Neural Caches for Monte Carlo Partial Differential Equation Solver

Zilu Li*
Cornell University
zl327@cornell.edu

Guandao Yang*
Cornell University
gy46@cornell.edu

Xi Deng
Cornell University
xd93@cornell.edu

Christopher De Sa
Cornell University
cdesa@cs.cornell.edu

Bharath Hariharan
Cornell University
bharathh@cs.cornell.edu

Steve Marschner
Cornell University
srm@cs.cornell.edu

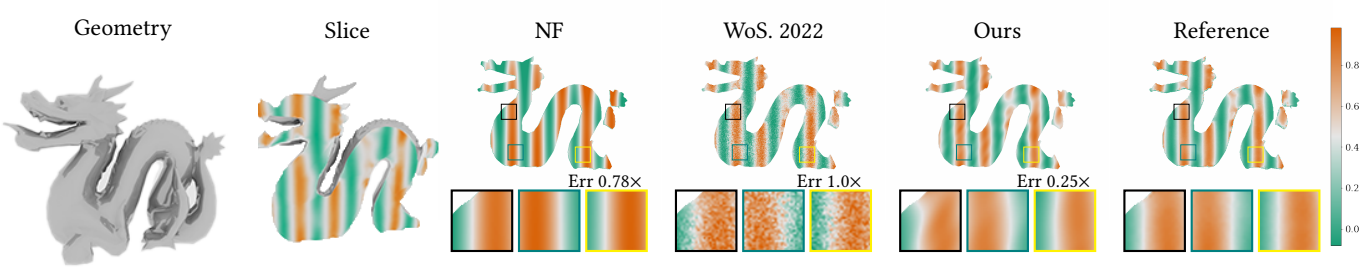

Figure 1: We visualize a slice of the solution to an elliptic PDE within a dragon-shaped boundary. Our hybrid solver can reduce the error of the neural field baseline, while achieving lower variance compared to the Walk-on-Spheres [Sawhney et al. 2022] method when working within the constraints of a limited computing budget

ABSTRACT

This paper presents a method that uses neural networks as a caching mechanism to reduce the variance of Monte Carlo Partial Differential Equation solvers, such as the Walk-on-Spheres algorithm [Sawhney and Crane 2020]. While these Monte Carlo PDE solvers have the merits of being unbiased and discretization-free, their high variance often hinders real-time applications. On the other hand, neural networks can approximate the PDE solution, and evaluating these networks at inference time can be very fast. However, neural-network-based solutions may suffer from convergence difficulties and high bias. Our hybrid system aims to combine these two potentially complementary solutions by training a neural field to approximate the PDE solution using supervision from a WoS solver. This neural field is then used as a cache in the WoS solver to reduce variance during inference. We demonstrate that our neural field training procedure is better than the commonly used self-supervised objectives in the literature. We also show that our hybrid solver exhibits lower variance than WoS with the same computational budget: it is significantly better for small compute budgets and provides smaller improvements for larger budgets, reaching the same performance as WoS in the limit.

KEYWORDS

PDE Solver, Monte Carlo, Neural Fields, Geometry Processing

ACM Reference Format:
Zilu Li, Guandao Yang, Xi Deng, Christopher De Sa, Bharath Hariharan, and Steve Marschner. 2023. Neural Caches for Monte Carlo Partial Differential Equation Solver. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3610548.3618141

1 INTRODUCTION

Solving elliptic PDEs is critical for various computer graphics applications, including 3D reconstruction, animation, and physics simulation. Conventional PDE solvers, however, typically involve time-consuming and error-prone discretization of space with finite elements or meshes. Monte Carlo PDE solvers based on the Walk on Spheres (WoS) algorithm [Sawhney and Crane 2020; Sawhney et al. 2023, 2022] offer a way to circumvent these issues by estimating solution values without discretization. These solvers, however, suffer from high variance, making them slow as they require numerous samples to reduce the variance. This prevents their use in many applications with limited computing budgets.

An alternative to both discretized and Monte Carlo solvers is to use neural fields to approximate the solution to a PDE. Neural fields are a class of neural networks that take spatial coordinates as input and output values of a continuous field [Raissi et al. 2019; Xie et al. 2022]. Prior works have developed self-supervised losses that can be used to optimize a neural field so that it satisfies a given PDE

*Equal Contribution

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0315-7/23/12…$15.00
https://doi.org/10.1145/3610548.3618141

and boundary conditions [Raissi et al. 2019; Sitzmann et al. 2020]. Neural fields are typically compact, fast to evaluate, and expressive. However, training neural fields with such self-supervised losses can be unstable, and they tend to produce biased solutions. Since neural field-based solvers are fast but biased and MC-based solvers are unbiased but slow due to high variance, it is natural to ask whether a hybrid solver that combines these two methods can be developed to achieve controllable bias, low variance, and fast evaluation.

This paper takes the first step towards building a hybrid solver combining neural fields with WoS approaches for variable-coefficient elliptic PDEs. Inspired by previous work [Müller et al. 2021] that deploys neural radiance fields as a cache to accelerate Monte Carlo rendering, we hypothesize that a neural field cache can also be used to reduce the variance of a Monte Carlo PDE solver. Adapting this idea, we have developed a novel hybrid PDE solver that first trains a neural field, then uses it to decrease the cost and variance of evaluating the solution. In the training phase, we optimize the neural field supervised by unbiased solution estimates from the WoS algorithm. Then to evaluate the solution, we run WoS but terminate its random walks at a prescribed depth by querying the neural field, providing solution estimates that are more accurate than the neural field alone and faster and less noisy than WoS alone.

To transfer the success of Müller et al. [2021] to reduce the variance of Monte Carlo PDE solvers, we identify the necessary change of neural network architecture and modification of the training procedure. We provide a theoretical analysis showing that our loss retains comparable convergence guarantees to conventional SGD algorithms. In practical testing, we find that our neural field has a lower average error than the unbiased but noisy one-sample WoS estimator, and as the number of samples increases, our hybrid solver produces a lower error when the depth limit is set appropriately.

2 RELATED WORK

This paper draws inspiration from the existing literature on Monte Carlo PDE solvers and PDE solvers with neural networks.

Monte Carlo PDE Solvers. The idea of using the Monte Carlo method to solve PDEs can date back to Courant et al. [1967] and Forsythe and Leibler [1950]. The Walk on Spheres (WoS) algorithm, initially proposed by Muller [1956], estimates the solution of a PDE by simulating a random walk from which boundary and source contributions are accumulated. Sawhney and Crane [2020] further applied the WoS algorithm in geometry processing tasks. After this seminal work, a number of projects have extended WoS to allow variable coefficients [Sawhney et al. 2022], Neuman boundary conditions [Sawhney et al. 2023], PDE parameter inversion [Yılmazer et al. 2022], and different applications such as fluid simulation [Rioux-Lavoie et al. 2022]. Although these works extend Monte Carlo methods to a broader range of PDEs, they are still limited by shortcomings, such as the high variance and expensive computations, of Monte Carlo estimators. To tackle these, the computer graphics community has also developed methods such as boundary caching [Bakbouk and Peers 2023; Miller et al. 2023; Müller et al. 2021], importance sampling [Müller et al. 2017, 2019; Veach and Guibas 1995], and denoising [Chaitanya et al. 2017; Gharbi et al. 2019]. Some of these techniques have been applied in WoS solvers [Qi et al. 2022; Sawhney et al. 2022]. In this paper, we propose an alternative method to mitigate the shortcomings of Monte Carlo PDE solvers by incorporating neural networks. Similar ideas have been applied in path tracing and radiosity [Hadadan et al. 2021; Müller et al. 2019; Müller et al. 2020; Müller et al. 2021; Ren et al. 2013], and we draw inspiration from these applications. Most relevant to this paper is Müller et al. [2021], which trains a neural radiance field to be used as a cache for real-time Monte Carlo rendering. Applying this idea to Walk-on-Spheres PDE solvers, however, is nontrivial. We identify the correct network architectures and training procedures to address such domain differences and provide theoretical analysis for convergence rates.

Neural Fields. Recently, neural fields have been shown to be a unique signal representation tool with the advantages of allowing high fidelity reconstruction [Mildenhall et al. 2020; Müller et al. 2022; Sitzmann et al. 2020; Tancik et al. 2020], enabling sampling at arbitrary locations , and providing fast training and inference [Chan et al. 2022; Chen et al. 2022b; Fridovich-Keil et al. 2022; Müller et al. 2022]. Most applications of neural fields have focused on image compression [Martel et al. 2021], view synthesis [Barron et al. 2021; Liu et al. 2020; Mildenhall et al. 2020; Verbin et al. 2021], 3D reconstruction [Mescheder et al. 2019; Park et al. 2019; Peng et al. 2020; Wang et al. 2021] and generation [Cai et al. 2020; Chen et al. 2020; Chen and Zhang 2019; Mescheder et al. 2019; Park et al. 2019; Yang et al. 2019]. Recently, Sitzmann et al. [2020] showed that neural fields with an appropriate architecture can be used to solve PDEs. Yang et al. [2021a] leveraged this idea and applied these neural field solvers to geometry processing tasks. Other researchers have successfully applied neural fields in character animation [Bergman et al. 2022; Noguchi et al. 2021], applications of level-set methods [Mehta et al. 2022], and solving time-dependent PDEs [Chen et al. 2022a]. Most of these existing methods aim to produce a network that deterministically approximates the physical field of interest, and accuracy can only be improved with additional training supervision. In contrast, we combine a neural field inside a WoS solver, so that it can produce better results when given more compute at test time.

Other Deep Learning-based PDE solvers. Another class of neural network-based PDE solvers is commonly known as neural operators [Li et al. 2020, 2021], which has shown success in many applications [Liu et al. 2022; Pathak et al. 2022; Trifan et al. 2021; Yang et al. 2021b]. A neural operator will train a neural network to predict how to evolve a physical system, learning from prior data generated from the simulation. Unlike neural operators, we will focus on learning a representation for the PDE solution. Improving our method to use data drive priors is an exciting direction, but it's out-of-the-scope for our discussion. Most closely related to the neural field PDE solver is a class of solvers called Physics-informed neural networks (PINNs) [Raissi et al. 2019], which has been applied in many PDE applications, including turbulence [Hennigh et al. 2021], elasticity [Rao et al. 2021], and topological optimization [Zehnder et al. 2021]. PINNs aim to train a neural network to approximate the PDE solution via a self-supervised loss derived from the PDE constraint and boundary conditions. We instead propose an alternative way to train neural fields as a representation of the PDE solution using labels provided by the WoS estimator, which works well in the class of PDE that the WoS estimator can be applied to.

3 BACKGROUND

Our work builds on two bodies of literature: Monte Carlo PDE solvers and neural-field PDE solvers.

3.1 Steady State Elliptic PDEs

Elliptic equations are a general class of PDEs that are important for various computer vision and graphics applications including (screened) Poisson surface reconstruction [Kazhdan et al. 2006; Kazhdan and Hoppe 2013] and fluid simulation [Rioux-Lavoie et al. 2022; Stam 1999]. In this paper, we are interested in obtaining the steady-state solution of an elliptic PDE.

Let $\Omega \subset \mathbb{R}^d$ denote the domain and $\partial\Omega$ be the boundary of this domain. ∇f denotes the gradient of f and $\nabla\cdot$ is the divergence operator $\nabla \cdot \mathbf{v}(x) = \sum_i \partial \mathbf{v}(x)_i / \partial x_i$. The class of Elliptic equations we consider in this paper can be expressed in the following form:

$$\begin{aligned} \nabla \cdot (\alpha(x)\nabla u(x)) + \vec{\omega}(x)\nabla u(x) - \sigma(x)u(x) = -f(x) \quad x \in \Omega \\ u(x) = g(x) \quad x \in \partial\Omega, \end{aligned} \tag{1}$$

where $\alpha : \mathbb{R}^d \to \mathbb{R}$, $\vec{\omega} : \mathbb{R}^d \to \mathbb{R}^d$ and $\sigma : \mathbb{R}^d \to \mathbb{R}$ are spatially varying coefficients. $f : \mathbb{R}^d \to \mathbb{R}$ denotes the source term, and $g : \mathbb{R}^d \to \mathbb{R}$ is the boundary condition. We will introduce two different ways to solve elliptic PDEs without discretization.

3.2 Monte Carlo PDE Solvers

The general idea of Monte Carlo PDE solvers is to express the solution of a PDE in the form of a recursive integral equation, and then define a Monte Carlo estimator for the integral equation. While our method can potentially be applied to other types of Monte Carlo PDE solvers, this paper focuses on PDEs in the form of Equation 1.

This PDE can be solved as an integral equation of the form:

$$u(x) = S(x) + \int_{B_r(x)} u(y)G_x(y)dy + \int_{\partial B_r(x)} u(z)K_x(z)dz, \tag{2}$$

where S, G_x, and K_x, which are functions depending on $\alpha(x)$, $\sigma(x)$, and $f(x)$. $B_r(x)$ is a ball centered at x with radius r: $\{y \mid \|x-y\| < r\}$, and $\partial B(x)$ is the sphere: $\{y \mid \|x-y\| = r\}$. Specifically,

$$S(x) = \int_{B(x)} \frac{f(y)G^{\bar{\sigma}}(x,y)}{\sqrt{\alpha(x)\alpha(y)}}dy, \quad K_x(z) = \sqrt{\alpha(z)}P^{\bar{\sigma}}(x,z) \tag{3}$$

$$G_x(y) = \sqrt{\alpha(y)/\alpha(x)}(\bar{\sigma} - \sigma'(y))G^{\bar{\sigma}}(x,y) \tag{4}$$

where $\bar{\sigma} = \max(\sigma'(x)) - \min(\sigma'(x))$. $G^{\bar{\sigma}}$ is the Green's function and $P^{\bar{\sigma}}$ is the Poisson kernel. A more detailed definition of these functions is provided by Sawhney et al. [2022].

Sawhney et al. [2022] provided the following Monte Carlo estimator for Equation 2, that uses delta-tracking [Coleman 1968; Raab et al. 2008] to avoid exponential number of walks:

$$\hat{u}(x) = \begin{cases} g(\bar{x}) & \text{if } d(x) < \epsilon \\ (\hat{S}_x(y_i) + G_x(y_i)\hat{u}(y_i))P_N(x)^{-1} & \text{w. prob. } P_N(x) \\ (\hat{S}_x(y_i) + K_x(z_i)\hat{u}(z_i))(1-P_N(x))^{-1} & \text{otherwise} \end{cases} \tag{5}$$

In this solver, $\hat{S}$ is a single sample Monte-Carlo estimator for the source contribution S, y_i is sampled from $B_{d(x)}(x)$, and z_i are sampled from $\partial B_{d(x)}(x)$. The function $d(x) = \min_{y\in\partial\Omega}\|y-x\|$ is the minimum distance of x to the boundary, $\bar{x} = \arg\min_{y\in\partial\Omega}\|y-x\|$ is the nearest projection of x the boundary, and ϵ defines a band around the boundary where walks will be terminated. When the first branch is not evaluated, the second branch will be evaluated with probability $P_N(x)$. The third branch will be evaluated if neither the first and the second is evaluated. Please refer to Sawhney et al. [2022] for definitions of $\hat{S}$ and P_N.

Limitations. While these Monte Carlo solvers are guaranteed to be unbiased, they experience very high variance due to the large space they need to integrate, so one needs to sample many independent walks to achieve good results. Moreover, each walk can be expensive, since the walk presented in Equation 5 can take hundreds of steps to reach the boundary. As a result, these solvers usually require additional variance reduction techniques.

3.3 PDE solvers using Neural Fields

A neural field is a neural network that takes the coordinates of an m-dimensional spatial point $\vec{x}$ and outputs a field value: $u_\theta : \mathbb{R}^n \to \mathbb{R}^m$ [Xie et al. 2022]. If u_θ is constructed to be smooth and continuous, then the spatial gradients such as $\partial u_\theta/\partial x_i$ can be obtained via automatic differentiation. These properties have been leveraged by prior works to apply neural fields to solve PDEs [Chen et al. 2022a; Raissi et al. 2019; Sitzmann et al. 2020] and to perform geometry processing [Mehta et al. 2022; Yang et al. 2021a]. For the PDE in Equation 1, we can define the following training objectives:

$$\mathcal{P}(\theta, x) = \|(u_\theta - \nabla\cdot(\alpha\nabla u_\theta) - \vec{\omega}\nabla u_\theta + \sigma u_\theta + f)(x)\|^2 \tag{6}$$

$$\mathcal{B}(\theta, x) = \|(u_\theta - g)(x)\|^2 \tag{7}$$

The loss $\mathcal{P}$ is trying to enforce the PDE condition, and the loss $\mathcal{B}$ is trying enforce the boundary condition. Solving Equation 1 can be formulated as an optimization problem [Raissi et al. 2019; Sitzmann et al. 2020; Yang et al. 2021a]

$$\arg\min_\theta \int_\Omega \mathcal{P}(\theta, x)\,\mathrm{d}x + \lambda \int_{\partial\Omega} \mathcal{B}(\theta, x)\,\mathrm{d}x, \tag{8}$$

where λ is a hyperparameter that balances the PDE constraints and boundary constraints.

Network architectures. The network architecture is chosen to strike a good balance between expressiveness and regularization to obtain good performance. A popular choice is a multi-layer perceptron (MLP) with sinusoidal activations [Sitzmann et al. 2020] or Fourier positional encoding [Lindell et al. 2022; Tancik et al. 2020; Yang et al. 2022; Zhang et al. 2020]. These MLPs are very compact to store, yet they can be slow to train and expensive to evaluate. Another class of of neural field architecture modulates an MLP with interpolated spatial features [Müller et al. 2022]: $u_\theta(x) = f_{\theta_{n+1}}(g(\text{interp}(x,\theta_1),\ldots,\text{interp}(x,\theta_n))$, where interp is bi-linear interpolation, $g(v_1,\ldots,v_n)$ is an aggregation operation, and f_θ is a small MLP. These neural fields with spatially modulated features are usually fast to converge, but they are harder to regularize to produce a good solution when supervision is limited.

Limitations. Once successfully trained, these neural fields can produce an approximate solution to the PDE very efficiently since only a forward pass is required to evaluate the field. However, training such neural fields using a self-supervised loss can be difficult.

In order for the training to converge, one needs to choose appropriate network architectures, initialization, as well as the learning rate schedule. For example, using a network with piecewise linear activation will not work since the Laplacian of the network will be zero [Lei and Jia 2020], even though the network is still a universal approximator. Also, the residual error between the neural field and the exact solution depends on the architecture, the training procedures, and the characteristics of the PDE, making it difficult to control the amount of bias in a neural field solution.

4 METHOD OVERVIEW

On one hand, the WoS method is slow at inference time due to high variance, but it has no bias. On the other hand, the neural field is comparatively fast at inference time since it produces deterministic output without variance, but it suffers from non-zero bias. Inspired by these complementary properties, we want to build a hybrid solver where we can reduce the inference time for WoS by querying the neural field after a fixed compute budget.

We achieve this hybrid solver in two steps. First, we need to build a mechanism to reliably train neural field solutions. Instead of using the self-supervised loss, we proposed to use a WoS estimator to provide target data to supervise the neural field to approximate the PDE solution (Section 5). Once we obtain a neural field with a small enough error, we use a hybrid WoS solver that terminates the recursive call in Equation 5 by querying the neural field (Section 6). Intuitively, this hybrid solver can lower the error of the neural field solution since it performs WoS-style random walks that can terminate at the boundary. At the same time, it can achieve lower variance than the WoS estimator since it conducts shorter walks, thus exploring a smaller sample space.

5 TRAINING A NEURAL FIELD SOLUTION

To build a hybrid solver, we first need to obtain a neural field that approximates the solution of the PDE in Equation 1. Specifically, the network u_θ will take a 2D or 3D spatial coordinate x as input and output a real number to approximate the ground truth $u(x)$.

One way to achieve this is to directly use a self-supervised loss like the one in Equation 8. Training with this type of loss can be unstable and often requires extensive hyper-parameter tuning. For example, Figure 3 shows that for the self-supervised loss, performance is sensitive to hyper-parameters such as network architecture.

One potential reason for such instability is the higher-order differential operator used in the self-supervised loss. If neural fields need to be expressive enough to approximate arbitrary solutions, the network needs to contain high-frequency components. The derivative operators will further amplify the contribution of these high-frequency components. As a result, the self-supervised loss can be high-frequency, making it difficult to optimize.

In this paper, we circumvent this issue by proposing loss functions that do not require evaluating gradients of the network. The key idea is that the WoS estimator provides statistical estimates of the exact solution to the PDE, and these estimates can be used as targets to train the neural networks. We will demonstrate the derivation of the WoS-supervision loss in Section 5.1, analyze its convergence properties in Section 5.2.

5.1 WoS Supervision Loss

One way to supervise the neural field solution without taking spatial derivatives is to use the MC estimator to create supervision targets to optimize the neural network. The straightforward way to achieve this is first running the Monte-Carlo estimator for enough iterations to accurately estimate the PDE solution for a fixed set of spatial locations $\{x_i \in \Omega\}_{i=1}^n$. Then we can use SGD to optimize the L2 objectives between the network's output and the estimated target:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(u_\theta(x_i) - \frac{1}{N}\sum_{j=1}^{N}\hat{u}(x_i)\right)^2, \tag{9}$$

where $\hat{u}$ is an unbiased Monte Carlo estimator for the PDE solution so $\mathbb{E}[\hat{u}] = u$ in Ω. Computing this target can take a substantial amount of time since it requires running N independent walks for each training location x_i.

Intuitively, we need not wait to start training until the supervision signal is very accurate, since SGD-based neural network training can tolerate noisy gradients. This suggests that we can run the data acquisition process and the neural network training process in parallel, similar to Müller et al. [2021].

To achieve this, we first create a running estimate y_i of the PDE solution at each location x_i target using the WoS estimator $\hat{u}$:

$$y_i^{(k+1)} = (k y_i^{(k)} + \hat{u}(x_i))/(k+1), \tag{10}$$

where k denotes the number of accumulation steps. The training loss at step t is simply the L2 loss between the network prediction and the accumulated sample average:

$$\mathcal{L}_t(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\|u_\theta(x_i) - y_i^{(t)}\right\|^2. \tag{11}$$

By using a loss defined in this way with a target value $y_i^{(t)}$ that improves in accuracy as training progresses, we are able to run the sample generation in parallel with training and also converge provably to low-noise results.

5.2 Convergence analysis

Intuitively, when training a neural field using stochastic gradient descent (SGD) with the loss function in Equation 11, the variance of the target from the Monte-Carlo estimator will be added to the variance created by SGD. As long as this estimator is unbiased, SGD can converge to a region with small gradients.

Theorem 5.1. *Let u be the solution of the PDE of interest, and $\hat{u}$ be an unbiased WoS estimator for the solution with bounded variance: $\mathbb{E}[\hat{u}] = u$ and $\mathbb{V}[\hat{u}] < C$. Let u_θ be the neural field to be optimized. Define the final objective to be $\mathcal{L}_i(\theta, y) = (u_\theta(x_i) - y_i)^2$. Further, assume that $\|\nabla_\theta u_\theta(x)\| \le F$, $\left\|v^T\left(\nabla_\theta^2 \mathcal{L}\right)v\right\| \le L\|v\|$ for all v, and $\|\nabla_\theta \mathcal{L}_i(\theta, u)\| \le G$. If we run the following SGD optimization for T steps: $\theta_{t+1} = \theta_t - \alpha\nabla_\theta \mathcal{L}(\theta, y_i^{(t)})$, where $y_i^{(t+1)}$ is obtained through Equation 10, then the expected gradient norm will converge at the following rate:*

$$\mathbb{E}[\nabla_\theta \mathcal{L}(\theta_\tau)] \le \frac{\mathcal{L}(\theta_0) - \mathcal{L}(\theta^*)}{T\alpha} + \frac{LCF^2}{2}\frac{\log(T)}{T} + \frac{LG^2\alpha}{2}, \tag{12}$$

where τ is a random variable on $[0, \ldots, T-1]$ indicating which step to stop, $P(\tau = t) = 1/T$, T is the maximum number of SGD steps, and

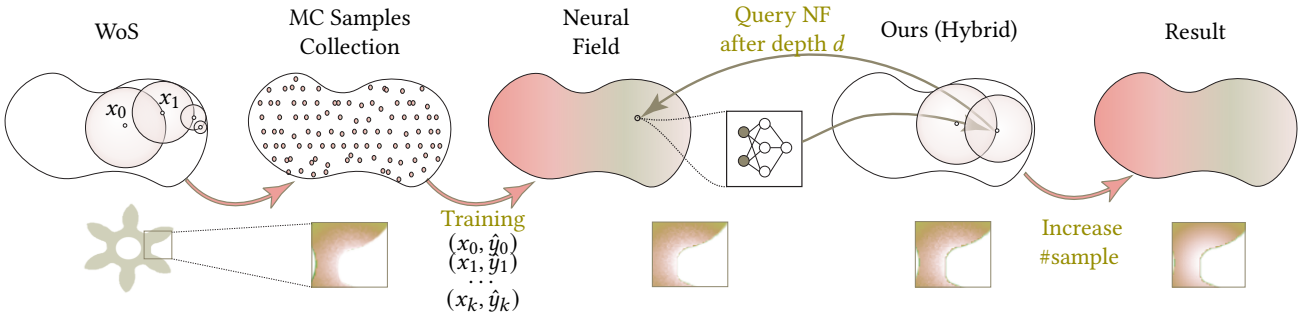

Figure 2: Illustration of our training and testing pipeline. We first use the WoS algorithm to generate target data for training a neural field approximation of the PDE solution. Once the neural field is trained, it will be used as cache in the hybrid solver, enabling the random walks to terminate early.

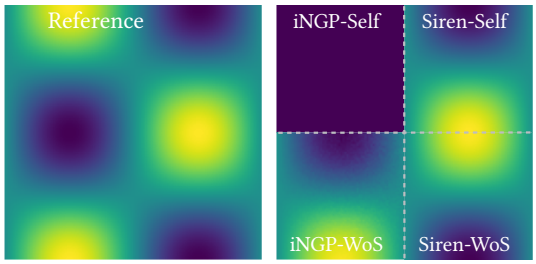

Figure 3: Result of training Neural Fields with the self-supervised loss and with WoS supervision. Here we train two different neural field architectures, SIREN and iNGP, using each of these losses. Swapping from SIREN to iNGP causes catastrophic failure when using self-supervision. Our loss produces good results for both architectures.

$\mathcal{L}(\theta_t)$ is the averaged loss over different sampling locations at time t: $\frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i(\theta, y_i^{(t)})$. The expectation is taken over stopping time τ, randomly generated target $y_i^{(t)}$, and optimization trajectory $\theta_{1,\ldots,T}$.

The proof is provided in the supplementary. This theorem suggests that supervising SGD with signals created from an unbiased WoS estimator will converge to a noise ball of size $LG^2\alpha/2$, which is the same as SGD algorithms. The convergence rate is understandably slower (i.e. $O(log(T)/T)$) compared to SGD training with variance-free labels (i.e. $O(1/T)$). This is because additional steps are required to average out the variance introduced by y_i^t.

Note that Theorem 5.1 only shows the most basic convergence analysis for constant learning rate SGD with labels created by WoS estimators. This convergence rate can be easily improved by standard techniques such as reducing the learning rate, adding momentum, or reducing the variance of the MC samples.

```
def train(net, WoS, domain, n_points):
  x = WoS.domain_sample(n_points)          # Initialize dataset
  data = {x_i: (0, 0) for x_i in x}
  for i in range(max_training_iters):
    xi, yi, ci = sample_batch(data)        # Sample a batch.
    y_new = WoS(xi)                        # Obtain new MC estimate
    yi = (yi * ci + y_new) / (ci + 1)      # Estimate new label.
    loss = ((net(xi) - yi)**2).mean()      # Compute loss.
    net = Adam(net, lr(t), loss.grad())    # SGD updates.
    data[xi] = (yi, ci + 1)                # Dataset update.
  return net
```

Listing 1: Training algorithm (Sec 5).

6 WALK-ON-SPHERES WITH NEURAL CACHE

Now we have a Monte Carlo solver $\hat{u}$ and a neural field $u_\theta(x)$ approximating the solution. How do we construct a hybrid solver with lower variance than the Monte Carlo solver but can also have controllable bias like the Monte Carlo solver? We propose to achieve this is by replacing the recursive call in the Monte Carlo solver of Equation 5 with neural network inference after a certain depth.Specifically, we define a hybrid solver in the following form:

$$\hat{u}_H(x,m) = \begin{cases} g(\bar{x}) & \text{if } d(x) < \epsilon \\ u_\theta(x) & \text{if } m = 0 \\ \frac{\hat{S}_x(y) + G_x(y)\hat{u}_H(y,m-1)}{P_N(x)} & \text{w. prob. } P_N(x) \\ \frac{\hat{S}_x(y) + K_x(z)\hat{u}_H(z,m-1)}{1-P_N(x)} & \text{otherwise} \end{cases}, \tag{13}$$

where $y \sim B_{d(x)}(x)$, $z \sim \partial B_r(x)$, and $\bar{x}$ is a projection of x to the closest point on the boundary: $\bar{x} = \arg\min_{y\in\partial\Omega} \|x - y\|$.

The idea of this solver is to run a random walk following the WoS algorithm, but keeping track of the number of steps, which is the key indicator of computational cost. The walk can terminate in two ways. If the walk reaches the boundary, then the solver returns the boundary condition g as in standard WoS. If the walk doesn't reach the boundary within the budgeted number of steps m, it queries the approximate solution from the neural field u_θ.

This hybrid solver generalizes both the neural field and the WoS estimator, and the parameter m lets us explore the trade-off between bias (from the neural field) and variance (from the Monte Carlo estimator). When calling $\hat{u}$ with $m = 0$, it simply returns the neural field value; there is no variance, but the bias is the approximation error of the neural field. When setting $m = \infty$, it computes exactly the WoS estimator, which has no bias but can have substantial variance. For intermediate values of m, the hybrid method can achieve lower bias than the NF alone and lower variance than the WoS estimator. This is supported by experiment results in Sec 7.4.

When dealing with low computation budgets, the parameter m can be assigned a low value or even set to zero, resulting in fast evaluation with minimal or no variance. When more computation is available, m can be set higher to decrease bias. But each sample can create walks with up to m steps, which results in higher variance (due to the large sample space caused by longer walks). As a result, as the depth increases, the hybrid solver will require more samples,

thus increasing the computational cost. However, in Section 7.4, we show that the hybrid estimator can achieve lower overall error in practice for most ranges of accuracy.

```
def test(net, WoS, x, m):
    dist, x_proj = WoS.domain.nn_query(x)
    if dist < eps: return WoS.boundary(x_proj)
    if m == 0:     return net(x)
    y, z = WoS.sample_walk(x)
    source = WoS.compute_source(x, y)
    if WoS.is_null(x):
      x_next, coef = y, WoS.compute_null_coef(x, y)
    else:
      x_next, coef = z, WoS.compute_non_null_coef(x, z)
    return test(net, WoS, x_next, m - 1) * coef + source
```

Listing 2: Hybrid solver inference (Sec 6).

7 RESULTS

This section will first compare the proposed hybrid solver and two baselines: pure WoS and pure neural field solutions trained with self-supervised loss. We first show equal-time comparisons on a 3D elliptic equation with spatially varying coefficients (Sec 7.1). We then provide an analysis under the same number of samples (Sec 7.3). Finally, we demonstrate how the depth hyperparameter of our hybrid solver allows users to trade off compute for bias (Sec 7.4).

7.1 Experiment Setup

In this section, we apply our solvers and the baseline solvers to the PDE in Equation 1 without a drifting term:

$$\begin{aligned} \nabla \cdot (\alpha(x)\nabla u(x)) - \sigma(x)u(x) &= -f(x) \quad x \in \Omega \\ u(x) &= g(x) \quad x \in \partial\Omega. \end{aligned} \tag{14}$$

This experiment section will focus on variable coefficient PDEs. Please refer to the supplementary for the results of combining our methods with constant coefficient WoS algorithms.

Domain representation. The domain Ω is defined by a signed distance function (SDF). In this experiment, we represent the domain SDF by training a neural field in instant-NGP [Müller et al. 2022; Park et al. 2019; Sitzmann et al. 2020]. We follow the training procedure of Müller et al. [2022] except that 1) we do not deploy hashing to compress the storage, and 2) we use multi-resolution grids with only four layers with following resolutions: 128, 64, 32, and 16. Our model takes about 20 minutes to finish training and contains 4MB parameters.

Diffusion, absorption, forcing, and boundary functions. Following Sawhney et al. [2022], we use smooth periodic patterns to generate the spatially varying diffusion, absorption, forcing, and boundary functions. Please refer to the released code for more details.

Baselines. Our baselines for comparison are (1) Walk-on-spheres solver (WoS) and (2) neural fields trained with self-supervised techniques (NF). For the WoS baseline, we follow the released C++ implementation of Sawhney et al. [2022]. We used the Delta Tracking algorithm, with importance sampling on the off-centered Green's function but didn't apply the next-flight variant. This WoS solver is also used as the basis for the hybrid solver. For the NF baseline, we implement the self-supervised loss as shown in Equation 8. This neural field baseline is trained for 2×10^4 iterations, with the best hyper-parameter obtained Ray-tune [Liaw et al. 2018] random search. The training of this baseline takes about 5 minutes to finish.

Reference solution. We use the unbiased WoS estimators to produce the reference solution. Specifically, for each geometry, we choose a slice of interest (i.e. the $z = 0$ plane). At that plane, we densely sample an image with resolution 512×512 unless otherwise noted. We average 10^4 WoS samples for each of these pixels to create the reference value. For pixels that are outside the domain, we set the reference to 0.

Hyperparameters for hybrid solvers. To obtain the neural field approximation using the method proposed in Section 5, we uniformly sample 20000 points inside the domain via rejection sampling. For each of these points, we run WoS algorithm with 50 walks to obtain the initial training label $y^{(0)}$. Then we start the training procedure for 20000 iterations using Adam optimizer. For every 5000 iterations, we sample another batch of data from WoS with 50 walks. We do not interrupt the training until the next batch of data accumulation of 50 walks is ready. To compute WoS for 50 independent walks for all data points takes about 40 seconds, during which we can run about 8000 training iterations. This allow us to update the training data at every 5000 iterations without waiting. We use SIREN with 512 hidden dimensions and 2 hidden layers. The training is done in NVIDIA RTX 2080 Ti GPUs and can be finished within 4 minutes. We implement the training pipeline and WoS solver using the automatic differentiation framework Jax [Bradbury et al. 2018].

7.2 Equal Time Comparison

We present qualitative results for three shapes: Sprocket, Missile [Koch et al. 2019], and Cow [Crane et al. 2013]. For each of these shapes, we allocate 5 minutes of compute time to obtain the result. We also report the mean square error for each frame. The results are shown in Figure 4. Our network is more accurate than the NF baseline, with a lower MSE compared with the reference solution. For our hybrid solution, we set $m = 1$. Compared to the WoS baseline, our network shows less noise, which is also reflected through both the cleaner image and the lower MSE error. This result verifies our hypothesis that our hybrid solver is able to produce less biased results than the NF baseline and also achieves lower variance than the WoS baseline using a similar amount of computing resources. Note that our method performs better when the solution function has higher spatial frequency (e.g. Missile and Cow example).

To further understand the allocation of time for our method and the WoS baseline, we report a detailed compute time breakdown in Figure 6. We report the amount of time it takes for both the WoS and the hybrid model to reach an MSE error of 5e-3 at different resolutions. We report three different resolutions: 256, 512, and 1024. For the hybrid model, we show the breakdown of training time and inference time. The hybrid model can outperform WoS in time efficiency as long as there are sufficient locations whose solution needs to be computed during inference. Its advantage is larger when the number of testing locations is larger (e.g. 1024×1024). This is because it is very fast for the hybrid method with shallow depth (e.g. $m = 1$) to produce a good estimate during inference. The training time remains unchanged regardless of the inference time resolution. As a result, when the slice resolution is large, the hybrid model can

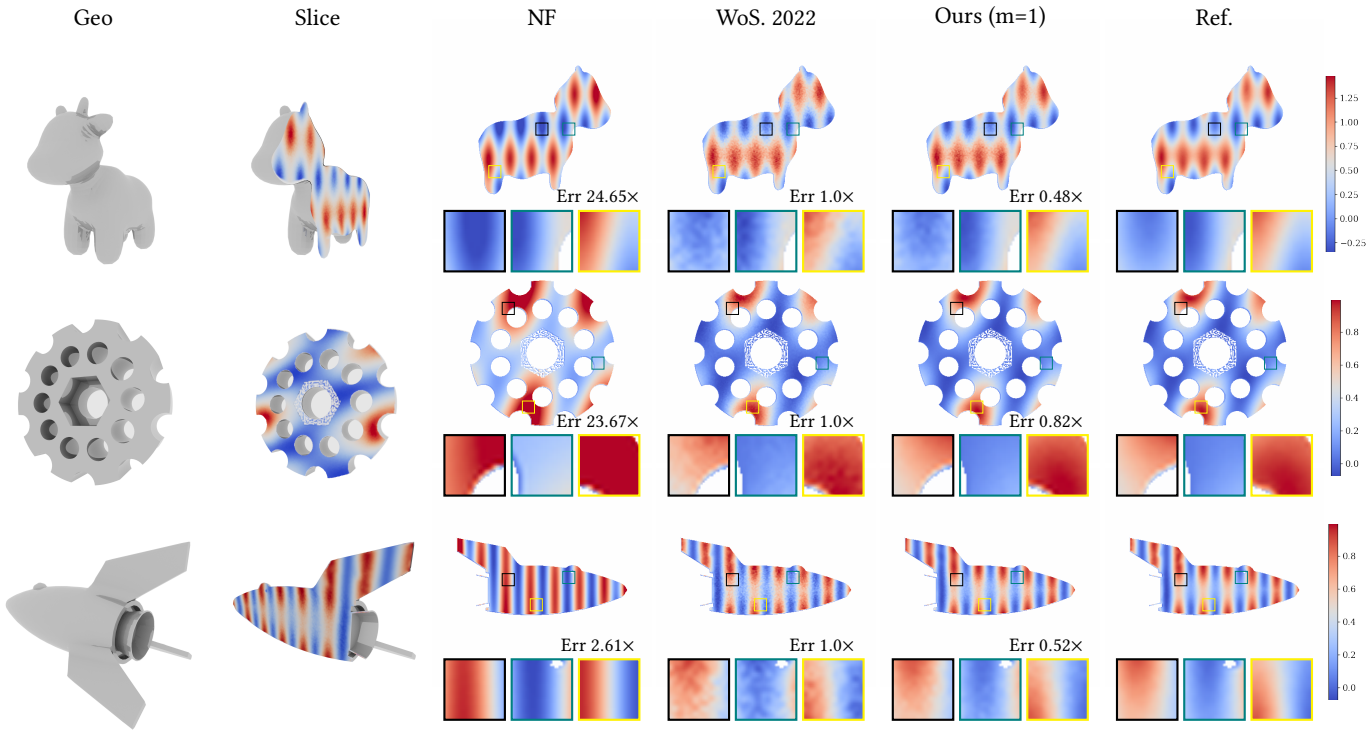

Figure 4: Equal time comparison. We solve a variable coefficient screened Poisson equation (Eq 1) with different domain shapes, including high-genus shapes (Row 2) and shapes with thin structures and sharp edges (Row 3). We sample a 512×512 slice for each method and allocate 5 minutes of compute time to obtain the result. We can see that the hybrid solver can achieve more accurate results than both the self-supervised baseline (with high bias) and the WoS baseline (with high variance). The performance gap is larger when the PDE solution is high-frequency.

amortize the training cost, which leads to a larger performance gap compared to the WoS baseline.

7.3 Equal Sample Analysis

Equal-time comparison can be unreliable since the compute time also tightly depends on implementation and hardware. To further compare the performance among these solvers, we conduct an equal sample analysis. For each solver, we averaged the output of k independent samples. This averaged output is then used to compute the MSE with the reference solution. In the left-hand side of Figure 5, we plot the MSE versus the number of samples in log scale. The NF baseline using self-supervised loss failed to produce accurate results and lacks any provision to reduce its error during inference time. This is indicated by a constant high MSE throughout different numbers of test-time samples. This error is also visible in the slice figure titled NF, which has different values near the ear and the front foot of the bunny. This can be caused by the training instability of the self-supervised loss, which requires additional hyperparameter tuning. On the other hand, our neural field trained with WoS supervision (Section 5.1, `Ours(m=0)`) has lower MSE and better qualitative results compared to the NF baseline. The qualitative result of `Ours(m=0)` is also closer to the reference.

Compared to WoS baseline, our hybrid solvers achieve lower MSE error when the number of walks per testing location is low. But as more walks are allowed during inference, the performance of hybrid solvers plateaus while the WoS baseline is able to sustain the $O(1/N)$ convergence rate with N being the number of independent walks. This is expected since the hybrid solver terminates by evaluating the neural cache u_θ, which is only an approximation of the actual solution u. Overall, the result suggests that our hybrid solver has an advantage over both the NF baseline and the WoS baseline when computation resource is limited.

7.4 Effect of Cache Depth

Finally, we want to analyze the key hyperparameter of our hybrid solver, the depth budget m in Eq 13. We want to study how this hyper-parameter allows us to trade off bias and compute. To achieve this, we first run our hybrid models but set the depth budget to different numbers: $m = 0, 1, 5$. As in the previous section, we sample 10^4 independent walks for each model. To produce outputs at the k^{th} step, we average k sampled walks. Each of these outputs is compared to the reference solution to compute MSE. Figure 7 shows a graph of the number of samples versus MSE.

First, as m increases, the curve plateaus at lower MSE. Intuitively as m increases, the hybrid solver should behave more and more like the WoS solver since more and more walks can terminate within the budget, making fewer (biased) neural network queries. At the same time, increasing m will result in more compute per walk since the random walk might not terminate until it hits the budget. Thus each such walk will take a longer time to finish. This suggests that

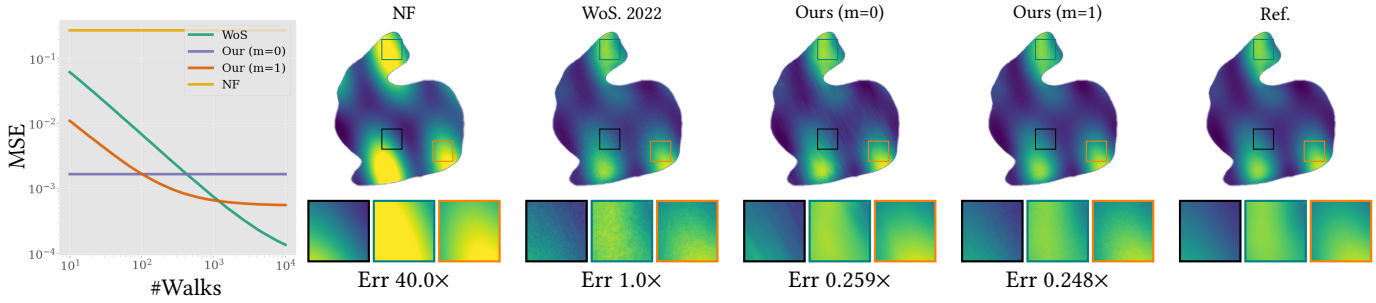

Figure 5: Equal sample convergence analysis. L: we show number of walks v.s. MSE curves. Our method achieves lower MSE when number of walks is limited. R: a qualitative comparison of different models at 100 walks per pixel.

Figure 6: Computational time breakdown for WoS and our method. We report the time it took each of the methods to reach an MSE error of less than 5e-3. Our hybrid solver is faster than WoS when there are more test samples.

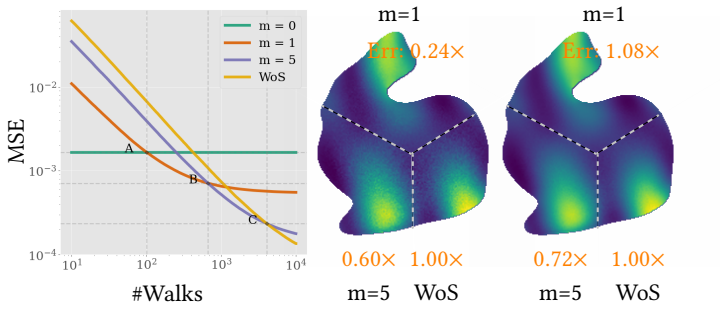

Figure 7: L: Convergence rate with different depths. The crossing points A, B, and C show that given different compute budgets we should choose different parameters m. R: Visualization at different depths at 80 and 1300 walks.

if the user desires more accurate results, they are able to achieve this by increasing the depth of the hybrid solver.

Second, as the depth m increases, the MSE curve starts higher. Intuitively, we can think of the hybrid solver as computing the expectation of a function that takes a random walk $x_1, \ldots, x_d$ with length m and return a value. A larger m means the expectation is taken over a larger space, which might require more samples to compute accurately. As a result, when the compute budget is limited, it's beneficial to use the hybrid solver with smaller m.

Finally, we are able to expand the advantages of our hybrid solver over the WoS solver at different compute budgets by considering different values of m. For example, when we only consider the configuration of $m = 0$ and $m = 1$, our hybrid solver is better than WoS when the budget is less than about 1000 walks, shown in the plot at the left-hand-side of Figure 7. This can be seen by the crossing point of the `m=1` and `WoS` curves. But when we expand the set of hybrid solvers to include $m = 0$ through $m = 5$, our hybrid solver outperforms WoS until about 4071 walks per sample, which is indicated by the crossing point of `m=5` and `WoS`.

8 CONCLUSION

In this paper, we propose to use neural fields to effectively reduce the variance of the Walk-on-spheres Monte Carlo PDE solvers. First, we develop a simple and effective training objective to obtain a neural field that approximates the PDE solution. Then, it's used as a cache to reduce the length of random walks from the WoS solvers. We also provide a convergence analysis of the proposed training algorithm showing similar convergence properties to SGD. Empirically, we show that our hybrid solver can reduce the variance of the WoS solver when working within a limited computational budget. It can converge to an unbiased solver as we increase the compute budget.

Currently, our method only applies to a specific class of PDEs and boundary conditions where there exists an unbiased Monte Carlo solver. Applying to a broader class of recursive Monte Carlo estimators is an interesting direction for future research. Our method also requires the manual selection of certain key hyper-parameters such as the inference depth budget. An automatic way to determine these hyper-parameters would be helpful to make our method applicable to more examples. To scale our method to larger scenes, we may need to efficiently train a neural field to approximate a function within such large and complex geometry. Moreover, how to leverage recent advances such as instant NGP [Müller et al. 2022] to achieve PDE simulation in large scenes is also interesting. A comprehensive convergence analysis with empirical evidence can be beneficial for the community to better understand the algorithm.

Acknowledgement. This research was supported in part by the National Science Foundation under grant 2212084, grant 2144117, and by the RI-CAREER award 2046760. We want to thank Rohan Sawhney and Wenqi Xian for their discussions.

REFERENCES

Ghada Bakbouk and Pieter Peers. 2023. Mean Value Caching for Walk on Spheres. (2023).

Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. 2021. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5855–5864.

Alexander Bergman, Petr Kellnhofer, Wang Yifan, Eric Chan, David Lindell, and Gordon Wetzstein. 2022. Generative neural articulated radiance fields. *Advances in Neural Information Processing Systems* 35 (2022), 19900–19916.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: composable transformations of Python+NumPy programs. http://github.com/google/jax

Ruojin Cai, Guandao Yang, Hadar Averbuch-Elor, Zekun Hao, Serge Belongie, Noah Snavely, and Bharath Hariharan. 2020. Learning gradient fields for shape generation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*. Springer, 364–381.

Chakravarty R. Alla Chaitanya, Anton S. Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. 2017. Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder. *ACM Trans. Graph.* 36, 4, Article 98 (jul 2017), 12 pages. https://doi.org/10.1145/3072959.3073601

Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. 2022. Efficient Geometry-aware 3D Generative Adversarial Networks. In *Proc. CVPR*.

Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. 2022b. TensoRF: Tensorial Radiance Fields. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXII* (Tel Aviv, Israel). Springer-Verlag, Berlin, Heidelberg, 333–350. https://doi.org/10.1007/978-3-031-19824-3_20

Honglin Chen, Rundi Wu, Eitan Grinspun, Changxi Zheng, and Peter Yichen Chen. 2022a. Implicit Neural Spatial Representations for Time-dependent PDEs. *arXiv preprint arXiv:2210.00124* (2022).

Zhiqin Chen, Andrea Tagliasacchi, and Hao Zhang. 2020. Bsp-net: Generating compact meshes via binary space partitioning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 45–54.

Zhiqin Chen and Hao Zhang. 2019. Learning implicit fields for generative shape modeling. In *Proc. CVPR*.

W. A. Coleman. 1968. Mathematical Verification of a Certain Monte Carlo Sampling Technique and Applications of the Technique to Radiation Transport Problems. *Nuclear Science and Engineering* 32, 1 (1968), 76–81. https://doi.org/10.13182/NSE68-1

Richard Courant, Kurt Friedrichs, and Hans Lewy. 1967. On the partial difference equations of mathematical physics. *IBM journal of Research and Development* 11, 2 (1967), 215–234.

Keenan Crane, Ulrich Pinkall, and Peter Schröder. 2013. Robust fairing via conformal curvature flow. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–10.

George E Forsythe and Richard A Leibler. 1950. Matrix inversion by a Monte Carlo method. *Math. Comp.* 4, 31 (1950), 127–129.

Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. 2022. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 5501–5510.

Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. 2019. Sample-Based Monte Carlo Denoising Using a Kernel-Splatting Network. *ACM Trans. Graph.* 38, 4, Article 125 (jul 2019), 12 pages. https://doi.org/10.1145/3306346.3322954

Saeed Hadadan, Shuhong Chen, and Matthias Zwicker. 2021. Neural Radiosity. *ACM Trans. Graph.* 40, 6, Article 236 (dec 2021), 11 pages. https://doi.org/10.1145/3478513.3480569

Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. 2021. NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework. In *Computational Science–ICCS 2021: 21st International Conference, Krakow, Poland, June 16–18, 2021, Proceedings, Part V*. Springer, 447–461.

Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. 2006. Poisson Surface Reconstruction. In *Symposium on Geometry Processing*, Alla Sheffer and Konrad Polthier (Eds.). The Eurographics Association. https://doi.org/10.2312/SGP/SGP06/061-070

Michael Kazhdan and Hugues Hoppe. 2013. Screened Poisson Surface Reconstruction. *ACM Trans. Graph.* 32, 3, Article 29 (jul 2013), 13 pages. https://doi.org/10.1145/2487228.2487237

Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. arXiv:1812.06216 [cs.GR]

Jiabao Lei and Kui Jia. 2020. Analytic marching: An analytic meshing solution from deep implicit surface networks. In *International Conference on Machine Learning*. PMLR, 5789–5798.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2020. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895* (2020).

Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. 2021. Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794* (2021).

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. 2018. Tune: A Research Platform for Distributed Model Selection and Training. *arXiv preprint arXiv:1807.05118* (2018).

David B. Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. 2022. BACON: Band-limited Coordinate Networks for Multiscale Scene Representation. *CVPR* (2022).

Burigede Liu, Nikola Kovachki, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, Andrew M Stuart, and Kaushik Bhattacharya. 2022. A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids* 158 (2022), 104668.

Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. 2020. Neural Sparse Voxel Fields. In *NeurIPS*.

Julien N. P. Martel, David B. Lindell, Connor Z. Lin, Eric R. Chan, Marco Monteiro, and Gordon Wetzstein. 2021. ACORN: Adaptive Coordinate Networks for Neural Scene Representation. *ACM Trans. Graph. (SIGGRAPH)* 40, 4, Article 58 (2021), 13 pages.

Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. 2022. A Level Set Theory for Neural Implicit Evolution under Explicit Flows. *arXiv preprint arXiv:2204.07159* (2022).

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Proc. CVPR*.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. 2020. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In *Proc. ECCV*.

Bailey Miller, Rohan Sawhney, Keenan Crane, and Ioannis Gkioulekas. 2023. Boundary Value Caching for Walk on Spheres. (Feb. 2023). arXiv:2302.11825 [cs.GR]

Mervin E Muller. 1956. Some continuous Monte Carlo methods for the Dirichlet problem. *The Annals of Mathematical Statistics* (1956), 569–589.

Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Trans. Graph.* 41, 4, Article 102 (July 2022), 15 pages. https://doi.org/10.1145/3528223.3530127

Thomas Müller, Markus Gross, and Jan Novák. 2017. Practical Path Guiding for Efficient Light-Transport Simulation. *Comput. Graph. Forum* 36, 4 (jul 2017), 91–100. https://doi.org/10.1111/cgf.13227

Thomas Müller, Brian Mcwilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. 2019. Neural Importance Sampling. *ACM Trans. Graph.* 38, 5, Article 145 (oct 2019), 19 pages. https://doi.org/10.1145/3341156

Thomas Müller, Fabrice Rousselle, Alexander Keller, and Jan Novák. 2020. Neural control variates. *ACM Transactions on Graphics (TOG)* 39, 6 (2020), 1–19.

Thomas Müller, Fabrice Rousselle, Jan Novák, and Alexander Keller. 2021. Real-Time Neural Radiance Caching for Path Tracing. *ACM Trans. Graph.* 40, 4 (Aug. 2021).

Atsuhiro Noguchi, Xiao Sun, Stephen Lin, and Tatsuya Harada. 2021. Neural articulated radiance field. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5762–5772.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. 2019. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation, In Proc. CVPR. *Proc. CVPR*.

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. 2022. Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214* (2022).

Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. 2020. Convolutional occupancy networks, In Proc. ECCV. *arXiv preprint arXiv:2003.04618*.

Yang Qi, Dario Seyb, Benedikt Bitterli, and Wojciech Jarosz. 2022. A bidirectional formulation for Walk on Spheres. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 51–62.

Matthias Raab, Daniel Seibert, and Alexander Keller. 2008. Unbiased global illumination with participating media. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer, 591–605.

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. *Journal of Computational physics* 378 (2019), 686–707.

Chengping Rao, Hao Sun, and Yang Liu. 2021. Physics-informed deep learning for computational elastodynamics without labeled data. *Journal of Engineering Mechanics* 147, 8 (2021), 04021043.

Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. 2013. Global Illumination with Radiance Regression Functions. *ACM Trans. Graph.* 32, 4, Article 130 (jul 2013), 12 pages. https://doi.org/10.1145/2461912.2462009

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 (Nov. 2022), 1–16.

Rohan Sawhney and Keenan Crane. 2020. Monte Carlo geometry processing. *ACM Trans. Graph.* 39, 4 (Aug. 2020).

Rohan Sawhney, Bailey Miller, Ioannis Gkioulekas, and Keenan Crane. 2023. Walk on Stars: A Grid-Free Monte Carlo Method for PDEs with Neumann Boundary Conditions. (Feb. 2023). arXiv:2302.11815 [cs.GR]

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2022. Grid-Free Monte Carlo for PDEs with Spatially Varying Coefficients. (Jan. 2022). arXiv:2201.13240 [cs.GR]

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*.

Jos Stam. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 121–128. https://doi.org/10.1145/311535.311548

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. 2020. Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains. In *Proc. NeurIPS*.

Anda Trifan, Defne Gorgun, Zongyi Li, Alexander Brace, Maxim Zvyagin, Heng Ma, Austin Clyde, David Clark, Michael Salim, David J Hardy, et al. 2021. Intelligent Resolution: Integrating Cryo-EM with AI-driven Multi-resolution Simulations to Observe the SARS-CoV-2 Replication-Transcription Machinery in Action. *bioRxiv* (2021), 2021–10.

Eric Veach and Leonidas J. Guibas. 1995. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '95)*. Association for Computing Machinery, New York, NY, USA, 419–428. https://doi.org/10.1145/218380.218498

Dor Verbin, Peter Hedman, Ben Mildenhall, Todd E. Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. 2021. Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields. *ArXiv* abs/2112.03907 (2021).

Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. 2021. NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction. *Proc. NeurIPS* (2021).

Yiheng Xie, Towaki Takikawa, Shunsuke Saito, Or Litany, Shiqin Yan, Numair Khan, Federico Tombari, James Tompkin, Vincent Sitzmann, and Srinath Sridhar. 2022. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, Vol. 41. Wiley Online Library, 641–676.

Guandao Yang, Serge Belongie, Bharath Hariharan, and Vladlen Koltun. 2021a. Geometry Processing with Neural Fields. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Guandao Yang, Sagie Benaim, Varun Jampani, Kyle Genova, Jonathan Barron, Thomas Funkhouser, Bharath Hariharan, and Serge Belongie. 2022. Polynomial Neural Fields for Subband Decomposition and Manipulation. In *Thirty-Sixth Conference on Neural Information Processing Systems*.

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. 2019. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4541–4550.

Yan Yang, Angela F Gao, Jorge C Castellanos, Zachary E Ross, Kamyar Azizzadenesheli, and Robert W Clayton. 2021b. Seismic wave propagation and inversion with neural operators. *The Seismic Record* 1, 3 (2021), 126–134.

Ekrem Fatih Yılmazer, Delio Vicini, and Wenzel Jakob. 2022. Solving Inverse PDE Problems using Grid-Free Monte Carlo Estimators. (Aug. 2022). arXiv:2208.02114 [cs.GR]

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.

Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. 2020. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492* (2020).