IMPLICIT NEURAL SPATIAL REPRESENTATIONS FOR TIME-DEPENDENT PDEs

Anonymous authors
Paper under double-blind review

ABSTRACT

Numerically solving partial differential equations (PDEs) often entails spatial and temporal discretizations. Traditional methods (e.g., finite difference, finite element, smoothed-particle hydrodynamics) frequently adopt *explicit* spatial discretizations, such as grids, meshes, and point clouds, where each degree-of-freedom corresponds to a location in space. While these explicit spatial correspondences are intuitive to model and understand, these representations are not necessarily optimal for accuracy, memory-usage, or adaptivity. In this work, we explore implicit neural representation as an alternative spatial discretization, where spatial information is *implicitly* stored in the neural network weights. With implicit neural spatial representation, PDE-constrained time-stepping translates into updating neural network weights, which naturally integrates with commonly adopted optimization time integrators. We validate our approach on a variety of classic PDEs with examples involving large elastic deformations, turbulent fluids, and multi-scale phenomena. While slower to compute than traditional representations, our approach exhibits higher accuracy, lower memory consumption, and dynamically adaptive allocation of degrees of freedom without complex remeshing.

1 INTRODUCTION

Many science and engineering problems can be formulated as spatiotemporal partial differential equations (PDEs),

$$\boldsymbol{\mathcal{F}}(\boldsymbol{f}, \boldsymbol{\nabla}\boldsymbol{f}, \boldsymbol{\nabla}^2\boldsymbol{f}, \ldots, \dot{\boldsymbol{f}}, \ddot{\boldsymbol{f}}, \ldots) = \boldsymbol{0}, \quad \boldsymbol{f}(\boldsymbol{x}, t) : \Omega \times \mathcal{T} \to \mathbb{R}^d \,. \tag{1}$$

where $\Omega \in \mathbb{R}^m$ and $\mathcal{T} \in \mathbb{R}$ are the spatial and temporal domains, respectively. Examples include the Navier-Stokes equations for fluid dynamics and the elastodynamics equation for solid mechanics.

To numerically solve these PDEs, we oftentimes introduce temporal discretizations, $\{t_n\}_{n=0}^T$, where T is the number of temporal discretization samples and $\Delta t = t_{n+1} - t_n$ is the time step size. The solution to Equation (1) then becomes a list of spatially dependent vector fields: $\{\boldsymbol{f}^n(\boldsymbol{x})\}_{n=0}^T$.

Traditional approaches represent these spatially dependent vector fields using grids, meshes, or point clouds. For example, the grid-based linear finite element method (Hughes, 2012) defines a shape function N^i on each grid node and represents the spatially dependent vector field as $\boldsymbol{f}^n(\boldsymbol{x}) = \sum_{i=1}^P \boldsymbol{f}_i^n N^i$, where P is the number of spatial samples.

While widely adopted in scientific computing applications, these traditional spatial representations are not without drawbacks:

1. Spatial discretization errors abound in fluid simulations as artificial numerical diffusion (Lantz, 1971), dissipation (Fedkiw et al., 2001), and viscosity (Roache, 1998). These errors also appear in solid simulations as inaccurate collision resolution (Müller et al., 2015) and numerical fractures (Sadeghirad et al., 2011).
2. Memory usage spikes with the number of spatial samples P (Museth, 2013).
3. Adaptive meshing (Narain et al., 2012) and data structures (Setaluri et al., 2014) can reduce memory footprints but are often computationally expensive and challenging to implement.

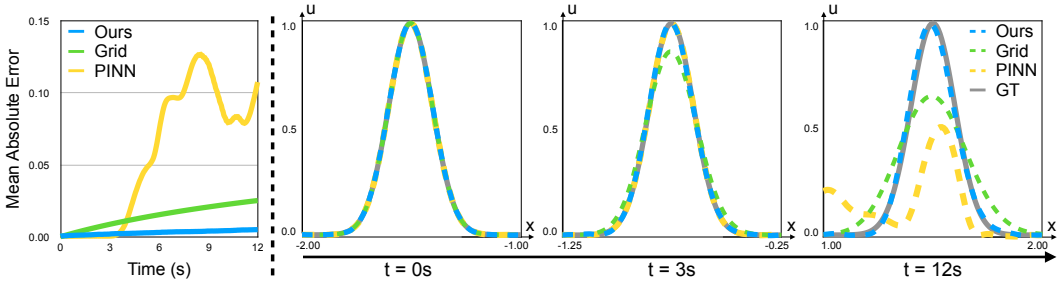

Figure 1: **1D advection example:** A Gaussian-shaped wave initially centered at $x = -1.5$ moves rightward with a constant velocity of 0.25. From left to right, we show mean absolute error plot over time and solutions at $t = 0s$, $t = 3s$ and $t = 12s$, respectively. The solution from grid-based finite difference method (green) tends to diffuse over time. PINN (yellow), trained within temporal range $0 \sim 3s$, fails to generalize for $t = 12s$. Our solution (blue) approximates the ground truth (grey) the best over time. All three representations have the same memory footprint: our approach and PINN (Raissi et al., 2019) both use $\alpha = 2$ hidden layers of width $\beta = 20$, and the finite difference grid resolution is 901.

We alleviate these limitations by exploring implicit neural representation (Park et al., 2019; Chen & Zhang, 2019; Mescheder et al., 2019) as an alternative *spatial* representation for PDE solvers. Unlike traditional representations that *explicitly* discretize the spatial vector via spatial primitives (e.g., points), neural spatial representations *implicitly* encode the field through neural network weights. In other words, the field is parameterized by a neural network (typically multilayer perceptrons), i.e., $\boldsymbol{f}^n(\boldsymbol{x}) = \boldsymbol{f}_{\theta^n}(\boldsymbol{x})$ with θ^n being the network weights. As such, the memory usage for storing the spatial field is independent of the number of spatial samples, but rather it is determined by the number of neural network weights. We show that under the same memory constraint, implicit neural representations indeed achieve higher accuracies than traditional discrete representations. Furthermore, implicit neural representations are adaptive by construction (Xie et al., 2021), allocating the network weights to resolve field details at *any* spatial location without changing the network architecture. Viewed from the lens of optimization-based time integrators, our PDE solver seeks neural network weights that optimize an incremental potential over time (Kane et al., 2000b; Bouaziz et al., 2014).

We compare the proposed solver to grid, mesh, and point cloud representations on time-dependent PDEs from various disciplines, and find that our approach trades wall-clock runtime in favor of three benefits: lower discretization error, lower memory usage, and built-in adaptivity.

2 Related Works

Many prior works have explored representing continuous vector fields with neural networks. Here we highlight two lines of work: implicit neural representation and physics informed neural network.

Implicit Neural Representation uses neural networks to parameterize spatially-dependent functions. It has successfully captured the radiance fields (Mildenhall et al., 2020) and the signed distance fields (Park et al., 2019) in computer vision and graphics settings. Chen et al. (2021), Pan et al. (2022), and Chen et al. (2022) also explore neural networks as spatial representations for dimension reduction. Memory consumptions of traditional representations, such as grids and point clouds, scale poorly with spatial resolutions. Adaptive discretizations can reduce memory but their generations are expensive. By contrast, neural representations are adaptive by construction and can use their representation capacities at arbitrary locations of interest without memory increases or data structures alterations. We refer to the recent review by Xie et al. (2021) for additional contexts.

Physics Informed Neural Network (PINN) is a powerful tool for solving PDEs. Traditional numerical methods, such as finite difference and finite element methods, represent tensor fields with well-studied polynomial basis functions (Hughes, 2012) constructed on meshes. Instead, PINN represents tensor fields with neural networks and converts the PDE solution process into finding network weights via PDE-based loss functions. Since the pioneering works by Raissi et al. (2019); Sirignano & Spiliopoulos (2018); Lagaris et al. (1998); Dissanayake & Phan-Thien (1994), PINN has been shown to excellently model forward simulation (Shin et al., 2020; Hennigh et al., 2021; Lu

et al., 2021), inverse design (Raissi et al., 2020; Mao et al., 2020; Mishra & Molinaro, 2022), and uncertain quantification (Lye et al., 2020). PINN has found success in a wide range of application domains, including turbulence (Hennigh et al., 2021), elasticity (Rao et al., 2020), acoustics (Sitzmann et al., 2020), and topology optimization (Zehnder et al., 2021). Due to its mesh-free nature, PINN can robustly handle high-dimensional PDEs. The recent review by Karniadakis et al. (2021) offers more details.

When it comes to using neural networks to represent vector fields, we face an important design choice as which dimension of the vector field is represented through the network. In the case of spatiotemporal vector fields, we have the choice of representing both the spatial and temporal dimensions via neural networks; we can also represent just the spatial dimension or just the temporal dimension with a network.

We observe that the spatial variable $\boldsymbol{x}$ is oftentimes bounded, e.g., a fixed geometry with well-defined boundaries. However, the temporal variable t can be unbounded, e.g., in a virtual reality application where the user interacts with a physical environment indefinitely (Sun et al., 2018). Modeling the additional temporal dimension also puts extra burden on the network. Motivated by these observations, we opt to treat the spatial and the temporal dimensions *differently*. In particular, we use the neural network strictly as a spatial representation and do not consider the temporal dimension as an input to the network. We then evolve this spatial representation by updating the network weights θ^n (See Figure 2), potentially for an indefinite amount of time. Such an approach is different from standard PINN that takes both the spatial dimension $\boldsymbol{x}$ and the temporal dimension t as an input to the network (Raissi et al., 2019; Karniadakis et al., 2021) which cannot resolve PDE solution outside a pre-defined temporal range (Kim et al., 2021) (See Figure 1). Neural spatial representation has been studied for strictly spatially dependent PDEs, such as elastostatics (Zehnder et al., 2021), elliptic PDEs (Chiaramonte et al., 2013), and geometry processing (Yang et al., 2021). To our best knowledge, computing neural spatial representations on time-dependent PDEs for long horizon tasks with multiple time steps has not been explored, and our work aims to fill this gap.

Optimization Time Integrators. Since our approach only replaces the spatial representations of traditional numerical solvers with neural networks while keeping the rest of the solver intact, it is compatible with *any* classical time integration schemes (e.g., implicit Euler). In particular, we formulate time integration as an energy minimization problem (Radovitzky & Ortiz, 1999; Kane et al., 2000b; Marsden & West, 2001; Kharevych et al., 2006). These integrators find wide applications in PDE solvers on traditional representations, such as grids (Batty et al., 2007), tetrahedral meshes (Bouaziz et al., 2014), and point clouds (Gast et al., 2015). In the case of neural spatial representations, time integration translates into optimizing the neural network weights at every time step.

3 Method: Time-stepping on Neural Spatial Representations

Our goal is to solve time-dependent PDEs on neural-network-based spatial representations. In Section 3.1, we first discuss representing spatial vector fields with neural networks. Afterward, we will describe our time-stepping technique that evolves from one neural spatial representation to another.

3.1 Neural Networks as Spatial Representations

We parameterize each of the time-discretized spatial vector fields with a neural network: $\boldsymbol{f}^n = \boldsymbol{f}_{\theta^n}$, where θ^n are the neural network weights at time t_n. Specifically, the field quantity at an arbitrary spatial location $\boldsymbol{x} \in \Omega$ can be queried via network inference $\boldsymbol{f}_{\theta^n}(\boldsymbol{x})$.

Traditional representations *explicitly* discretize the spatial vector field using primitives such as points, tetrahedra, or voxels. These primitives *explicitly* correspond to spatial locations due to their compactly supported basis functions (Hughes, 2012). By contrast, neural spatial representations *implicitly* encode the vector field via neural network weights. These weights do not directly correspond to specific spatial locations. Instead, each weight affects the vector field globally. Such global support is also an attribute of spectral methods (Canuto et al., 2007a;b). Compared to spectral methods, our approach does not need to know the required complexity ahead of time in order to determine the ideal basis functions (Xie et al., 2021). Our neural representation automatically optimizes its parameters to where field detail is present.

Whereas memory consumption of traditional *explicit* representations scales poorly with the number of spatial samples, memory consumption for *implicit* neural representations is independent of the number of spatial samples (Xie et al., 2021). Rather, memory use is determined by the number of neural network weights.

Network Architecture Following the implicit neural representation literature, we adopt a multilayer perceptron (MLP) architecture with SIREN activation function for its accuracy and quick convergence speed advantages (Sitzmann et al., 2020). Each MLP has a total of α hidden layers, each layer of width β. The specific choice of these hyper-parameters will be described in Section 4.

Spatial Gradients Traditional spatial representations (e.g., the finite element method) compute spatial gradients via basis functions. Higher-order gradients require higher-order basis functions. By contrast, a neural spatial representation is C^∞ by construction. We evaluate their gradients via computation-graph-based auto-differentiation with respect to the input (not the weights).

3.2 Temporal Evolution

Given previous-time spatial vector fields $\{\boldsymbol{f}^n(\boldsymbol{x})\}_{k=0}^n$, optimization-based time integrators compute the next time-step (t_{n+1}) vector field by optimizing

$$\boldsymbol{f}^{n+1} = \underset{\boldsymbol{f}^{n+1}}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} \boldsymbol{\mathcal{I}}(\Delta t, \{\boldsymbol{f}^k\}_{k=0}^{n+1}, \{\nabla \boldsymbol{f}^k\}_{k=0}^{n+1}, \{\nabla^2 \boldsymbol{f}^k\}_{k=0}^{n+1}, \ldots) \,. \tag{2}$$

Traditional time integrators, whether explicit and implicit, can be expressed in optimization forms (Kharevych et al., 2006). Furthermore, this optimization formulation applies to *any* spatial representation, and has been explored thoroughly for traditional discretizations (Batty et al., 2007; Bouaziz et al., 2014; Gast et al., 2015), which is defined over a finite number of the spatial integration samples $\mathcal{M} := \{\boldsymbol{x}^j \in \Omega \mid 1 \leq j \leq |\mathcal{M}|\}$, e.g., grids or meshes. Applying this formulation to a neural spatial representation, we optimize for

$$\theta^{n+1} = \underset{\theta^{n+1}}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} \boldsymbol{\mathcal{I}}(\Delta t, \{\boldsymbol{f}_{\theta^k}\}_{k=0}^{n+1}, \{\nabla \boldsymbol{f}_{\theta^k}\}_{k=0}^{n+1}, \{\nabla^2 \boldsymbol{f}_{\theta^k}\}_{k=0}^{n+1}, \ldots) \tag{3}$$

where $\{\theta^k\}_{k=0}^n$ are the (fixed, not variable) neural network weights from previous time steps. Figure 2 illustrates our time integration process. The particular choice of the objective function $\boldsymbol{\mathcal{I}}$ depends on the PDE of interest. In all the examples presented in this work, we solve this time-integration optimization problem via Adam (Kingma & Ba, 2014), a first-order stochastic gradient descent method.

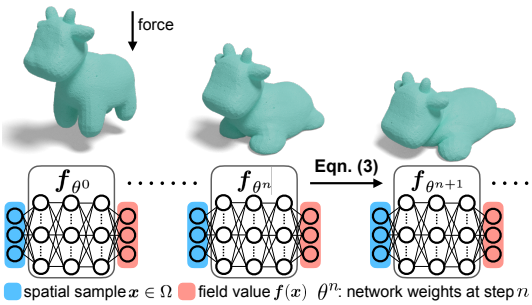

Figure 2: **Time integration.** We represent the field of interest using a neural network $\boldsymbol{f}_{\theta^n}$, whose weights θ^n are updated at each timestep via an optimization problem (Equation (3)). In this case, the spatial domain Ω is the interior of the initial object and the represented field $\boldsymbol{f}$ is the deformation map. The governing PDE is the elastodynamics equation (see Section 4.3).

Spatial Sampling *Explicit* spatial representations (e.g., tetrahedra mesh) are often tied to a particular spatial sampling; remeshing is sometimes possible, but can also have drawbacks, especially in higher dimensions (Alliez et al., 2002; Narain et al., 2012). By contrast, *implicit* spatial representations allow for arbitrary spatial sampling by construction (Equation (3)). Following Sitzmann et al. (2020), we dynamically sample $\mathcal{M}$ during optimization. For every gradient descent iteration in every time step, we use a stochastic sample set $\mathcal{M}$ from the spatial domain Ω; $\mathcal{M}$ corresponds to the "mini-batch" in stochastic gradient descent, with batch size $|\mathcal{M}|$. By directly drawing samples from the entire spatial domain Ω, our approach is reminiscent of mesh-free Monte Carlo methods (Sawhney & Crane, 2020).

Boundary Condition PDEs are typically accompanied by spatial (e.g., Dirichlet or Neumann) boundary conditions, which we formulate as additional penalty terms in the objective Equation (3),

$$\begin{aligned}\theta^{n+1} = \underset{\theta^{n+1}}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} \boldsymbol{\mathcal{I}}(\Delta t, \{\boldsymbol{f}_{\theta^k}\}_{k=0}^{n+1}, \{\nabla \boldsymbol{f}_{\theta^k}\}_{k=0}^{n+1}, \{\nabla^2 \boldsymbol{f}_{\theta^k}\}_{k=0}^{n+1}, \ldots) \\ + \lambda \sum_{\boldsymbol{x}^b\in\mathcal{M}^b\subset\partial\Omega} \boldsymbol{\mathcal{C}}(\boldsymbol{f}_{\theta^{n+1}}, \nabla \boldsymbol{f}_{\theta^{n+1}}, \nabla^2 \boldsymbol{f}_{\theta^{n+1}}, \ldots)\ ,\end{aligned} \tag{4}$$

where λ is the weighting factor and $\partial\Omega$ is the boundary of the spatial domain. The particular choice of the boundary constraint function $\boldsymbol{\mathcal{C}}$ depends on the problem of interest.

Initial Condition The neural network is initialized using the given initial condition, i.e., the field value at time $t = 0$, by optimizing

$$\theta^0 = \underset{\theta^0}{\operatorname{argmin}} \sum_{\boldsymbol{x}\in\mathcal{M}\subset\Omega} ||\boldsymbol{f}_{\theta^0}(\boldsymbol{x}) - \hat{\boldsymbol{f}}^0(\boldsymbol{x})||_2^2\ , \tag{5}$$

where $\hat{\boldsymbol{f}}^0$ is the given initial condition. Similar to Equation (3), we solve this optimization problem using Adam (Kingma & Ba, 2014) and stochastically sample $\mathcal{M}$ at each gradient descent iteration.

4 Experiments

In this section, we evaluate our method on three classic time-dependent PDEs: the advection equation, the Navier-Stokes equation and, the elastodynamics equation. For each problem, we first discuss the continuous PDE and the specific objective function $\boldsymbol{\mathcal{I}}$ for temporal evolution (recall Equation (3)). Then we demonstrate the advantages of our approach by comparing with baselines using discrete spatial representations (i.e. a grid, tetrahedral mesh, or point cloud). We refer readers to Appendices A and B for other implementation details (e.g., initial and boundary conditions) and additional results. The temporal evolutions of the PDEs are best illustrated by the **supplementary video**.

4.1 Advection Equation

Consider the classic 1D advection equation,

$$\frac{\partial u}{\partial t} + (a \cdot \nabla) u = 0\ , \tag{6}$$

where a is the advection velocity, and the vector field of interest is the advected quantity $\boldsymbol{f} = u$. It is well known that traditional spatial representations, such as grid-based finite differences, exhibit numerical dissipation for the advection equation (Courant et al., 1952; Selle et al., 2008).

Time Integration We adopt the same time integration scheme in both the traditional representation and ours. Choosing the energy-preserving midpoint method (Mullen et al., 2009) yields the time integration operator

$$\boldsymbol{\mathcal{I}} = \|\frac{u^{n+1}(\boldsymbol{x}) - u^n(\boldsymbol{x})}{\Delta t} + (a \cdot \nabla)(\frac{u^{n+1}(\boldsymbol{x}) + u^n(\boldsymbol{x})}{2})\|_2^2\ . \tag{7}$$

Results Figure 1 compares our results with those of grid-based finite differences and PINN (Raissi et al., 2019), subject to equal memory usage of the three methods. A Gaussian-shaped wave moves with constant velocity $a = 0.25$. Our approach and PINN both use $\alpha = 2$ hidden layers of width $\beta = 20$, and the finite difference grid resolution is 901. For ours and grid-based methods, we set $\Delta t = 0.05$. PINN does not require Δt but needs a *pre-specified* temporal range for training. For this temporal range, we use $[0, 3]$. As shown in Figure 1, the solution from the grid-based method diffuses over time due to its spatial discretization. While PINN can accurately capture the result up to $t = 3s$, it fails to produce meaningful solutions beyond its trained temporal range (see $t = 12s$) (Kim et al., 2021). By contrast, our solution does not suffer from numerical dissipation and agrees well with the ground truth at all frames (see the error plot in Figure 1).

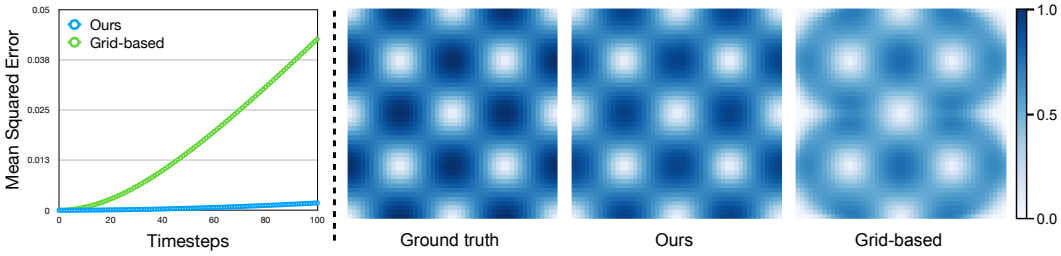

Figure 3: **2D Taylor-Green vortex simulation.** Left: mean squared error of the velocity field for 100 timesteps. Right: velocity magnitude of solutions from the ground truth, ours, and the grid-based method at timestep $n = 100$. Under the same memory usage (for storing the spatial representation), our solution has a significantly smaller error than the grid-based method.

4.2 Navier-Stokes Equations

In the incompressible and inviscid Navier-Stokes Equations

$$\rho_f(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \rho_f \boldsymbol{g}, \qquad \nabla \cdot \boldsymbol{u} = 0, \tag{8}$$

the vector field of interest is the fluid velocity field $\boldsymbol{f} = \boldsymbol{u}$; p is the pressure, $\boldsymbol{g}$ is the external force, and ρ_f is the fluid density. In our experiments, we consider $\rho_f = 1$ and $\boldsymbol{g} = 0$. The pressure field p is represented with another MLP network.

Time Integration We apply the Chorin-style operator splitting scheme (Chorin, 1968; Stam, 1999) to both the neural spatial and finite-difference grid representations. The scheme involves three sequential steps: advection (adv), pressure projection (pro), and velocity correction (cor).

Advection uses a semi-Lagrangian method, encoded by the operator (Staniforth & Côté, 1991)

$$\boldsymbol{\mathcal{I}}_{adv} = \|{\boldsymbol{u}_{adv}}^{n+1}(\boldsymbol{x}) - \boldsymbol{u}^n(\boldsymbol{x}_{backtrack})\|_2^2 \,, \tag{9}$$

whose optimization yields the advected velocity ${\boldsymbol{u}_{adv}}^{n+1}$. The backtracked location is given by $\boldsymbol{x}_{backtrack} = \boldsymbol{x} - \Delta t \boldsymbol{u}^n(\boldsymbol{x})$. While traditional spatial representations compute the backtracked velocity using interpolation (e.g., linear basis function), our approach *requires no interpolation*, only direct evaluation via network inference at the location $\boldsymbol{x}_{backtrack}$.

Pressure projection is encapsulated by the operator

$$\boldsymbol{\mathcal{I}}_{pro} = \|\nabla^2 p^{n+1}(\boldsymbol{x}) - \nabla \cdot {\boldsymbol{u}_{adv}}^{n+1}(\boldsymbol{x})\|_2^2. \tag{10}$$

Plugging $\boldsymbol{\mathcal{I}}_{pro}$ into the optimization solver, we obtain the pressure p^{n+1} that enforces incompressibility. Note that the MLP that represents the velocity field $\boldsymbol{u}_{adv}$ is kept fixed in this step.

Velocity correction is formulated by the operator

$$\boldsymbol{\mathcal{I}}_{cor} = \|\boldsymbol{u}^{n+1} - ({\boldsymbol{u}_{adv}}^{n+1}(\boldsymbol{x}) - \nabla p^{n+1}(\boldsymbol{x}))\|_2^2 \,, \tag{11}$$

which adds the pressure gradient to the advected velocity yielding the *incompressible* velocity $\boldsymbol{u}^{n+1}$.

Results We first test our method on the 2D Taylor-Green vortex with zero viscosity (Taylor & Green, 1937; Brachet et al., 1983). The closed-form analytical solution is given by: $\boldsymbol{u}(\boldsymbol{x}, t) = (\sin x \cos y, -\cos x \sin y)$ for $\boldsymbol{x} \in [0, 2\pi] \times [0, 2\pi]$. To compare under the same memory usage (for storing the velocity field), we use $\alpha = 3$ hidden layers of width $\beta = 32$ for our MLP and set grid resolution to 48 for the grid-based projection method. We set $\Delta t = 0.05$ and execute both methods for 100 timesteps. In Figure 3, we show the mean squared error of the solved velocity field over time. Compared to the grid-based projection method, our neural representation achieves less diffusion and higher accuracy.

For discrete grid representation, efficiently capturing multi-scale details usually requires difficult-to-implement adaptive data structures (Setaluri et al., 2014). Instead, implicit neural representations are adaptive by construction (Xie et al., 2021) and enable us to capture more details under the

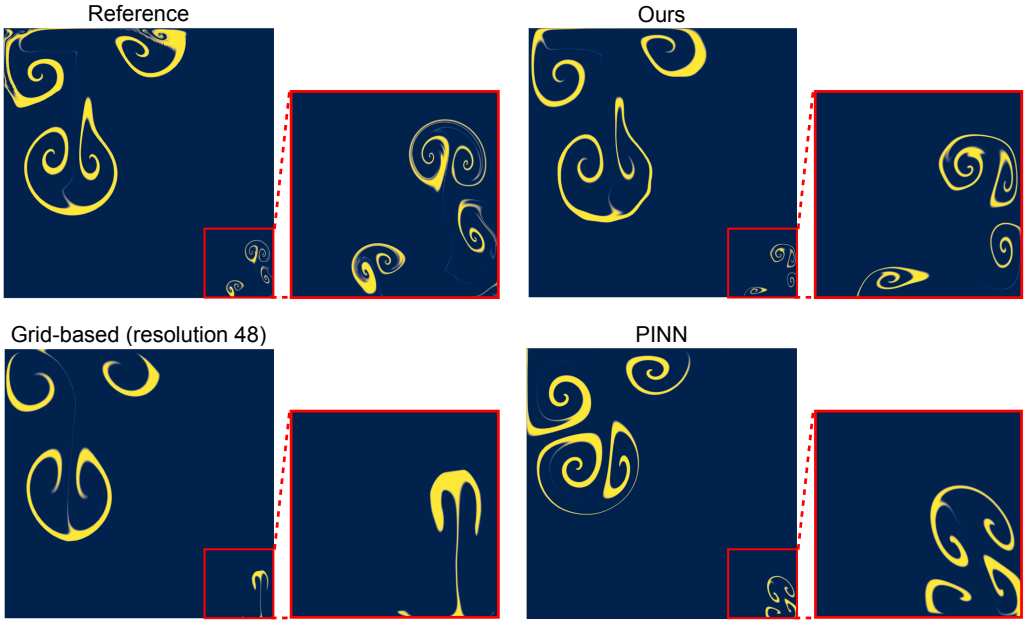

Figure 4: **Two vortices of different scales.** We show the advected density field after 2.5 seconds from the reference (top-left), our method (top-right), the grid-based method of resolution 48 (bottom-left) and PINN (bottom-right). The reference is obtained by running the high-resolution grid-based method (we use resolution 1024). Our MLP ($\alpha = 3, \beta = 32$) has the same memory footprint as grids of resolution 48. PINN uses the same MLP network as ours. Under the same memory constraint, our approach suffers from less dissipation, captures more vorticity, and best resembles the reference solution, whose grids takes $\sim 450\times$ memory compared to our network. See Figure 8 for the initial condition of this example.

same memory storage. We setup an example where the initial velocity field is composed by two Taylor-Green vortices of different scales (see Figure 8 for illustration). We compare our approach with PINN and the grid-based projection method under the same memory constraint for storing the spatial representations. Specifically, our approach and PINN uses a MLP with $\alpha = 3$ hidden layers of width $\beta = 32$ and the grid-based projection method uses resolution 48. We execute our approach and the grid-based method for 50 timesteps with $\Delta t = 0.05$, and train PINN with the same temporal range of 2.5 seconds. Using the solved velocity field, we advect a density field to visualize the amount of fine details captured by different representations. As shown in Figure 4, we are able to capture the fine details of the smaller vortex and best approximate the reference solution. The grid-based method (resolution 48) suffers from severe dissipation and fails to capture the vorticity. PINN is unable to correctly capture this two-vortices field and we found its training loss remains high ($\sim 1e^{-3}$) after convergence. This is in agreement with previous findings (Chuang & Barba, 2022) that suggest PINN approaches have difficulty solving Navier-Stokes equations for non-trivial examples.

4.3 Elastodynamics Equation

In the third experiment, we study the Elastodynamics equations

$$\rho_0 \ddot{\boldsymbol{\phi}} = \nabla \cdot \boldsymbol{P}(\boldsymbol{F}) + \rho_0 \boldsymbol{b} \tag{12}$$

that describe the motions of deformable solids (Gonzalez & Stuart, 2008). The vector field of interest is the deformation map $\boldsymbol{f} = \boldsymbol{\phi}$. Here ρ_0 is the density in the reference space, $\boldsymbol{P}$ is the first Piola-Kirchhoff stress, $\boldsymbol{F} = \nabla \boldsymbol{\phi}$ is the deformation gradient, $\dot{\boldsymbol{\phi}}$ and $\ddot{\boldsymbol{\phi}}$ are the velocity and acceleration, and $\boldsymbol{b}$ is the body force.

We assume a hyper-elasticity constitutive law, i.e., $\boldsymbol{P} = \frac{\partial \Psi}{\partial \boldsymbol{F}}$, where Ψ is the energy density function. In particular, we assume a variant of the stable Neo-Hookean energy (Smith et al., 2018)

$$\Psi = \frac{\lambda}{2} \mathrm{tr}^2(\Sigma - \boldsymbol{I}) + \mu(\det(\boldsymbol{F}) - 1)^2, \tag{13}$$

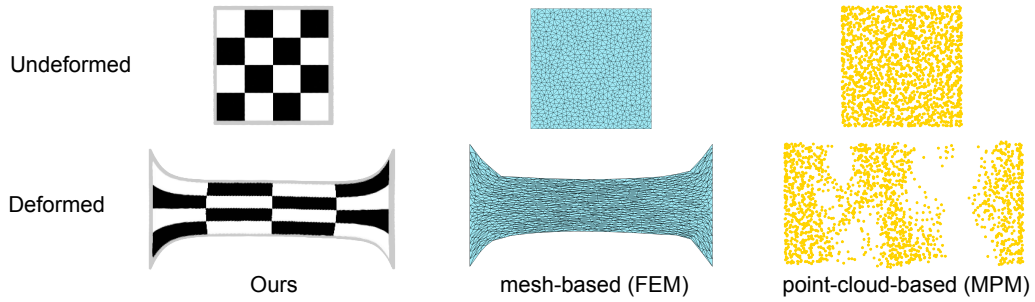

Figure 5: **Elastic tension test**. We compare the quasi-static simulation using our implicit neural representation (left) with the mesh representation (middle), and the point cloud representation (right). Our method handles large deformations matching the mesh-based finite element method (FEM), while the point cloud based material point method (MPM) suffers from incorrect numerical fracture.

where λ and μ are the first and second lame parameters, Σ are the singular values of the deformation gradient $\boldsymbol{F}$, and $\det(\boldsymbol{F})$ is the determinant of the deformation gradient $\boldsymbol{F}$. When $\mu = 0$, the elastic energy recovers the As-Rigid-As-Possible energy (Sorkine & Alexa, 2007).

Time Integration We apply the implicit Euler time integration scheme (Gast et al., 2015; Kane et al., 2000a) to the (1) tetrahedral finite element, (2) material point method, and (3) our neural representation, using the operator

$$\boldsymbol{\mathcal{I}} = \underbrace{\frac{1}{2}\rho_0(\dot{\boldsymbol{\phi}}^{n+1} - \dot{\boldsymbol{\phi}}^{n})^T(\dot{\boldsymbol{\phi}}^{n+1} - \dot{\boldsymbol{\phi}}^{n})}_{\text{kinematic energy}} + \underbrace{\Psi(\boldsymbol{\phi}^{n+1})}_{\substack{\text{elastic}\\ \text{energy}}} \underbrace{- \rho_0 \boldsymbol{b}^T \boldsymbol{\phi}^{n+1}}_{\substack{\text{external force}\\ \text{potential}}}, \tag{14}$$

where $\dot{\boldsymbol{\phi}}^{n+1} = (\boldsymbol{\phi}^{n+1} - \boldsymbol{\phi}^{n})/\Delta t$, ρ_0 is the density, $\boldsymbol{b}$ is the external force. We can also incorporate boundary conditions, e.g., positional and contact constraints, by introducing additional energy terms (Bouaziz et al., 2014; Li et al., 2020) (see Appendix A.4).

Results We first compare our implicit neural representation to the traditional tetraheral mesh representation (Finite Element Method, FEM (Hughes, 2012; Reddy, 2019)) and the point cloud representation (Material Point Method, MPM (Sulsky et al., 1995; Jiang et al., 2016)). We use $\alpha = 3$ hidden layers of width $\beta = 68$ for our MLP, which takes the same memory as the FEM mesh (0.8K vertices, 1.5K faces) and MPM point cloud (1.7K points). As shown in Figure 5, our method is capable of handling the large elastic deformations and matches the result of the traditional mesh-based method (FEM), while the point-cloud-based method (MPM) suffers from incorrect numerical fracture due to its meshless nature. To avoid these fractures, meshless methods require sophisticated modifications of the underlying kernel and basis functions (Gray et al., 2001; Su et al., 2022).

By virtue of its *implicit* nature, our representation is able to represent more intricate details compared to the traditional *explicit* representations under the same memory usage. In Figure 6, we show that our implicit neural representation allows the deformed square to gracefully fit the boundary of the sphere during the non-trivial collision. In contrast, the traditional mesh-based representation struggles to produce smooth result due to its insufficient mesh resolution. To alleviate such artifacts, the traditional mesh-based representation either needs to increase resolutions thusly inducing higher memory cost or conducts complex remeshing (Narain et al., 2012). In Figure 7, our implicit neural representation allows for more complex dynamics and fine geometry details compared to the traditional tetrahedral mesh representation.

Note that we adopt the same collision detection and handling strategy for both the neural representation and the mesh-based representation (FEM). Specifically, we use a spring-like penalty force and the corresponding energy to move the collided point out of its collision surface, similar to (McAdams et al., 2011; Xian et al., 2019). Since our approach and FEM share the same time integration scheme and the same collision handling method, the difference reported in Figure 6 and Figure 7 strictly stems from the underlying spatial representations.

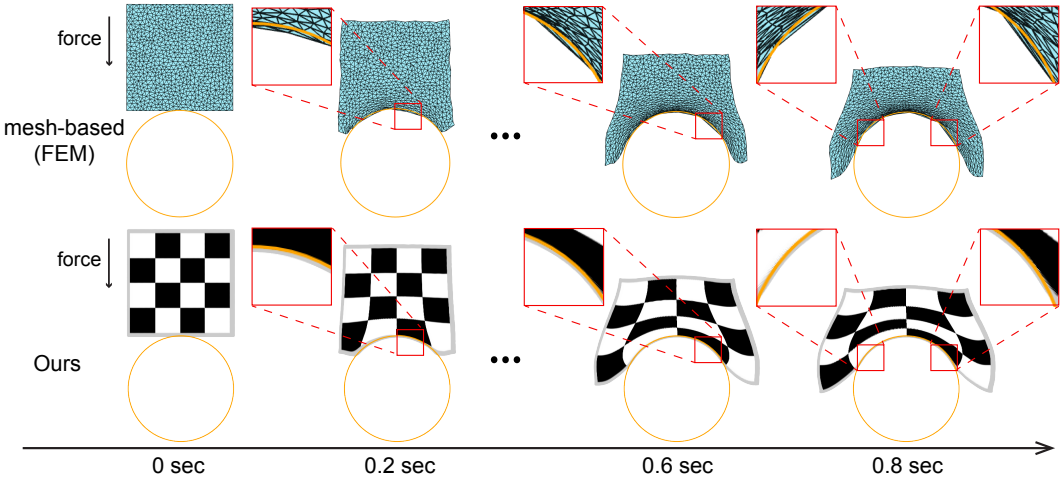

Figure 6: **When a elastic square collides with a circle,** the finite element mesh (top) conforms poorly at the interface compared to the neural representation (bottom), for equal memory usage.

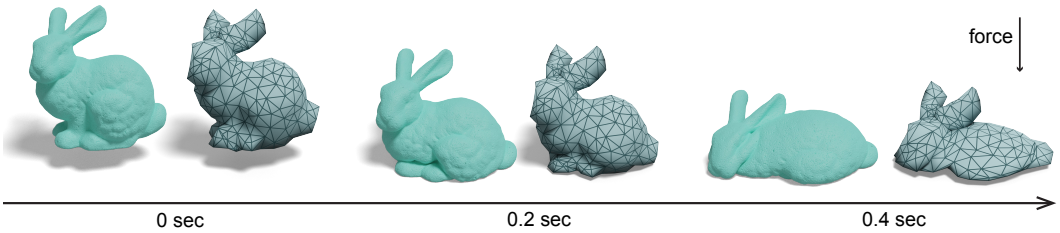

Figure 7: **A bunny collides with the ground in 3D.** Our neural representation (green, left) captures more intricate geometry details and complex dynamics compared to the traditional tetrahedral mesh representation (blue, right) under the same memory usage.

These advantages extend to other complex 3D simulations. Figure 2 and Figure 11 depict a cow and statue deforming as they collide with the ground, exhibiting complex geometry, and rich contact-induced deformations.

5 Discussion and Conclusion

In this work, we explore *implicit* neural representations as spatial representations for numerically modeling time-dependent PDEs. This representation naturally integrates with widely adopted optimization-based time integrators. PDE solvers with neural spatial representation offers improved accuracy, reduced memory, and automatic adaptivity compared to traditional *explicit* representations such as a mesh, grid, or point cloud.

While offering important benefits, neural-spatial-representation-based PDE time-stepping requires longer wall-clock computation time than existing methods (see also Table 1 by Zehnder et al. (2021) and Section 7 by Yang et al. (2021)). Optimizing neural networks weights takes longer than optimizing grid values even if there are fewer number of neural network weights than the number of grid nodes. For instance, for the bunny example (Figure 7), our neural network optimization takes around 30 minutes per timestep while the corresponding FEM simulation takes less than 1 minute. Future work therefore lies in exploring advanced training techniques that reduce training time (Liu et al., 2020; Martel et al., 2021; Takikawa et al., 2021). In particular, Müller et al. (2022) offer a promising direction where they show that we can reduce implicit neural representation training time from hours to seconds via advanced data structures and optimized implementation.

Our work demonstrates the effectiveness of neural spatial representations in solving time-dependent PDEs and observes empirical convergence under refinement (see Figure 10). Future work should consider theoretical analysis (Mishra & Molinaro, 2022) on convergence and stability. More challenging boundary conditions, such as turbulence (Wilcox et al., 1998) and intricate contacts (Johnson & Johnson, 1987), are also important future directions.

REFERENCES

Pierre Alliez, Mark Meyer, and Mathieu Desbrun. Interactive geometry remeshing. *ACM Transactions on Graphics (TOG)*, 21(3):347–354, 2002.

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

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM transactions on graphics (TOG)*, 33(4):1–11, 2014.

Marc E Brachet, Daniel I Meiron, Steven A Orszag, BG Nickel, Rudolf H Morf, and Uriel Frisch. Small-scale structure of the taylor–green vortex. *Journal of Fluid Mechanics*, 130:411–452, 1983.

Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, and Thomas A Zang. *Spectral methods: fundamentals in single domains*. Springer Science & Business Media, 2007a.

Claudio Canuto, M Yousuff Hussaini, Alfio Quarteroni, and Thomas A Zang. *Spectral methods: evolution to complex geometries and applications to fluid dynamics*. Springer Science & Business Media, 2007b.

Peter Yichen Chen, Maurizio Chiaramonte, Eitan Grinspun, and Kevin Carlberg. Model reduction for the material point method via an implicit neural representation of the deformation map. *arXiv preprint arXiv:2109.12390*, 2021.

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

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

M Chiaramonte, M Kiener, et al. Solving differential equations using neural networks. *Machine Learning Project*, 1, 2013.

Alexandre Joel Chorin. Numerical solution of the navier-stokes equations. *Mathematics of computation*, 22(104):745–762, 1968.

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

Richard Courant, Eugene Isaacson, and Mina Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on pure and applied mathematics*, 5(3): 243–255, 1952.

MWMG Dissanayake and Nhan Phan-Thien. Neural-network-based approximations for solving partial differential equations. *communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

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

Theodore F Gast, Craig Schroeder, Alexey Stomakhin, Chenfanfu Jiang, and Joseph M Teran. Optimization integrator for large time steps. *IEEE transactions on visualization and computer graphics*, 21(10):1103–1115, 2015.

Oscar Gonzalez and Andrew M Stuart. *A first course in continuum mechanics*, volume 42. Cambridge University Press, 2008.

James P Gray, Joseph J Monaghan, and RP1021 Swift. Sph elastic dynamics. *Computer methods in applied mechanics and engineering*, 190(49-50):6641–6662, 2001.

Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pp. 447–461. Springer, 2021.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):1–16, 2019.

Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and Andrew Selle. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, SIGGRAPH '16. Association for Computing Machinery, 2016.

Kenneth Langstreth Johnson and Kenneth Langstreth Johnson. *Contact mechanics*. Cambridge university press, 1987.

C. Kane, J. E. Marsden, M. Ortiz, and M. West. Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems. *International Journal for Numerical Methods in Engineering*, 49(10):1295–1325, 2000a.

Couro Kane, Jerrold E Marsden, Michael Ortiz, and Matthew West. Variational integrators and the newmark algorithm for conservative and dissipative mechanical systems. *International Journal for numerical methods in engineering*, 49(10):1295–1325, 2000b.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Liliya Kharevych, W Wei, Yiying Tong, Eva Kanso, Jerrold E Marsden, Peter Schröder, and Matthieu Desbrun. *Geometric, variational integrators for computer animation*. Eurographics Association, 2006.

Jungeun Kim, Kookjin Lee, Dongeun Lee, Sheo Yon Jhin, and Noseong Park. Dpm: a novel training method for physics-informed neural networks in extrapolation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 8146–8154, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

RB Lantz. Quantitative evaluation of numerical diffusion (truncation error). *Society of Petroleum Engineers Journal*, 11(03):315–320, 1971.

David I.W. Levin. Bartels: A lightweight collection of routines for physics simulation, 2020. https://github.com/dilevin/Bartels.

Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy R Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M Kaufman. Incremental potential contact: intersection- and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4):49, 2020.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Advances in Neural Information Processing Systems*, 33:15651–15663, 2020.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

Kjetil O Lye, Siddhartha Mishra, and Deep Ray. Deep learning observables in computational fluid dynamics. *Journal of Computational Physics*, 410:109339, 2020.

Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

Jerrold E Marsden and Matthew West. Discrete mechanics and variational integrators. *Acta Numerica*, 10:357–514, 2001.

Julien NP Martel, David B Lindell, Connor Z Lin, Eric R Chan, Marco Monteiro, and Gordon Wetzstein. Acorn: Adaptive coordinate networks for neural scene representation. *arXiv preprint arXiv:2105.02788*, 2021.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. Efficient elasticity for character skinning with contact and collisions. In *ACM SIGGRAPH 2011 papers*, pp. 1–12. 2011.

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

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for pdes. *IMA Journal of Numerical Analysis*, 42(2):981–1022, 2022.

Patrick Mullen, Keenan Crane, Dmitry Pavlov, Yiying Tong, and Mathieu Desbrun. Energy-preserving integrators for fluid animation. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

Matthias Müller, Nuttapong Chentanez, Tae-Yong Kim, and Miles Macklin. Air meshes for robust collision handling. *ACM Transactions on Graphics (TOG)*, 34(4):1–9, 2015.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

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

Rahul Narain, Armin Samii, and James F O'brien. Adaptive anisotropic remeshing for cloth simulation. *ACM transactions on graphics (TOG)*, 31(6):1–10, 2012.

Shaowu Pan, Steven L Brunton, and J Nathan Kutz. Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *arXiv preprint arXiv:2204.03216*, 2022.

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

Raul Radovitzky and Michael Ortiz. Error estimation and adaptive meshing in strongly nonlinear dynamic problems. *Computer Methods in Applied Mechanics and Engineering*, 172(1-4):203–240, 1999.

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

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

Chengping Rao, Hao Sun, and Yang Liu. Physics informed deep learning for computational elastodynamics without labeled data. *arXiv preprint arXiv:2006.08472*, 2020.

J. N. Reddy. *Introduction to the Finite Element Method*. McGraw-Hill Education, New York, 4th edition edition, 2019.

Patrick J Roache. *Fundamentals of computational fluid dynamics*. Hermosa Publishers, 1998.

Alireza Sadeghirad, Rebecca M Brannon, and Jeff Burghardt. A convected particle domain interpolation technique to extend applicability of the material point method for problems involving massive deformations. *International Journal for numerical methods in Engineering*, 86(12): 1435–1456, 2011.

Rohan Sawhney and Keenan Crane. Monte carlo geometry processing: A grid-free approach to pde-based methods on volumetric domains. *ACM Transactions on Graphics*, 39(4), 2020.

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

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

Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *arXiv preprint arXiv:2004.01806*, 2020.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

Breannan Smith, Fernando De Goes, and Theodore Kim. Stable neo-hookean flesh simulation. *ACM Trans. Graph.*, mar 2018.

Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, volume 4, pp. 109–116, 2007.

Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128, 1999.

Andrew Staniforth and Jean Côté. Semi-lagrangian integration schemes for atmospheric models—a review. *Monthly weather review*, 119(9):2206–2223, 1991.

Haozhe Su, Tao Xue, Chengguizi Han, and Mridul Aanjaneya. A-ulmpm: An adaptively updated lagrangian material point method for efficient physics simulation without numerical fracture. In *Computer Graphics Forum*, volume 41, pp. 325–341. Wiley Online Library, 2022.

Deborah Sulsky, Shi-Jian Zhou, and Howard L Schreyer. Application of a particle-in-cell method to solid mechanics. *Computer physics communications*, 87(1-2):236–252, 1995.

Qi Sun, Anjul Patney, Li-Yi Wei, Omer Shapira, Jingwan Lu, Paul Asente, Suwen Zhu, Morgan McGuire, David Luebke, and Arie Kaufman. Towards virtual reality infinite walking: dynamic saccadic redirection. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018.

Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11358–11367, 2021.

Geoffrey Ingram Taylor and Albert Edward Green. Mechanism of the production of small eddies from large ones. *Proceedings of the Royal Society of London. Series A-Mathematical and Physical Sciences*, 158(895):499–521, 1937.

David C Wilcox et al. *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA, 1998.

Zangyueyang Xian, Xin Tong, and Tiantian Liu. A scalable galerkin multigrid method for real-time simulation of deformable objects. *ACM Trans. Graph.*, 38(6), nov 2019.

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. Neural fields in visual computing and beyond. *arXiv preprint arXiv:2111.11426*, 2021.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. Geometry processing with neural fields. *Advances in Neural Information Processing Systems*, 34, 2021.

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

A Implementation Details

A.1 Optimization

We solve our time-integration optimization problem (Equation (3)) with the Adam optimizer (Kingma & Ba, 2014). For all examples in our experiments, we set an initial learning rate $\mathbf{lr}_0$ and reduce it by a factor of 0.1 if the loss value does not decrease for $\mathbf{iter}_\text{p}$ iterations. We stop the optimization process when the learning rate is smaller than $\mathbf{lr}_\text{min}$ or until it reaches a maximum of $\mathbf{iter}_\text{max}$ iterations. Specific values of these hyper-parameters are described for each example below. We implement our method using PyTorch library and performed our experiments on an NVIDIA GeForce RTX 3090 GPU.

A.2 Advection Equation

For our advection example in Figure 1, the 1D spatial domain is $\Omega = [-2, 2]$. We consider the Dirichlet boundary condition, i.e., the advected quantity at boundaries equals zero. Hence we set the boundary constraint term in Equation (4) as

$$\boldsymbol{\mathcal{C}} = ||u^{n+1}(\boldsymbol{x})||_2^2, \tag{15}$$

with the weighting factor $\lambda = 1$. The initial condition for this example is

$$\hat{u^0}(\boldsymbol{x}) = e-\frac{(\boldsymbol{x}-\mu)^2}{2\sigma^2}, \tag{16}$$

with $\mu = -1.5$ and $\sigma = 0.1$. We set the optimization hyper-parameters $\mathbf{lr}_0 = 1e{-}4$, $\mathbf{lr}_\text{min} = 1e{-}8$, $\mathbf{iter}_\text{p} = 500$ and $\mathbf{iter}_\text{max} = 20000$. For each gradient descent iteration, we randomly sample $|\mathcal{M}| = 5000$ points within the spatial domain $[-2, 2]$. For this example, our method takes $\sim 80s$ to compute per timestep, while the grid-based method (using the same memory) takes $\sim 4e{-}3s$.

A.3 Navier-Stokes Equations

For our 2D fluid examples, the spatial domain is $\Omega = [-1, 1] \times [-1, 1]$. We consider solid boundary conditions, i.e., the fluid cannot go through the boundaries. Recall that we adopt the operator splitting scheme. Therefore, the boundary constraint terms for the three sequential steps are

$$\begin{aligned} \boldsymbol{\mathcal{C}}_{adv} &= ||\boldsymbol{u}_{adv\perp}^{n+1}(\boldsymbol{x})||_2^2 \\ \boldsymbol{\mathcal{C}}_{pro} &= ||\nabla_\perp p^{n+1}(\boldsymbol{x})||_2^2 \\ \boldsymbol{\mathcal{C}}_{cor} &= ||\boldsymbol{u}_\perp^{n+1}(\boldsymbol{x})||_2^2 \end{aligned} \tag{17}$$

where $\perp$ indicates the perpendicular direction against the boundary. The weighting factor $\lambda = 1$.

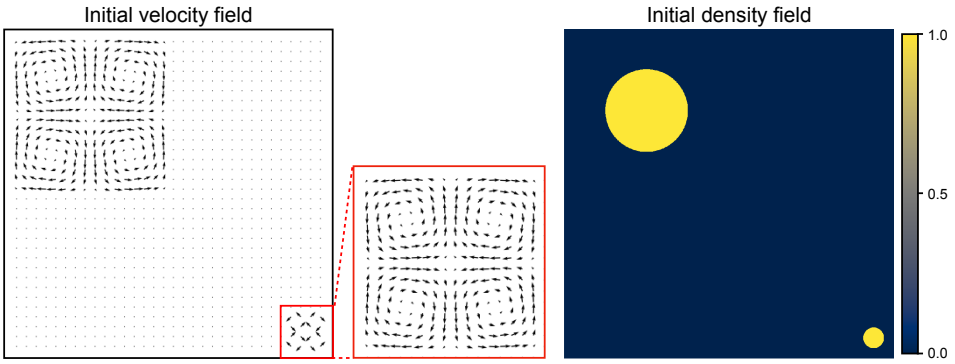

Figure 8: **Initial condition for the example in Figure 4.** Left: velocity field (Equation (19)). Right: density field (Equation (20)).

2D Taylor-Green vortex Standard 2D Taylor-Green is originally defined in domain $[0, 2\pi] \times [0, 2\pi]$. We translate and scale the domain to $[-1, 1] \times [-1, 1]$ such that the input range fits our MLP with the SIREN activation (Sitzmann et al., 2020). Therefore, the initial condition for the velocity field becomes

$$\hat{\boldsymbol{u}}^0(\boldsymbol{x}) = (\frac{1}{\pi}\sin[\pi(x+1)]\cos[\pi(y+1)], -\frac{1}{\pi}\cos[\pi(x+1)]\sin[\pi[y+1]]). \tag{18}$$

After the simulation, we convert it back to domain $[0, 2\pi] \times [0, 2\pi]$ for evaluation and comparison. We set the optimization hyper-parameters $\mathbf{lr}_0 = 1e{-}5$, $\mathbf{lr}_{\text{min}} = 1e{-}8$, $\mathbf{iter}_{\text{p}} = 500$ and $\mathbf{iter}_{\text{max}} = 20000$. The size of sample set $|\mathcal{M}| = 256^2$. For this example, our method takes $\sim$ 10min to compute per timestep, while the grid-based method (using the same memory) takes $\sim 0.03s$.

Two vortices of different scale For the example shown in Figure 4, the initial condition for the velocity field is

$$\hat{\boldsymbol{u}}^0(\boldsymbol{x}) = \begin{cases} (\sin[2\pi(x+1)]\cos[2\pi(y+1)], -\cos[2\pi(x+1)]\sin[2\pi(y+1)]) & \boldsymbol{x} \in [-1, 0]^2 \\ (\sin[8\pi(x-\frac{7}{4})]\cos[8\pi(y-\frac{7}{4})], -\cos[8\pi(x-\frac{7}{4})]\sin[8\pi(y-\frac{7}{4})]) & \boldsymbol{x} \in [\frac{7}{4}, 1]^2 \\ (0, 0) & \text{otherwise.} \end{cases} \tag{19}$$

The density field that we advect is initialized as

$$\hat{d}^0(\boldsymbol{x}) = \begin{cases} 1 & ||2\boldsymbol{x}+1|| \leq 0.5 \text{ or } ||8\boldsymbol{x}+7|| \leq 0.5 \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

Figure 8 visually illustrates the above initial conditions. After the simulation, we convert it back to domain $[0, 2\pi] \times [0, 2\pi]$ for evaluation and comparison. We set the optimization hyper-parameters $\mathbf{lr}_0 = 1e{-}5$, $\mathbf{lr}_{\text{min}} = 1e{-}8$, $\mathbf{iter}_{\text{p}} = 500$ and $\mathbf{iter}_{\text{max}} = 20000$. The size of sample set $|\mathcal{M}| = 128^2$. For this example, our method takes $\sim$ 10min to compute per timestep.

A.4 Elastodynamics Equation

Initial and Boundary Conditions For our 2D elasticity examples in Figure 5, Figure 9 and Figure 6, the 2D spatial domain is $\Omega = [-1, 1] \times [-1, 1]$. For our 3D elasticity examples in Figure 10, the 3D spatial domain is $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$. For our 2D and 3D examples involving nonregular geometry (Figure 7, Figure 2 and Figure 11), the spatial domain is the interior of the shape, including the boundary. The initial condition for all the elasticity examples is

$$\begin{aligned} \hat{\boldsymbol{\phi}}^0(\boldsymbol{x}) &= (0, 0) \text{ (2D)}, \\ \hat{\boldsymbol{\phi}}^0(\boldsymbol{x}) &= (0, 0, 0) \text{ (3D)} \end{aligned} \tag{21}$$

The boundary constraint for elasticity examples involves positional constraints or collision constraints. Positional constraints, or *Dirichlet boundary conditions*, can be realized by defining the position of the constraint set $\partial\Omega$ as the desired goal positions $\overline{\boldsymbol{\phi}}_{\partial\Omega}$:

$$\boldsymbol{\mathcal{I}}_{\text{pos}} = \|\boldsymbol{\phi}^{n+1}_{\partial\Omega} - \overline{\boldsymbol{\phi}}_{\partial\Omega}\|_2^2. \tag{22}$$

Table 1: **Experiment setup for the elasticity examples.** ρ_0 is the density. λ and μ are the first and second lame parameters. α and β are the number of hidden layers and the dimension of the hidden features. $\mathbf{lr}_0$ and $\mathbf{iter}_{\text{max}}$ are the initial learning rate and the maximum number of iterations. $\mathbf{t}_{\text{avg}}$ is average training time per time step. Note that the density ρ_0 and timestep size **dt** are reported as N/A for the quasistatic example Stretch (2D) (Figure 5).

Example	**Dim**	$\lvert\mathcal{M}\rvert$	**dt**	ρ_0	λ	μ	α	β	$\mathbf{lr}_0$	$\mathbf{iter}_{\text{max}}$	$\mathbf{t}_{\text{avg}}$(s)
Collision (2D) (Figure 6)	2	100^2	0.1	$1e1$	$2e1$	$1e3$	3	68	$1e{-}5$	$1e4$	$1.38e2$
Stretch (2D) (Figure 5)	2	100^2	N/A	N/A	$1e0$	$1e3$	3	68	$1e{-}4$	$5e4$	$2.30e3$
Bunny (Figure 7)	3	20^3	0.1	$1e0$	$1e2$	$1e3$	3	66	$1e{-}5$	$2e4$	$1.70e3$
Spot (Figure 2)	3	20^3	0.1	$1e0$	$1e2$	$1e3$	3	66	$1e{-}3$	$5e3$	$1.74e3$
Lucy (Figure 11)	3	20^3	0.1	$1e0$	$1e3$	$1e3$	3	128	$1e{-}4$	$2e4$	$1.16e3$

Collision constraints can be handled by adding unilateral constraints dynamically and viewing the collision penalty force as external force. Specifically, for a colliding point $\boldsymbol{q}_c$, we first find the closest surface point $\boldsymbol{b}_c$ with normal $\boldsymbol{n}_c$, and define our spring-like collision penalty force as:

$$\boldsymbol{f}_{\text{col}} = k_{\text{col}}((\boldsymbol{b}_c - \boldsymbol{q}_c)^\top \boldsymbol{n}_c)\boldsymbol{n}_c. \tag{23}$$

where k_{col} is the ratio for the collision penalty force.

The corresponding collision energy can be defined as the work exerted by the collision force:

$$\boldsymbol{\mathcal{I}}_{\text{col}} = \rho_0 \boldsymbol{f}_{\text{col}}^T \boldsymbol{\phi}^{n+1}. \tag{24}$$

Experiment Setup For all the 2D comparison under the same memory usage, we use $\alpha = 3$ hidden layers of width $\beta = 68$ with SIREN activation function (Sitzmann et al., 2020) for our MLP, which takes the same memory (57 KB) as the FEM mesh (0.8K vertices, 1.5K faces) and MPM point cloud (1.7K points) in use. We initialize the 2D deformation field of the network to be zero using $|\mathcal{M}| = 1000^2$ uniform and random samples. Then we train the network using $|\mathcal{M}| = 100^2$ uniform and random samples at each training iteration. We use Bartels (Levin, 2020) and Taichi (Hu et al., 2019) to perform the FEM and MPM simulation, respectively. We run our FEM and MPM comparison on CPU using a MacBook Pro with Apple M2 processor and 24GB of RAM.

For the 3D comparison under the same memory usage, for the bunny example (Figure 7), we use $\alpha = 3$ hidden layers of width $\beta = 66$ with SIREN activation function for our MLP, which takes the same memory (53 KB) as the FEM mesh (0.5K vertices, 1.5K tetrahedra) in use. For the statue example (Figure 11), we use $\alpha = 3$ hidden layers of width $\beta = 128$ with SIREN activation function for our MLP, which takes the same memory (197 KB) as the FEM mesh (2.0K vertices, 7.0K tetrahedra) in use. We initialize the 3D deformation field of the network to be zero using $|\mathcal{M}| = 100^3$ uniform and random samples. Then we train the network using $|\mathcal{M}| = 20^3$ uniform and random samples at each training iteration. Here for simplicity we use the mesh vertices as the uniform samples. We further report all the parameters and experiment setup in Table 1. In addition, we set the hyper-parameters $\mathbf{iter}_{\text{p}} = 800$ and $\mathbf{lr}_{\text{min}} = 1e{-}8$ for all elasticity examples.

For sampling of the shapes involving nonregular geometry, for simplicity we choose to use a triangle or tetrahedral mesh and perform sampling within it. An ideal alternative would be adopting the implicit representation of the surface and performing rejection sampling based on it.

For rendering, we simply sample sufficient number of points from the undeformed shape and evaluate the trained model at time t on the sample positions to predict their deformation. Here we only sample the surface of the shape in 3D cases. Then we render the shape as a dense point cloud.

B Additional Results

B.1 Elastodynamics Equation

We validate the physical plausibility of our method using a small 2D patch test Figure 9. We show that our implicit neural representation can exhibit volume-preserving property under both stretching and compression.

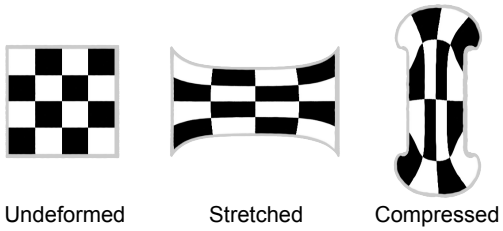

Figure 9: **Elasticity patch test.** Quasistatic simulation in 2D (undeformed, stretched, compressed).

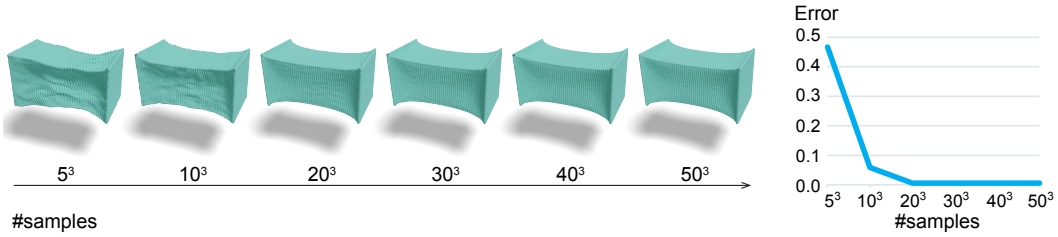

Figure 10: **Sampling convergence test**. Quasi-static elasticity simulation using different number of samples. We trained the implicit neural representation using different number of samples and visualized the result using samples $|\mathcal{M}| = 50^3$ (left). We further compute the error with respect to $|\mathcal{M}| = 50^3$ when using different number of samples (right). As we increase the number of the training samples in use, the deformation field converges to the result trained on the highest resolution.

We demonstrate *qualitative* and *quatitative* convergence of our method when increasing the number of training samples in use. In Figure 10, we compare the quasi-static stretching results visualized using $|\mathcal{M}| = 50^3$ uniform samples when using different number of training samples ($|\mathcal{M}| = 5^3, 10^3, 20^3, 30^3, 40^3, 50^3$), and report the error with respect to the high-resolution trained result (trained and visualized both on $|\mathcal{M}| = 50^3$).

Finally, we provide an additional example for elasticity involving complex contact-induced deformations in Figure 11. Our implicit neural representation is able to maintain more intricate geometry details compared to the traditional tetrahedral mesh representation under the same memory usage.

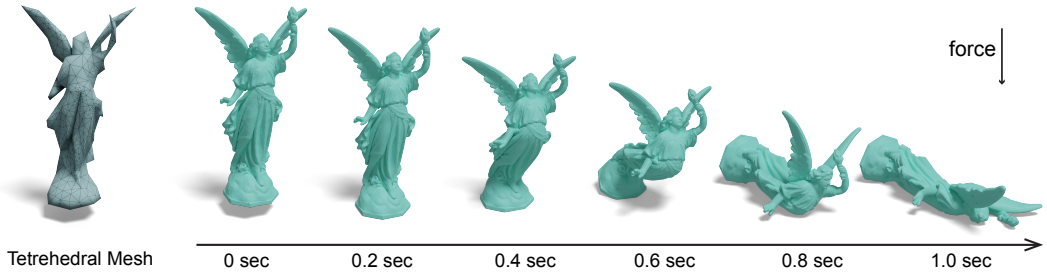

Figure 11: **The statue collides with the ground and deforms.** Our implicit neural representation (green, right) is capable of capturing more fine geometry details compared to the traditional tetrahedral mesh representation (blue, left) under the same memory footprint.