larionov, E., Batty, C., and Bridson, R. Variational stokes: a unified pressure-viscosity solver for accurate viscous liquids. *ACM Trans. Graph.*, 36(4), jul 2017.

Leng, J., Peng, Y., Qiao, Y.-L., Lin, M. C., and Wu, X. Differentiable analog quantum computing for optimization and control. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

Li, Y., Du, T., Wu, K., Xu, J., and Matusik, W. Diffcloth: Differentiable cloth simulation with dry frictional contact. *ACM Trans. Graph.*, 42(1), 2022.

Li, Y., Sun, Y., Ma, P., Sifakis, E., Du, T., Zhu, B., and Matusik, W. Neuralfluid: Neural fluidic system design and control with differentiable simulation, 2024. URL https://arxiv.org/abs/2405.14903.

Li, Z., Xu, Q., Ye, X., Ren, B., and Liu, L. Difffr: Differentiable sph-based fluid-rigid coupling for rigid body control. *ACM Trans. Graph.*, 42(6), dec 2023.

Liang, J., Lin, M., and Koltun, V. Differentiable cloth simulation for inverse problems. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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

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

Ma, P., Du, T., Zhang, J. Z., Wu, K., Spielberg, A., Katzschmann, R. K., and Matusik, W. Diffaqua: A differentiable computational design pipeline for soft underwater swimmers with shape interpolation. *ACM Transactions on Graphics (TOG)*, 40(4):132, 2021.

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

McNamara, A., Treuille, A., Popović, Z., and Stam, J. Fluid control using the adjoint method. *ACM Trans. Graph.*, 23 (3):449–456, aug 2004.

Mora, M. A. Z., Peychev, M., Ha, S., Vechev, M., and Coros, S. 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.

Nava, E., Zhang, J. Z., Michelis, M. Y., Du, T., Ma, P., Grewe, B. F., Matusik, W., and Katzschmann, R. K. Fast aquatic swimmer optimization with differentiable projective dynamics and neural network hydrodynamic models, 2022.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

Prantl, L., Bonev, B., and Thuerey, N. Generating liquid simulations with deformation-aware neural networks, 2019.

Qiao, Y.-L., Liang, J., Koltun, V., and Lin, M. C. Scalable differentiable physics for learning and control, 2020.

Qiao, Y.-L., Liang, J., Koltun, V., and Lin, M. C. Efficient differentiable simulation of articulated bodies, 2021.

Qiao, Y.-L., Liang, J., Koltun, V., and Lin, M. C. Differentiable simulation of soft multi-body systems, 2022.

Raissi, M., Perdikaris, P., and Karniadakis, G. 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.

Ramos, B., Trost, F., and Thuerey, N. Control of two-way coupled fluid systems with differentiable solvers, 2022.

Robert, A., Yee, T. L., and Ritchie, H. A semi-lagrangian and semi-implicit numerical integration scheme for multilevel atmospheric models. *Monthly Weather Review*, 113 (3), 1985.

Robinson-Mosher, A., Schroeder, C., and Fedkiw, R. A symmetric positive definite formulation for monolithic fluid structure interaction. *Journal of Computational Physics*, 230(4):1547–1566, 2011.

Son, S., Qiao, Y.-L., Sewall, J., and Lin, M. C. Differentiable hybrid traffic simulation, 2022.

Song, C. and Boularias, A. Identifying mechanical models of unknown objects with differentiable physics simulations. In *L4DC*, 2020.

Spielberg, A., Zhao, A., Hu, Y., Du, T., Matusik, W., and Rus, D. Learning-in-the-loop optimization: End-to-end control and co-design of soft robots through learned deep latent representations. In *Neural Information Processing Systems*, 2019.

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

Takahashi, T. and Batty, C. Monolith: a monolithic pressure-viscosity-contact solver for strong two-way rigid-rigid rigid-fluid coupling. *ACM Trans. Graph.*, 39(6), nov 2020.

Takahashi, T. and Batty, C. Frictionalmonolith: a monolithic optimization-based approach for granular flow with contact-aware rigid-body coupling. *ACM Trans. Graph.*, 40(6), dec 2021.

Takahashi, T. and Batty, C. Elastomonolith: A monolithic optimization-based liquid solver for contact-aware elastic-solid coupling. *ACM Trans. Graph.*, 41(6), nov 2022.

Takahashi, T. and Lin, M. C. A geometrically consistent viscous fluid solver with two-way fluid-solid coupling. *Computer Graphics Forum*, 38, 2019.

Takahashi, T., Liang, J., Qiao, Y.-L., and Lin, M. C. Differentiable fluids with solid coupling for learning and control. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(7):6138–6146, May 2021.

Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. Accelerating eulerian fluid simulation with convolutional networks, 2022.

Toussaint, M., Allen, K., Smith, K., and Tenenbaum, J. Differentiable physics and stable modes for tool-use and manipulation planning. In *Robotics: Science and Systems (RSS)*, 2018.

Treuille, A., McNamara, A., Popović, Z., and Stam, J. Keyframe control of smoke simulations. *ACM Trans. Graph.*, 22(3):716–723, July 2003.

Um, K., Hu, X., and Thuerey, N. Liquid splash modeling with neural networks, 2018.

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

Werling, K., Omens, D., Lee, J., Exarchos, I., and Liu, C. K. Fast and feature-complete differentiable physics for articulated rigid bodies with contact, 2021.

Wiewel, S., Becher, M., and Thuerey, N. Latent-space physics: Towards learning the temporal evolution of fluid flow, 2019.

Xie, Y., Franz, E., Chu, M., and Thuerey, N. tempogan: a temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Trans. Graph.*, July 2018.

Xu, J., Chen, T., Zlokapa, L., Foshey, M., Matusik, W., Sueda, S., and Agrawal, P. An end-to-end differentiable framework for contact-aware robot design. In *Robotics: Science and Systems XVII*, RSS2021, July 2021.

Xu, J., Makoviychuk, V., Narang, Y., Ramos, F., Matusik, W., Garg, A., and Macklin, M. Accelerated policy learning with parallel differentiable simulation, 2022.

Zarifi, O. and Batty, C. A positive-definite cut-cell method for strong two-way coupling between fluids and deformable bodies. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2017.

A. Derivation: Gradients of Pressure Solve

The pressure solve or projection step transforms the combined input state $(\boldsymbol{u}_t, \boldsymbol{v}_t, \boldsymbol{x}_t)$ 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 $\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t}\right)$, using the upstream gradients $\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_{t+1}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{t+1}}\right)$. Let's say we require $\frac{\partial \mathcal{L}}{\partial \boldsymbol{q}}$, where $\boldsymbol{q} = \boldsymbol{u}_t, \boldsymbol{v}_t, \boldsymbol{x_t}$. Then, on differentiating the linear system of eqn. (2),

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

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

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

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

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

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:

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{u}_t} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}} + \frac{1}{\rho}\mathbf{M}_F\mathbf{G}\mathbf{s} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}} - \mathbf{J}\mathbf{s} \\
\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t} &= \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_{t+1}} + \Delta t \left[\mathbf{M}_S^{-1}\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{p}\right]^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_{t+1}} + 2\,\Delta t \left[\mathbf{M}_S^{-1}\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{p}\right]^T \mathbf{J}\mathbf{s} + \boldsymbol{v}_t^T\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)\mathbf{s} \\
&\text{where } \mathbf{A}\mathbf{s} = \Delta t \mathbf{J}^T \mathbf{M}_S^{-1} \frac{\partial \mathcal{L}}{\partial \boldsymbol{v_{t+1}}} - \frac{\Delta t}{\rho}\mathbf{G}^T \frac{\partial \mathcal{L}}{\partial \boldsymbol{u_{t+1}}}
\end{aligned}
$$

In our experiments, we set only rigid body velocity v as optimization parameter, for which only $\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}_t}$ is required at the end of every (adjoint) frame. It receives the contribution $\Delta t\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}_t}$ every frame, and so on ignoring the above expression, we introduce an error which is $O(\Delta t^2)$ in two terms and $O(\Delta t)$ in one term. So, using a sufficiently small time step should keep the error small, and thus the performance same.

If instead, we were to include rigid body position x_t as an optimization parameter, we would expect to see a significant performance difference, because of the time step independent term $v_t^T\left(\frac{\partial \mathbf{J}}{\partial \boldsymbol{x}_t}\right)$.

B. Simulator Pseudo Code

Algorithm 1 Differentiable Simulation

Global:

1. Grid Dimensions, (Nx, Ny, Nz)
2. Grid Cell Width, Δx
3. Boundary SDF, $\boldsymbol{\phi}_B$
4. Rigid Body(ies) Geometry + Mass ($\mathbf{M}_S$)
5. Constant external forces i.e. gravity, control forces

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

$\boldsymbol{q}, \boldsymbol{pt} \leftarrow \boldsymbol{q}_0, \boldsymbol{pt}_0$
for $t = 1$ **to** f **do**
  Forward Simulation: $\boldsymbol{q}, \boldsymbol{pt}, \widetilde{\boldsymbol{q}} \leftarrow \Phi(\boldsymbol{q}, \Delta t)$
  Store intermediate results: $\widetilde{\boldsymbol{q}}$
end for
$\boldsymbol{Q} \leftarrow$ getLossDerivative($\mathcal{L}, \boldsymbol{q}$)
for $t = f$ **to** 1 **do**
  Adjoint Simulation: $\boldsymbol{Q} \leftarrow \Psi(\boldsymbol{Q}, \boldsymbol{q}_t, \widetilde{\boldsymbol{q}}_t, \Delta t)$
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 $\boldsymbol{q}_t$, Marker Particles $\boldsymbol{pt}_t$

$\boldsymbol{pt}_{t+1} \leftarrow$ advectParticles($\boldsymbol{pt}_t, \boldsymbol{q}_t, \Delta t$)
$\boldsymbol{q}_{t+\frac{1}{5}} \leftarrow$ advectVelocity($\boldsymbol{q}_t, \Delta t$)
$\boldsymbol{q}_{t+\frac{2}{5}} \leftarrow$ externalForce($\boldsymbol{q}_{t+\frac{1}{5}}, \Delta t$)
$\boldsymbol{\phi}_L \leftarrow$ computeLiquidSDF($\boldsymbol{pt}_{t+1}$)
$\boldsymbol{q}_{t+\frac{3}{5}} \leftarrow$ solvePressure($\boldsymbol{q}_{t+\frac{2}{5}}, \boldsymbol{\phi}_L, \Delta t$)
$\boldsymbol{q}_{t+\frac{4}{5}} \leftarrow$ velocityCorrection($\boldsymbol{q}_{t+\frac{3}{5}}, \Delta t$)
$\boldsymbol{q}_{t+1} \leftarrow$ updateRigidBody($\boldsymbol{q}_{t+\frac{4}{5}}, \Delta t$)

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

Algorithm 3 Adjoint Simulation (Ψ), One Time Step

Input: Adjoint state $\boldsymbol{Q}_{t+1}$, Intermediate state $\widetilde{\boldsymbol{q}}_t$

$\boldsymbol{Q}_{t+\frac{4}{5}} \leftarrow$ adjointUpdateRigidBody($\boldsymbol{Q}_{t+1}, \widetilde{\boldsymbol{q}}_t, \Delta t$)
$\boldsymbol{Q}_{t+\frac{3}{5}} \leftarrow$ adjointVelocityCorrection($\boldsymbol{Q}_{t+\frac{4}{5}}, \widetilde{\boldsymbol{q}}_t, \Delta t$)
$\boldsymbol{Q}_{t+\frac{2}{5}} \leftarrow$ adjointSolvePressure($\boldsymbol{Q}_{t+\frac{3}{5}}, \widetilde{\boldsymbol{q}}_t, \Delta t$)
$\boldsymbol{Q}_t \leftarrow$ adjointAdvectVelocity($\boldsymbol{Q}_{t+\frac{2}{5}}, \widetilde{\boldsymbol{q}}_t, \Delta t$)

Output: Adjoint state $\boldsymbol{Q}_t$