Neural Monte Carlo Fluid Simulation

Pranav Jain
pranavj@usc.edu
University of Southern California
USA

Ziyin Qu
ziyinq@seas.upenn.edu
University of Pennsylvania
USA

Peter Yichen Chen
pyc@csail.mit.edu
MIT CSAIL
USA

Oded Stein
ostein@usc.edu
University of Southern California
USA

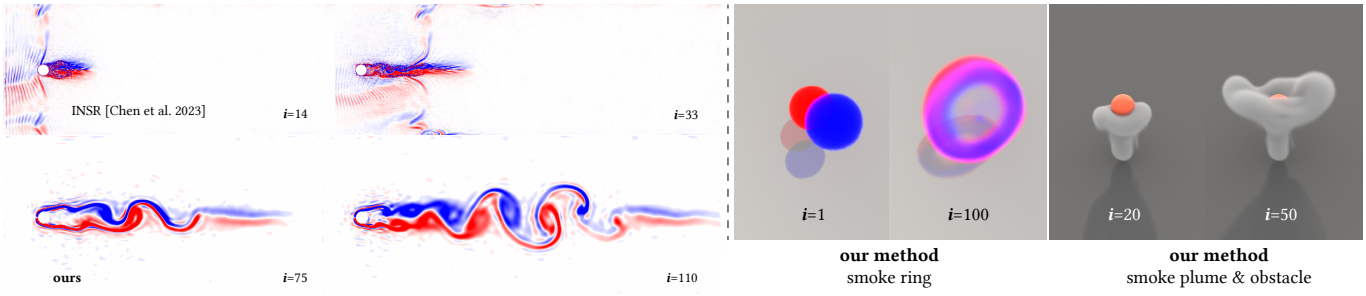


Figure 1: Our method simulates fluids in the presence of obstacles with a combined neural network and Monte Carlo approach to operator splitting for the Navier Stokes equations. With our method, we can simulate important qualitative vorticity-based phenomena, such as vortex shedding in the von Kármán vortex street experiment, previous neural spatial representation papers [Chen et al. 2023b] cannot *(left)*.

ABSTRACT

The idea of using a neural network to represent continuous vector fields (i.e., neural fields) has become popular for solving PDEs arising from physics simulations. Here, the classical spatial discretization (e.g., finite difference) of PDE solvers is replaced with a neural network that models a differentiable function, so the spatial gradients of the PDEs can be readily computed via autodifferentiation. When used in fluid simulation, however, neural fields fail to capture many important phenomena, such as the vortex shedding experienced in the von Kármán vortex street experiment. We present a novel neural network representation for fluid simulation that augments neural fields with explicitly enforced boundary conditions as well as a Monte Carlo pressure solver to get rid of all weakly enforced boundary conditions. Our method, the *Neural Monte Carlo* method (NMC), is completely mesh-free, i.e., it doesn't depend on any grid-based discretization. While NMC does not achieve the state-of-the-art accuracy of the well-established grid-based methods, it significantly outperforms previous mesh-free neural fluid methods on fluid flows involving intricate boundaries and turbulence regimes.

SIGGRAPH Conference Papers '24, July 27–August 01, 2024, Denver, CO, USA

ACM ISBN 979-8-4007-0525-0/24/07
https://doi.org/10.1145/3641519.3657438

CCS CONCEPTS

• **Computing methodologies** → *Computer graphics*; **Physical simulation**; **Neural networks**; • **Mathematics of computing** → **Probabilistic algorithms**.

KEYWORDS

fluid simulation, neural networks, Monte Carlo

ACM Reference Format:
Pranav Jain, Ziyin Qu, Peter Yichen Chen, and Oded Stein. 2024. Neural Monte Carlo Fluid Simulation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers '24 (SIGGRAPH Conference Papers '24), July 27–August 01, 2024, Denver, CO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3641519.3657438

1 INTRODUCTION

Simulating fluids boils down to solving the Navier-Stokes partial differential equation (PDE). The difficulty of solving this PDE has led to a wide variety of *classical* simulation methods that are based on the discretization of the domain with the help of a grid or mesh, where the domain is subdivided into discrete elements with associated degrees of freedom to model a reduced function space. This function is then evolved forward in time.

Meshing the spatial domain to solve PDEs brings with it major challenges, such as the difficulty of adaptivity, handling higher dimensions, and large memory consumption. Neural-network-based physical simulation methods have the potential to overcome these mesh-dependent drawbacks by compactly representing the spatial functions underlying the simulation as neural networks that take a

spatial coordinate as input and output the function value at each point. These networks handle high dimensions by design, requiring no intricate meshing. They are also adaptive by design, allowing for optimizing neural network weights for arbitrary scales and resolutions. Since these networks are also continuously differentiable, the derivatives inherent in the PDEs can be readily computed by simply differentiating the network. This approach has recently found success in cloth simulations [Kairanda et al. 2023], solid simulations [Zesch et al. 2023], and fluid simulations [Chen et al. 2023b]. We refer to the recent course on deep learning and physics simulation by [Du 2023] for more details.

These neural networks, however, can struggle with two things in particular: First, since they often enforce boundary constraints weakly, i.e., by adding a penalty term to the training loss that encourages compliance to the boundary condition when minimized, they can not guarantee that these boundary conditions are always fulfilled after training – this is especially problematic in situations with complicated boundaries. Second, neural network approaches have difficulties resolving challenging situations characterized by turbulent flows (see Figure 1).

Another way to deal with the drawbacks of classical spatial discretization are Monte Carlo methods. A disadvantage of some Monte Carlo methods is their inability to employ Neumann boundary conditions which are common in many real-world scenarios for both fluid velocity and pressure. Another disadvantage is the non-differentiability of most Monte Carlo models – while neural networks are smooth functions defined everywhere, computing the gradient or Hessian of a Monte Carlo model is nontrivial.

In this paper, we aim to push the boundaries of how far we can get **without any grid discretization**. We marry neural networks and Monte Carlo methods to create the *Neural Monte Carlo* (NMC) fluid simulation method that has the advantages of both and none of the drawbacks (see Figure 1 for an overview of our results). We model the *velocity* of our fluids using neural networks with explicitly enforced Dirichlet boundary conditions, and the *pressure* using a Monte Carlo method, combined with caching approaches to speed up computation. While our Neural Monte Carlo method does not achieve the state-of-art accuracy of the well-established gird methods, it is:

- mesh-free (does not require any kind of grid discretization);
- guaranteeing Dirichlet and Neumann boundary conditions are exactly fulfilled;
- modeling complex phenomena such as von Kármán limited-cycle vortex shedding.

We combine the existing Implicit Neural Spatial Representation (INSR) approach by Chen et al. [2023b] with the existing Monte-Carlo method by Sawhney et al. [2023] while enforcing hard boundary conditions. Our contribution lies in combining these existing methods and adding explicit boundary conditions to create our Neural Monte Carlo method that is able to handle obstacles and simulate complex fluid phenomena.

2 RELATED WORK

2.1 Classical fluid simulation

The seminal work by Stam [1999] lays the foundation for fluid simulation in visual computing. Since then, fluid simulations have long been leveraging the projection method originally developed by Chorin [1968], where the highly nonlinear Navier-Stokes equation is integrated in time in an operator-splitting fashion. Visual computing researchers [Bridson 2015] have made a lot of efforts to improve its accuracy and efficiency [Bargteil et al. 2006; Batty et al. 2007; Bender and Koschier 2016; Carlson et al. 2004; De Goes et al. 2015; Fedkiw et al. 2001; Fei et al. 2017; Hyde and Fedkiw 2019; Jiang et al. 2015; Kim et al. 2008; Li et al. 2020; Nabizadeh et al. 2022; Qu et al. 2019; Ren et al. 2014; Ruan et al. 2021; Selle et al. 2008; Solenthaler and Pajarola 2009; Yuksel et al. 2007; Zehnder et al. 2018; Zhu et al. 2013]. Virtually all of these methods leverage classical basis functions to discretize the spatial vector field after temporal discretization. These basis functions include finite difference [Godunov and Bohachevsky 1959], finite volume [Moukalled et al. 2016], smoothed-particle hydrodynamics [Müller et al. 2003], particle-in-cells [Zhu and Bridson 2005], and spectral methods [De Witt et al. 2012]. These traditional discretizations, however, face challenges in handling high-dimensional inputs, a large number of memory consumptions, and difficulty of adaptivity [Ando et al. 2013; Museth 2013; Setaluri et al. 2014].

2.2 Neural fluid simulation

An alternative basis function is the neural network. With the recent trend in implicit neural representations (also known as neural fields, coordinate-based neural representations) [Chen and Zhang 2019; Mescheder et al. 2019; Park et al. 2019; Sitzmann et al. 2020] and physics-informed neural networks [Raissi et al. 2019; Wang et al. 2021], visual computing researchers have explored various ways to leverage neural networks to solve PDEs arising in geometry processing [Chetan et al. 2023; Dodik et al. 2023; Yang et al. 2021], topology optimization [Zehnder et al. 2021], cloth modeling [Kairanda et al. 2023; Santesteban et al. 2022], contact handling [Zesch et al. 2023], soft bodies [Chang et al. 2023], and elastoplasticity [Chen et al. 2023a; Zong et al. 2023].

Unlike classical fluid simulations, neural fluid simulations excel at adaptivity thanks to the grid-free nature of neural networks but suffer significant drawbacks in speed and accuracy. Chen et al. [2023b] use a neural network based on SIREN [Sitzmann et al. 2020] to represent the velocity and pressure in a classical operator-splitting fluid simulation, forgoing the grid that is usual in non-neural methods. Deng et al. [2023] leverages a hybrid neural-grid formulation to model the spatiotemporal flow map and achieves greater fidelity than the classic grid methods. Nevertheless, as discussed by Chuang and Barba [2022], capturing the signature von Kármán vortex street without any training data and with only a physics-informed loss remains challenging for neural fluid simulation methods that are entirely grid-free and mesh-free. Similar to our approach, the recent work of Wang et al. [2023] use neural networks to successfully model the von Kármán vortex street. Their approach utilizes non-dimensionalization of the PDEs, advanced network architectures, and special training schemes. By contrast,

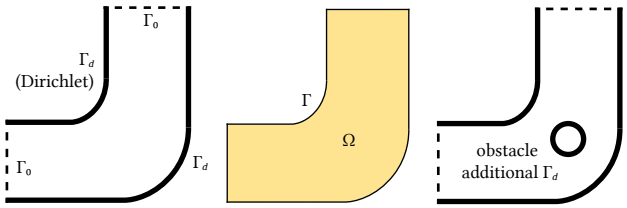


Figure 2: An overview over our domain and naming conventions. Our domain Ω is bounded by Γ. The parts of the boundary where the Dirichlet conditions are enforced are Γ_d. Obstacles are just additional Dirichlet boundaries.

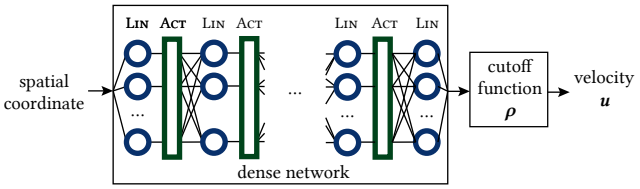


Figure 3: Our velocity neural network takes in a spatial coordinate in $\mathbb{R}^d$, then runs it through a number of alternating fixed-width linear layers and SIREN activation layers, then multiplies the output with the cutoff function, and returns a vector in $\mathbb{R}^d$.

our method employs the operator splitting technique to handle PDEs while relying on conventional network architectures and off-the-shelf training schemes.

2.3 Monte Carlo methods

The Monte Carlo technique is an old method to compute integrals and solve PDEs by randomly sampling the integration domain [Metropolis and Ulam 1949]. Monte Carlo methods, traditionally used for rendering in computer graphics, have recently become popular in geometry processing and simulation as well to solve spatial PDEs without meshes or grids [Li et al. 2023; Miller et al. 2023; Sawhney and Crane 2020; Sawhney et al. 2022, 2023]. Methods like the one by Sawhney et al. [2023] can solve elliptic PDEs by starting random walks from an arbitrary evaluation point, and stopping the walk when some criterion (such as a boundary encounter or an absorption site) is reached.

Monte Carlo methods have also started seeing applications in fluid simulation, where they circumvent classical methods' need for spatial grids or meshes, such as the work of Rioux-Lavoie et al. [2022] (which cannot model zero Neumann boundary conditions for the pressure field). In our work, we use the Monte Carlo method of Sawhney et al. [2023] to model the pressure field of our fluid simulation – this enables us to impose Neumann boundary conditions.

3 PROBLEM STATEMENT

Consider a domain $\Omega \subseteq \mathbb{R}^d$ $(d = 2, 3)$ with boundary $\Gamma = \partial\Omega$. We model our fluid using the *Euler equation*, the inviscid case of the general Navier-Stokes equations (see, e.g., the work of Stam [1999]):

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nabla p + \mathbf{g}\,, \tag{1}$$

where

- $\mathbf{u} : \Omega \to \mathbb{R}^d$ is the *velocity* of the fluid;
- $p : \Omega \to \mathbb{R}$ is the *pressure* of the fluid;
- and $\mathbf{g} : \Omega \to \mathbb{R}^d$ are external forces.

In addition, we have an *incompressibility condition* on our velocity,

$$\nabla \cdot \mathbf{u} = 0. \tag{2}$$

Our boundary Γ is divided into impassable (Γ_d) and passable (Γ_0) sections, $\Gamma = \Gamma_d \cup \Gamma_0$. On impassable sections, we apply a Dirichlet (no-through) boundary condition to the velocity,

$$\mathbf{u}(x) \cdot \mathbf{n}_\Gamma(x) = 0 \quad \forall x \in \Gamma_d,\ \mathbf{n}_\Gamma \text{ boundary normal}, \tag{3}$$

reflecting the fact that fluid can not enter or leave at all through Γ_d, and is not constrained in any way on Γ_0.[1] On the entire boundary Γ, we apply Neumann boundary conditions to the pressure,

$$\mathbf{n}_\Gamma(x) \cdot \nabla p(x) = 0 \quad \forall x \in \Gamma,\ \mathbf{n}_\Gamma \text{ boundary normal}. \tag{4}$$

Moreover, at some points in the domain, we can artificially inject velocity if the problem demands it. Figure 2 shows how our domain and its boundaries are defined.

We specify initial conditions $\mathbf{u}^0$, p^0 for the Euler equation (1), and then evolve them in time using the PDE.

4 METHOD

The Euler equation (1) is notoriously difficult to discretize. In this section, we explain our method, which combines neural networks and Monte Carlo methods and applies them to the classic operator splitting approach to fluid simulation. Unlike many machine learning fluids works [Kim et al. 2019], our method **does not require any training data**, neither from other solves nor from experiments. It works just like the classic grid-based solver. The only difference between ours and a classic grid-based solver is that ours does not require grid discretization of any kind.

4.1 Operator-splitting time integration: the projection method

Assuming a timestep size of Δt, we evolve our velocity and pressure in an operator splitting fashion. Specifically, we leverage the projection method originally developed by Chorin [1968], as described in the work of [Chen et al. 2023b; Stam 1999]. It consists of three steps *advection*, *pressure projection*, and *velocity correction*.

Advection. We first advect the velocity forward in time with the formula

$$\mathbf{u}^{i+1}_{\text{adv}}(x) = \mathbf{u}^i(x - \Delta t\ \mathbf{u}^i(x))\,, \tag{5}$$

subject to Dirichlet boundary conditions (3).

We can formulate this as the optimization problem

$$\begin{aligned} \mathbf{u}^{i+1}_{\text{adv}} &= \operatorname*{argmin}_{\mathbf{u}} E_{\text{adv}}(\mathbf{u}, \mathbf{u}^i), \\ E_{\text{adv}}(\mathbf{u}, \mathbf{u}^i) &= \left\|\mathbf{u} - \mathbf{u}^i_b\right\|_2^2,\ \mathbf{u}^i_b(x) = \mathbf{u}^i(x - \Delta t\ \mathbf{u}^i(x))\,, \end{aligned} \tag{6}$$

where the norm is the L^2 norm of functions over the domain Ω, $\|f\|_2^2 = \int_\Omega \|f\|^2\, dx$.

[1] In some experiments, we also set all of the velocity, including the tangential part, to zero at some parts of the boundary.

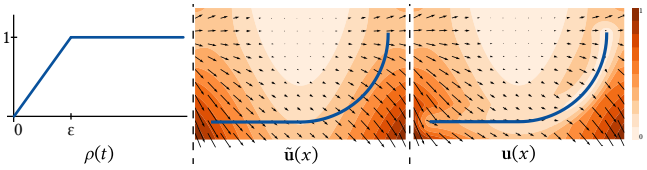


Figure 4: The cutoff function ρ moves continuously from 0 to 1. We plot the output of the neural network $\tilde{\mathbf{u}}$ multiplied with the cutoff function ρ. The resulting function $\mathbf{u}$ fulfills Dirichlet boundary conditions.

Pressure projection. We then evolve the pressure by solving the pressure projection Poisson equation,

$$\Delta p^{i+1}(x) = \nabla \cdot \mathbf{u}_{\text{adv}}^{i+1}(x)\,, \tag{7}$$

subject to pure Neumann boundary conditions (4).

Velocity correction. Finally, we ensure that our velocity fulfills the incompressibility condition with the velocity correction step,

$$\mathbf{u}^{i+1}(x) = \mathbf{u}_{\text{adv}}^{i+1}(x) - \nabla p^{i+1}(x)\,, \tag{8}$$

subject to Dirichlet boundary conditions (3).

This can be done by solving the optimization problem

$$\begin{aligned} \mathbf{u}^{i+1}(x) &= \operatorname*{argmin}_{\mathbf{u}} E_{\text{corr}}(\mathbf{u}, \mathbf{u}_{\text{adv}}^{i+1}, p^{i+1}), \\ E_{\text{corr}}(\mathbf{u}, \mathbf{u}_{\text{adv}}^{i+1}, p^{i+1}) &= \left\| \mathbf{u} - \left(\mathbf{u}_{\text{adv}}^{i+1} - \nabla p^{i+1} \right) \right\|_2^2 , \end{aligned} \tag{9}$$

where the norm is an L^2 norm of functions over the domain Ω.

Details on imposing Dirichlet boundary conditions for the advection and velocity correction step and imposing pure Neumann boundary condition for the pressure projection step is discussed in Section 4.4 and 4.3 respectively.

Remark: Note that this approach has only discretized the Euler equation in the *time domain*. No spatial discretization has happened. Indeed, this time discretization is compatible with any spatial representation of the continuous velocity field and the pressure field. Next, we introduce our spatial representation that is free from any kind of grid discretization.

4.2 Velocity

Adapting the work of [Chen et al. 2023b], we represent the velocity as a neural network that takes a spatial coordinate $x \in \mathbb{R}^d$ as input and returns a vector $\mathbf{u}(x) \in \mathbb{R}^d$.

Velocity network. The network consists of alternating linear layers and SIREN activation layers [Sitzmann et al. 2020], where all linear layers have the same width. The time-discretized vector fields are parametrized by the linear layer parameters θ; we write $\mathbf{u}_\theta^i \sim \mathbf{u}_i$. The width of the linear layers and the depth of the network can be varied depending on user preference. Figure 3 shows a schematic overview of the velocity network. In general, more and wider layers are required to capture fine details. Figure 13 shows an example of a shallow network unable to resolve a vortex ring, while the dense network displays the correct behavior.

The implicit neural representation $\mathbf{u}_\theta^i \sim \mathbf{u}_i$ has a multitude of advantages over traditional spatial representations such as grids or meshes [Chen et al. 2023b].

- Every point in space can be exactly sampled, without relying on interpolation.
- The memory requirement of the neural network is not related to the density of spatial samples, it only depends on the width and depth of the network.
- The network is inherently differentiable, and exact gradients can be computed at every point using autodifferentiation. The gradients are not approximations that merely converge to the true gradients.

Training. Training a neural network is analogous to minimizing the energy functions defined in Section 4.1. The velocity neural network must be trained twice in every time step: once for the advection step $\mathbf{u}_{\theta,\text{adv}}^{i+1}$, and once for the velocity correction step $\mathbf{u}_\theta^{i+1}$. We use similar, but slightly different training schemes for each.

To train the advected velocity field $\mathbf{u}_{\theta,\text{adv}}^{i+1}$ we use Adam for a fixed number of iterations set as a parameter. At each training step, we randomly sample k random points $x_1, \ldots, x_k \in \Omega$, and use these points to evaluate the integral in the advection loss E_{adv} (6):

$$\mathsf{E}_{\text{adv}}(\mathbf{u}) = \sum_j \left\| \mathbf{u}_\theta(x_j) - \mathbf{u}_\theta^i(x_j - \Delta t\, \mathbf{u}_\theta^i(x_j)) \right\|^2 . \tag{10}$$

This loss is used to backpropagate during the training. Note that it is easy to sample $\mathbf{u}_\theta^i(x_j)$ at any arbitrary point in the domain since the neural network accepts any spatial coordinate as input. The parameters we use in training are listed in the supplemental material.

To train the corrected velocity field $\mathbf{u}_\theta^{i+1}$, we use a similar strategy employing Adam and a fixed number of iterations. We train on the correction loss

$$\mathsf{E}_{\text{corr}}(\mathbf{u}) = \sum_j \left\| \mathbf{u}(x_j) - \left(\mathbf{u}_{\theta,\text{adv}}^{i+1}(x_j) - \nabla p^{i+1}(x_j) \right) \right\|^2 . \tag{11}$$

Our sampling strategy is slightly different when training on the loss (11). Both the neural network based $\mathbf{u}_{\theta,\text{adv}}^{i+1}$ and the Monte Carlo based $p^{i+1}(x_j)$ (Explained in Section 4.3) can be evaluated at any arbitrary point in the domain. Re-evaluating $p^{i+1}(x_j)$ at every training step, however, would hurt performance significantly since it would require the Monte Carlo pressure solver to run for every training step. To that end, to train the corrected velocity network, we uniformly sample a large number of random points $x_1^*, \ldots, x_{k^*}^* \in \Omega$ before we start the training procedure at each timestep, and then randomly subsample a set of actual training samples $(x_1, \ldots, x_k) \subseteq \left(x_1^*, \ldots, x_{k^*}^*\right)$ for each training timestep. That way we can precompute our pressure samples, i.e compute $p^{i+1}(x_j^*)\ \forall x_j^* \in \{x_1^*, \ldots, x_{k^*}^*\}$ which significantly improves performance while still ensuring effective random sampling during the training of the corrected velocity field.

Weight reset. We reset the weights of the neural network at the beginning of each timestep and do not reuse the previous timestep as initialization, as we find that that significantly increases the noisiness of $\mathbf{u}$. We suspect this to be due to overfitting during training. Figure 5 shows a comparison.

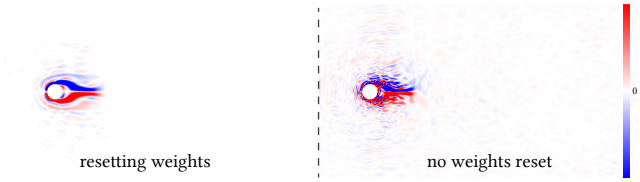


Figure 5: Vorticity plot for the von Kármán vortex street experiment (Section 5.2) with *(left)* and without *(right)* resetting weights at every timestep. We find that resetting weights significantly reduces noise.

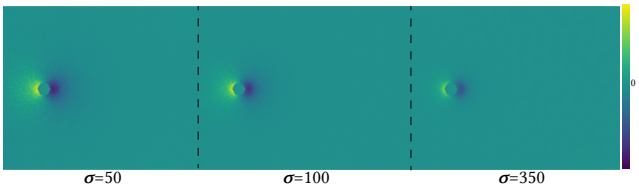


Figure 6: The pressure in the von Kármán vortex street experiment for multiple screening weights. Increasing the screening weight reduces the noise produced by the WoSt method, while not changing the qualitative behavior of the simulation much. We thus tend to use larger screening weights.

4.3 Pressure

Solving the Poisson equation for the pressure projection is often a computation bottleneck in traditional grid methods [McAdams et al. 2010]. In fact, it is also a challenge in neural methods where the large errors in the Poisson solve propagate to the rest of the simulation [Chen et al. 2023b] (see Figure 7). To address this issue, we decide to use Monte Carlo methods to solve the pressure projection step to arbitrary precision. Like neural networks, Monte Carlo methods also do not require grid or mesh discretization of any kind.

Walk on Stars. The *Walk on Stars* (WoSt) Monte Carlo method [Miller et al. 2023; Sawhney et al. 2023] solves the Poisson Equation with Neumann boundary conditions,

$$-\Delta p = f\,, \tag{12}$$

using a random walk approach. Given the right-hand side f, WoSt computes the solution $p(x)$ to (12) by sending *random walkers* from x on a random walk until they hit the domain boundary, where they are either absorbed or reflected back into the domain. The Green's function of the Poisson equation and the Poisson kernel are then used to accumulate the data gathered during the random walk and compute a solution to the Poisson equation. Using this tactic, and a variety of performance enhancements, we can solve the *pressure projection* step(7) (which consists only of a Poisson equation) – see the articles [Miller et al. 2023; Sawhney et al. 2023] for more details.

The Walk on Stars approach assures that we can evaluate the pressure field p^i at *every* point in the domain, and the Neumann boundary conditions are guaranteed to be fulfilled by construction. WoSt's main advantage – faithful PDE solution on domains with intricate boundaries – transfers to our method and enables us to faithfully model even complicated obstacles.

Screened Poisson equation. WoSt can not actually directly solve the Poisson equation (12) with pure Neumann boundary exactly, as is the case with our pressure p^i, since random walkers are reflected at Neumann boundaries, which would result in infinite reflection. For such boundaries, WoSt employs a regularization technique that boils down to solving the screened Poisson equation

$$-\Delta p - \sigma p = f, \tag{13}$$

for a screening scalar $\sigma > 0$, resulting in an approximate solution of (12) depending on the screening parameter. We find that this regularization still results in believable fluid simulation. An exploration of the effect of different screening weights can be found in Figure 6.

Gradients. We do not actually need the pressure, p^i for the next step (the velocity correction step (11)). We only need the *gradient* ∇p^{i+1}. The WoSt method can return the exact gradient of the solution as well using the random walker strategy – no numerical approximations of the gradient are needed. This is done by using the gradients of the Green's function and the Poisson kernel inside WoSt returning the gradients of the solution.

Encoding the right-hand side. In our implementation, we pass the right-hand side of the pressure projection step (7), $\nabla \cdot \mathbf{u}_{\text{adv}}^{i+1}(x)$, to WoSt [Sawhney et al. 2023] (using the official implementation [Sawhney and Miller 2023]), by evaluating it on a spatial grid. This grid is only needed because of the interface of Sawhney et al. [2023]'s WoSt implementation [Sawhney and Miller 2023] – it is not a theoretical limitation of our method and can be chosen independently of the NNs sizes.

4.4 Boundary conditions

While our pressure p^i fulfills Neumann boundary conditions by construction, our velocity (so far a neural network that can return any value in $\mathbb{R}^d$), does not fulfill any boundary conditions yet. We make sure that the boundary conditions for the velocity are fulfilled by multiplying the output of the neural network, $\tilde{\mathbf{u}}$, by a special cutoff function $\rho : \mathbb{R} \to [0, 1]$:

$$\begin{aligned} \tilde{\mathbf{u}}_{\mathbf{n}}(x) &= (\mathbf{n}(b_\Gamma(x)) \cdot \tilde{\mathbf{u}}(x))\, \mathbf{n}(b_\Gamma(x)) \\ \tilde{\mathbf{u}}_{\mathbf{t}}(x) &= \tilde{\mathbf{u}} - \tilde{\mathbf{u}}_{\mathbf{n}} \\ \mathbf{u}(x) &= \tilde{\mathbf{u}}_{\mathbf{t}}(x) + \rho(s_d(x))\tilde{\mathbf{u}}_{\mathbf{n}}(x)\,, \end{aligned} \tag{14}$$

where $s_d : \mathbb{R}^d \to \mathbb{R}$ is the distance function of the Dirichlet boundary segments Γ_d, and $b_\Gamma(x)$ is the closest point on Γ to x. We then input the multiplied velocity $\mathbf{u}$ into the losses (10), (11) during training.

If the cutoff function fulfills $\rho(0) = 0, \rho(\varepsilon) = 1, \rho(t > \varepsilon) = 1$, then (14) guarantees that

- $\mathbf{n}_\Gamma(x) \cdot \mathbf{u}(x) = 0\ \forall x \in \Gamma_d$, i.e., $\mathbf{u}$ fulfills the Dirichlet boundary conditions;
- $\mathbf{u}(x) = \tilde{\mathbf{u}}(x)\ \forall x$ farther than ε from Γ_d.

There is a variety of functions that fulfill these conditions. We pick the simplest choice,

$$\rho(t) = \begin{cases} \frac{t}{\varepsilon} & t < \varepsilon \\ 1 & t \geq \varepsilon \end{cases}\,. \tag{15}$$

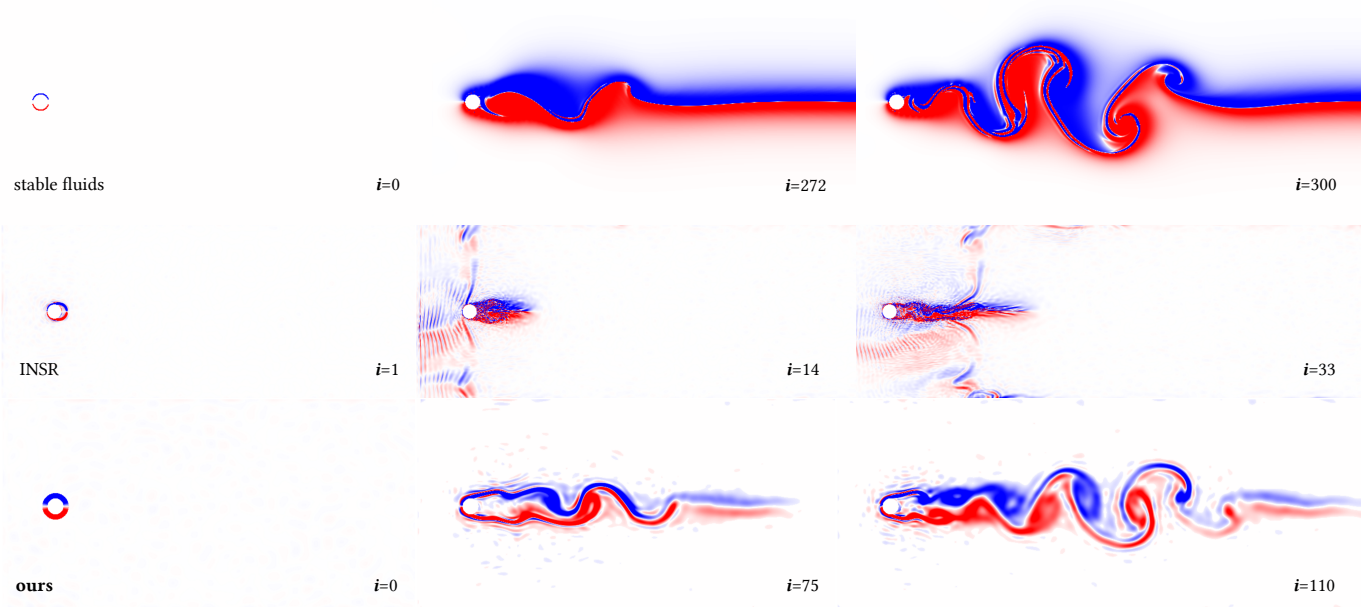


Figure 7: The von Kármán vortex street experiment simulated with stable fluids [Stam 1999], INSR [Chen et al. 2023b], and with our method. While INSR degenerates into noise very quickly, our gridless method shows vortex shedding behavior which qualitatively matches grid-based stable fluids.

This function can be differentiated once, which is enough to autodifferentiate $\mathbf{u}$ for the purposes of computing E_{adv} and E_{corr}. For higher-order losses, one has to make sure that the function is differentiable enough, which requires higher-order polynomials.

This cutoff strategy guarantees that our velocity always fulfills the Dirichlet boundary condition, while not modifying the output of the neural network $\tilde{\mathbf{u}}$ too far away from the boundary. While we have a parameter (ϵ), this parameter does not influence whether the boundary conditions are fulfilled – it influences the approximation quality of the neural network. This is different from the INSR method [Chen et al. 2023b], which uses weak boundary conditions enforced via penalty. Figure 4 shows how our cutoff functions work to ensure that $\mathbf{u}^i$ always fulfills the Dirichlet boundary conditions.

Similar strategies are employed by Berg and Nyström [2018]; Liu et al. [2022], and there are other promising approaches for enforcing explicit boundary conditions in neural networks [Chen et al. 2024; Sukumar and Srivastava 2022; Zhong et al. 2023].

4.5 Algorithm

A pseudocode implementation of our method can be found in Algorithm 1. It follows the operator splitting approach of Section 4.1 with our neural network velocity and Monte Carlo pressure.

We implement the neural network and its training in Pytorch for the velocity and adapt the official C++ implementation of WoSt for the pressure. The experiments were run on an Intel i7 5.2GHz with a NVIDIA RTX 4080 GPU.

Algorithm 1 Method overview

function NMC($\mathbf{u}^0$, p^0)
  for i = 0, ..., n **do**
    Train $\mathbf{u}_{\theta,\text{adv}}^{i+1}$ by minimizing E_{adv} (10)
      ▷ randomly sample x_j in Ω
    Pick random points $\left(x_j^*\right)_j$ in Ω
    evaluate $p^{i+1}(x_j^*)$ $\forall j$ using Monte Carlo.
    Train $\mathbf{u}_\theta^{i+1}$ by minimizing E_{corr} (11)
      ▷ randomly sub-sample x_j from $\left(x_j^*\right)_j$
  return $\mathbf{u}_\theta^n, p^n$

5 EXPERIMENTS AND RESULTS

In this section, we present the results of our method applied to a variety of fluid simulation problems. Details on the initial conditions, boundary conditions, and parameters used can be found in the supplemental material.

5.1 Taylor-Green

We use our method to simulate the classical Taylor-Green flow going back to the work of Taylor and Green [1936]. This experiment is of particular interest because an analytical solution is known – the result should remain stationary. We set up the initial condition to be

$$\mathbf{u}^0(x, y) = (\sin(x)\cos(y), -\cos(x)\sin(y)) \,. \tag{16}$$

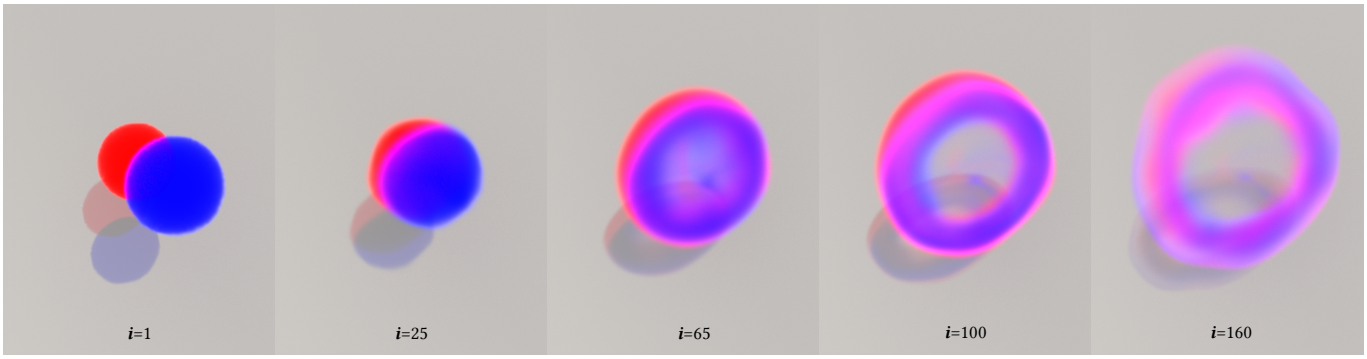


Figure 8: Two balls of smoke collide, resulting in the creation of a smoke ring. Our method is capable of simulating qualitative vorticity-dominant phenomena such as the formation of vortices spreading out to form a smoke ring.

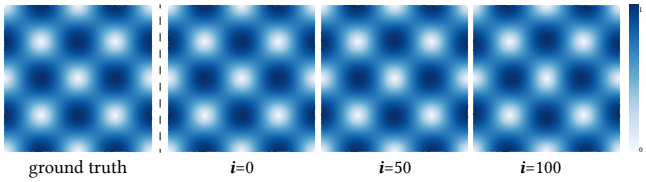


Figure 9: Density plot of the Taylor-Green experiment. The stationary ground-truth density *(left)* is well-preserved by our method for multiple timesteps *(right)*.

The results can be seen in Figure 9. Our method satisfactorily preserves the stationary flow.

5.2 Von Kármán vortex street

In the von Kármán vortex street example, flow enters a tube from the left where it hits a cylindrical obstacle. On this obstacle, we set *all* boundary velocity to zero (not just the usual conditions on Γ_d) – both tangential and normal components of the velocity are set to zero. The flow around the obstacle creates a plume of vorticity behind the obstacle, and after some time we begin to experience *vortex shedding*, where little pockets of vorticity detach from the main flow. The von Kármán vortex street is of particular interest to us because it is well known in the neural fluids literature that neural networks fail to capture the vortex shedding phenomena [Chuang and Barba 2022].

Figure 7 shows our method applied to the two-dimensional von Kármán vortex street experiments. Unlike the previous work of Chen et al. [2023b], our implementation exhibits vortex shedding, much like the dense classical reference [Stam 1999]. We employ a rectangular domain that is open on the left and right and has hard Dirichlet boundaries on the top and bottom. The inflow velocity at the left is set to be constant.

5.3 Smoke

We conducted three different smoke experiments: a rising smoke plume, a rising smoke plume with a spherical object, and two smoke balls colliding to form a smoke ring. In all of these examples, we visualize the density of the smoke.

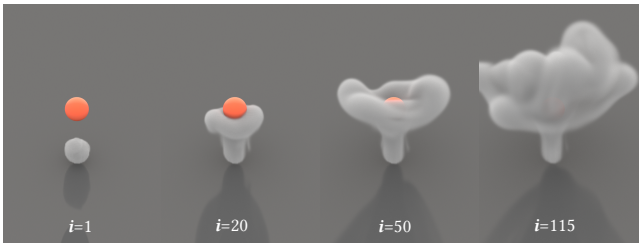


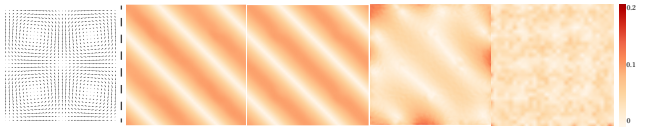
Figure 10: A jet of smoke (without buoyancy) colliding with a spherical obstacle. A video of this simulation can be found in the supplemental material.

Smoke plume with obstacle. In Figure 10 we simulate a jet of smoke (without buoyancy) shooting from the floor and colliding with a spherical obstacle where both normal and tangential components of the boundary velocity are zeroslip. This example demonstrates our method's ability to deal with boundary conditions precisely, and thus interact with obstacles. Similar to the rising smoke plume, the experiment is set in a cubical container and initialized with smoke in a spherical ball with random velocities.

Smoke ring. In this experiment (Figure 8), two differently-colored smoke balls collide with each other. We observe the expected behavior (see, e.g., the work of Qu et al. [2019]) – a smoke ring forming and slowly spreading out. The experiment is again set in a cubical domain.

5.4 Quantitative comparison to previous work

In Figure 11 we perform a quantitative comparison of our method with previous methods using the Taylor-Green example since an analytical solution is readily available. Our method has lower error than previous neural fluid simulation methods [Chen et al. 2023b; Raissi et al. 2019; Wang et al. 2021] on the Taylor-Green example. For a fair comparison, we ran all methods on networks of comparable sizes ($\sim 2^{11}$ trainable weights) with the respective method's default parameters (including stopping conditions). Note that we can not perform a quantitative comparison for other examples due to lack of an analytical or ground truth solution.



L^2 error	PINN	piDeepONet	INSR	**ours**
$t = 1$	$3.955 \cdot 10^{-3}$	$3.941 \cdot 10^{-3}$	$8.000 \cdot 10^{-4}$	$\mathbf{1.835 \cdot 10^{-4}}$
$t = 25$	$3.950 \cdot 10^{-3}$	$3.945 \cdot 10^{-3}$	$9.737 \cdot 10^{-4}$	$\mathbf{3.505 \cdot 10^{-4}}$
$t = 50$	$3.948 \cdot 10^{-3}$	$3.947 \cdot 10^{-3}$	$1.387 \cdot 10^{-3}$	$\mathbf{8.957 \cdot 10^{-4}}$

Figure 11: The Taylor-Green stationary flow velocity *(left)*, and errors for various methods *(right)*: PINN [Raissi et al. 2019], piDeepONet [Wang et al. 2021], INSR [Chen et al. 2023b], and ours. Note the INSR error near the boundary which we avoid using explicit boundary conditions for u and p.

5.5 Convergence

In Figure 14 we show a plot of how the log of the loss varies over training iterations for both the advection and velocity correction step. The plot is computed for the first timestep of the von Kármán vortex street example (Figure 7). The loss value gradually decreases, showing convergence. We choose a fixed number of iterations for training, as oscillations increase over time.

5.6 Timings

Table 1 reports the running time for each operator splitting step – advection, projection, and correction. The timings are computed for a single timestep. Since we use the same number of training iterations for each timestep of the simulation, the runtime for each operator splitting step remains similar for every timestep.

6 LIMITATIONS

While our method improves the state of the art on neural networks for spatial discretization, it inherits some of its limitations [Chen et al. 2023b]. Our method does not yet match classical grid-based methods in performance or accuracy. This is a well-known limitation of physics-informed neural networks [Chuang and Barba 2022; Grossmann et al. 2023]. Our method also fails to preserve symmetry in some cases, as can be seen in our simulation of a rising smoke plume without buoyancy (see Figure 12). Like other neural-network-based methods, our method requires intensive parameter tuning, the details of which are listed in the supplement. For example, instead of classical parameters like grid size, we have to tune network widths and depths. At last, we currently do not explicitly enforce Neumann boundary conditions on the velocity, but let the network's inherent smoothness decide the value of the velocity field where Dirichlet conditions are not enforced. This limitation could be overcome in future work with an approach similar to the work by Berg and Nyström [2018].

Table 1: Table showing the computational time (in seconds) of each operator split substep for each example. The timings are computed for a single simulation timestep.

	adv.	proj.	corr.
taylorgreen	15.80s	0.561s	22.57s
von Kármán	18.37s	13.12s	21.60s
smoke plume	29.34s	4.16s	28.62s
smoke with obstacle	26.78s	4.13s	27.46s
smoke ring	16.32s	4.12s	20.40s

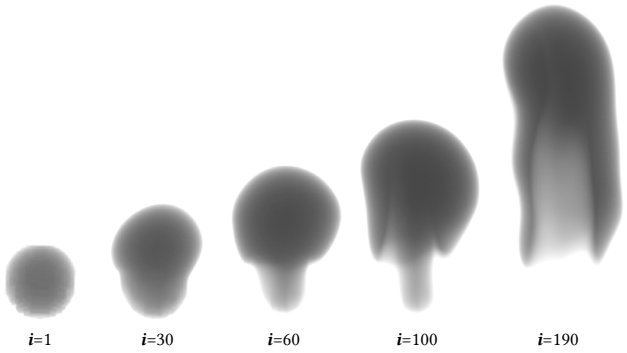


Figure 12: Failure case: our method fails to preserve the symmetry inherent in the problem during the simulation. Here we simulate a rising smoke plume (without buoyancy).

7 CONCLUSION

We have introduced a neural Monte Carlo method for simulating fluids that can handle intricate boundaries and model phenomena such as the von Kármán vortex street that can not be satisfactorily modeled with previous neural-only methods [Chen et al. 2023b; Chuang and Barba 2022]. A clear direction for future work is improving the runtime of our method, e.g., leveraging techniques from meta-learning [Dupont et al. 2022] and reduced-order modeling [Chen et al. 2022]. Currently, we are training the neural network from scratch at every timestep. Another way to reduce the computational time could be to compute the neural network weights at the next timestep by interpolating the weights from the previous timesteps instead of retraining. Another promising direction for future work is to try to improve the (currently very simple) architecture of the neural network to get accuracies similar to classical higher-order methods. Promising network architectures include the recent works of Finzi et al. [2023]; Kovachki et al. [2023]. Future research could also reduce the amount of noise that is generated by the Monte Carlo pressure solver.

ACKNOWLEDGMENTS

We thank Bailey Miller and Rohan Sawhney for technical help with WoSt [Sawhney et al. 2023]. We thank Rundi Wu and Honglin Chen for technical help with previous work. We thank Sifan Wang and Shyam Sankaran for technical help with Wang et al. [2023].

REFERENCES

Ryoichi Ando, Nils Thürey, and Chris Wojtan. 2013. Highly adaptive liquid simulations on tetrahedral meshes. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

Adam W Bargteil, Tolga G Goktekin, James F O'brien, and John A Strain. 2006. A semi-Lagrangian contouring method for fluid simulation. *ACM Transactions on Graphics (TOG)* 25, 1 (2006), 19–38.

Christopher Batty, Florence Bertails, and Robert Bridson. 2007. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 100–es.

Jan Bender and Dan Koschier. 2016. Divergence-free SPH for incompressible and viscous fluids. *IEEE Transactions on Visualization and Computer Graphics* 23, 3 (2016), 1193–1206.

Jens Berg and Kaj Nyström. 2018. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing* 317 (2018), 28–41.

Robert Bridson. 2015. *Fluid simulation for computer graphics*. CRC press.

Mark Carlson, Peter J Mucha, and Greg Turk. 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. *ACM Transactions on Graphics (TOG)* 23, 3 (2004), 377–384.

Yue Chang, Peter Yichen Chen, Zhecheng Wang, Maurizio M Chiaramonte, Kevin Carlberg, and Eitan Grinspun. 2023. LiCROM: Linear-Subspace Continuous Reduced Order Modeling with Neural Fields. In *SIGGRAPH Asia 2023 Conference Papers*. 1–12.

Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2023b. Implicit neural spatial representations for time-dependent PDEs. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*. Article 202, 16 pages.

Peter Yichen Chen, Maurizio M Chiaramonte, Eitan Grinspun, and Kevin Carlberg. 2023a. Model reduction for the material point method via an implicit neural representation of the deformation map. *J. Comput. Phys.* 478 (2023), 111908.

Peter Yichen Chen, Jinxu Xiang, Dong Heon Cho, Yue Chang, GA Pershing, Henrique Teles Maia, Maurizio M Chiaramonte, Kevin Carlberg, and Eitan Grinspun. 2022. CROM: Continuous reduced-order modeling of PDEs using implicit neural representations. *arXiv preprint arXiv:2206.02607* (2022).

Simin Chen, Zhixiang Liu, Wenbo Zhang, and Jinkun Yang. 2024. A Hard-Constraint Wide-Body Physics-Informed Neural Network Model for Solving Multiple Cases in Forward Problems for Partial Differential Equations. *Applied Sciences* 14, 1 (2024).

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5939–5948.

Aditya Chetan, Guandao Yang, Zichen Wang, Steve Marschner, and Bharath Hariharan. 2023. Accurate Differential Operators for Hybrid Neural Fields. *arXiv preprint arXiv:2312.05984* (2023).

Alexandre Joel Chorin. 1968. Numerical Solution of the Navier-Stokes Equations. *Math. Comp.* 22, 104 (1968), 745–762.

Pi-Yueh Chuang and Lorena A Barba. 2022. Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration. *arXiv preprint arXiv:2205.14249* (2022).

Fernando De Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Trans. Graph.* 34, 4 (2015), 50–1.

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid simulation using laplacian eigenfunctions. *ACM Transactions on Graphics (TOG)* 31, 1 (2012), 1–11.

Yitong Deng, Hong-Xing Yu, Diyang Zhang, Jiajun Wu, and Bo Zhu. 2023. Fluid Simulation on Neural Flow Maps. *ACM Trans. Graph.* 42, 6, Article 248 (2023).

Ana Dodik, Oded Stein, Vincent Sitzmann, and Justin Solomon. 2023. Variational Barycentric Coordinates. *ACM Transactions on Graphics* (2023). https://doi.org/10.1145/3618403

Tao Du. 2023. Deep Learning for Physics Simulation. In *ACM SIGGRAPH 2023 Courses*. Article 6, 25 pages.

Emilien Dupont, Hyunjik Kim, SM Eslami, Danilo Rezende, and Dan Rosenbaum. 2022. From data to functa: Your data point is a function and you can treat it like one. *arXiv preprint arXiv:2201.12204* (2022).

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 15–22.

Yun Fei, Henrique Teles Maia, Christopher Batty, Changxi Zheng, and Eitan Grinspun. 2017. A multi-scale model for simulating liquid-hair interactions. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–17.

Marc Anton Finzi, Andres Potapczynski, Matthew Choptuik, and Andrew Gordon Wilson. 2023. A Stable and Scalable Method for Solving Initial Value PDEs with Neural Networks. In *The Eleventh International Conference on Learning Representations*.

Sergei K Godunov and I Bohachevsky. 1959. Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematičeskij sbornik* 47, 3 (1959), 271–306.

Tamara G. Grossmann, Urszula Julia Komorowska, Jonas Latz, and Carola-Bibiane Schönlieb. 2023. Can Physics-Informed Neural Networks beat the Finite Element Method? arXiv:2302.04107 [math.NA]

David AB Hyde and Ronald Fedkiw. 2019. A unified approach to monolithic solid-fluid coupling of sub-grid and more resolved solids. *J. Comput. Phys.* 390 (2019), 490–526.

Chenfanfu Jiang, Craig Schroeder, Andrew Selle, Joseph Teran, and Alexey Stomakhin. 2015. The affine particle-in-cell method. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–10.

Navami Kairanda, Marc Habermann, Christian Theobalt, and Vladislav Golyanik. 2023. Neuralclothsim: Neural deformation fields meet the kirchhoff-love thin shell theory. *arXiv preprint arXiv:2308.12970* (2023).

Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. 2019. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, Vol. 38. Wiley Online Library, 59–70.

Theodore Kim, Nils Thürey, Doug James, and Markus Gross. 2008. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 1–6.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2023. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *Journal of Machine Learning Research* 24, 89 (2023), 1–97.

Wei Li, Yixin Chen, Mathieu Desbrun, Changxi Zheng, and Xiaopei Liu. 2020. Fast and scalable turbulent flow simulation with two-way coupling. *ACM Transactions on Graphics* 39, 4 (2020), Art–No.

Zilu Li, Guandao Yang, Xi Deng, Christopher De Sa, Bharath Hariharan, and Steve Marschner. 2023. Neural Caches for Monte Carlo Partial Differential Equation Solvers. In *SIGGRAPH Asia 2023 Conference Papers*. 1–10.

Songming Liu, Hao Zhongkai, Chengyang Ying, Hang Su, Jun Zhu, and Ze Cheng. 2022. A Unified Hard-Constraint Framework for Solving Geometrically Complex PDEs. In *Advances in Neural Information Processing Systems*, Vol. 35. 20287–20299.

Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids.. In *Symposium on Computer Animation*, Vol. 65. 74.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4460–4470.

Nicholas Metropolis and Stanisław Ulam. 1949. The Monte Carlo Method. *J. Amer. Statist. Assoc.* 44, 247 (1949), 335–341.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2023. Boundary Value Caching for Walk on Spheres. *ACM Trans. Graph.* 42, 4, Article 82 (2023).

Fadl Moukalled, Luca Mangani, Marwan Darwish, F Moukalled, L Mangani, and M Darwish. 2016. *The finite volume method*. Springer.

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 154–159.

Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM transactions on graphics (TOG)* 32, 3 (2013), 1–22.

Mohammad Sina Nabizadeh, Stephanie Wang, Ravi Ramamoorthi, and Albert Chern. 2022. Covector fluids. *ACM Transactions on Graphics (TOG)* 41, 4 (2022), 1–16.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 165–174.

Ziyin Qu, Xinxin Zhang, Ming Gao, Chenfanfu Jiang, and Baoquan Chen. 2019. Efficient and conservative fluids using bidirectional mapping. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–12.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* 378 (2019), 686–707.

Bo Ren, Chenfeng Li, Xiao Yan, Ming C Lin, Javier Bonet, and Shi-Min Hu. 2014. Multiple-fluid SPH simulation using a mixture model. *ACM Transactions on Graphics (TOG)* 33, 5 (2014), 1–11.

Damien Rioux-Lavoie, Ryusuke Sugimoto, Tümay Özdemir, Naoharu H. Shimada, Christopher Batty, Derek Nowrouzezahrai, and Toshiya Hachisuka. 2022. A Monte Carlo Method for Fluid Simulation. *ACM Trans. Graph.* 41, 6, Article 240 (2022).

Liangwang Ruan, Jinyuan Liu, Bo Zhu, Shinjiro Sueda, Bin Wang, and Baoquan Chen. 2021. Solid-fluid interaction with surface-tension-dominant contact. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–12.

Igor Santesteban, Miguel A. Otaduy, Nils Thuerey, and Dan Casas. 2022. ULNeF: Untangled Layered Neural Fields for Mix-and-Match Virtual Try-On. In *Advances in Neural Information Processing Systems, (NeurIPS)*.

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo geometry processing: a grid-free approach to PDE-based methods on volumetric domains. *ACM Trans. Graph.* 39, 4, Article 123 (2020).

Rohan Sawhney and Bailey Miller. 2023. *Zombie: A Grid-Free Monte Carlo Solver for PDEs*.

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2022. Grid-free Monte Carlo for PDEs with spatially varying coefficients. *ACM Trans. Graph.* 41, 4, Article 53 (2022).

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2023. Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions. *ACM Trans. Graph.* (2023).

Andrew Selle, Ronald Fedkiw, Byungmoon Kim, Yingjie Liu, and Jarek Rossignac. 2008. An unconditionally stable MacCormack method. *Journal of Scientific Computing* 35, 2 (2008), 350–371.

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 1–12.

Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. 2020. Implicit Neural Representations with Periodic Activation Functions. In *Proc. NeurIPS*.

Barbara Solenthaler and Renato Pajarola. 2009. Predictive-corrective incompressible SPH. In *ACM SIGGRAPH 2009 papers*. 1–6.

Jos Stam. 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. 121—-128.

N. Sukumar and Ankit Srivastava. 2022. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering* 389 (2022).

Geoffrey I. Taylor and Albert E. Green. 1936. Mechanism of the Production of Small Eddies from Large Ones. *Proceedings of the Royal Society of London* 158, 895 (1936), 499–521.

Sifan Wang, Shyam Sankaran, Hanwen Wang, and Paris Perdikaris. 2023. An Expert's Guide to Training Physics-informed Neural Networks. *arXiv preprint arXiv:2308.08468* (2023).

Sifan Wang, Hanwen Wang, and Paris Perdikaris. 2021. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances* 7, 40 (2021), eabi8605.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021. Geometry Processing with Neural Fields. *Advances in Neural Information Processing Systems* 34 (2021).

Cem Yuksel, Donald H House, and John Keyser. 2007. Wave particles. *ACM Transactions on Graphics (TOG)* 26, 3 (2007), 99–es.

Jonas Zehnder, Yue Li, Stelian Coros, and Bernhard Thomaszewski. 2021. Ntopo: Mesh-free topology optimization using implicit neural representations. *Advances in Neural Information Processing Systems* 34 (2021), 10368–10381.

Jonas Zehnder, Rahul Narain, and Bernhard Thomaszewski. 2018. An advection-reflection solver for detail-preserving fluid simulation. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–8.

Ryan S Zesch, Vismay Modi, Shinjiro Sueda, and David IW Levin. 2023. Neural Collision Fields for Triangle Primitives. In *SIGGRAPH Asia 2023 Conference Papers*. 1–10.

Fangcheng Zhong, Kyle Fogarty, Param Hanji, Tianhao Wu, Alejandro Sztrajman, Andrew Spielberg, Andrea Tagliasacchi, Petra Bosilj, and Cengiz Oztireli. 2023. Neural Fields with Hard Constraints of Arbitrary Differential Order. arXiv:2306.08943 [cs.LG]

Bo Zhu, Wenlong Lu, Matthew Cong, Byungmoon Kim, and Ronald Fedkiw. 2013. A new grid structure for domain extension. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.

Zeshun Zong, Xuan Li, Minchen Li, Maurizio M Chiaramonte, Wojciech Matusik, Eitan Grinspun, Kevin Carlberg, Chenfanfu Jiang, and Peter Yichen Chen. 2023. Neural Stress Fields for Reduced-order Elastoplasticity and Fracture. In *SIGGRAPH Asia 2023 Conference Papers*. 1–11.

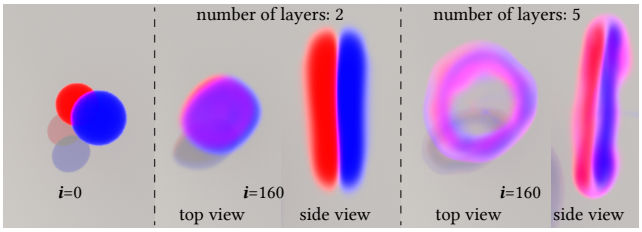


Figure 13: Performing the smoke ring experiment of Figure 8 with two different network depths. With a shallow network *(middle)*, the method cannot resolve vorticities sufficiently, resulting in no formation of a smoke ring and the two different smoke balls not mixing. With a deep enough network *(right)*, the smoke ring correctly forms and the two smoke balls mix. Layer sizes are 64 for both.

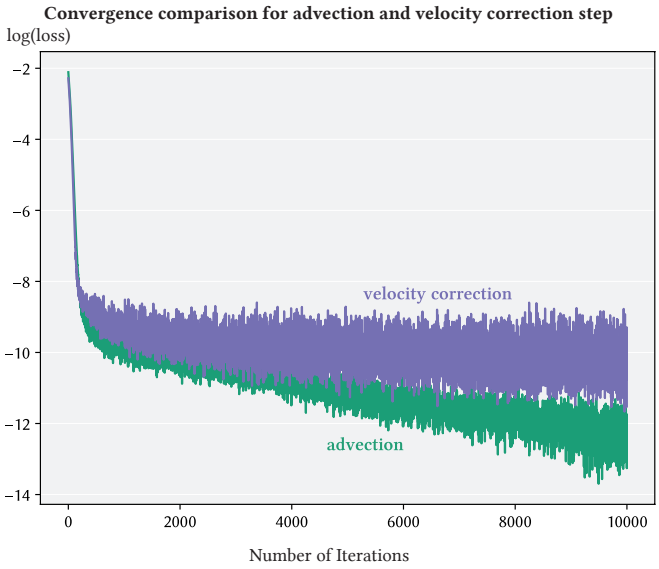


Figure 14: Plot showing how loss varies for 10,000 iterations for advection and velocity correction step.