Computers & Graphics (2023)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Neural Green's Function for Laplacian Systems

Jingwei Tang[a], Vinicius C. Azevedo[b], Guillaume Cordonnier[c], Barbara Solenthaler[a]

[a] *ETH Zürich, Switzerland*
[b] *DisneyResearch—Studios, Switzerland*
[c] *Inria Université Côte d'Azur, France*

ARTICLE INFO

Article history:
Received March 18, 2022

Keywords: Machine Learning, Modeling and Simulation, Poisson Equation, Green's Function

ABSTRACT

Solving linear system of equations stemming from Laplacian operators is at the heart of a wide range of applications. Due to the sparsity of the linear systems, iterative solvers such as Conjugate Gradient and Multigrid are usually employed when the solution has a large number of degrees of freedom. These iterative solvers can be seen as sparse approximations of the Green's function for the Laplacian operator. In this paper we propose a machine learning approach that regresses a Green's function from boundary conditions. This is enabled by a Green's function that can be effectively represented in a multi-scale fashion, drastically reducing the cost associated with a dense matrix representation. Additionally, since the Green's function is solely dependent on boundary conditions, training the proposed neural network does not require sampling the right-hand side of the linear system. We show results that our method outperforms state of the art Conjugate Gradient and Multigrid methods.

1. Introduction

Efficiently solving linear systems originating from discrete Partial Differential Equations (PDEs) is central to many modern applications, ranging from modelling of natural phenomena to image processing. Since these equations model local relationships, their discretized counterparts are sparse and only few entries are non-zero. Therefore, direct matrix inversion is not efficient since it yields a dense representation, and iterative solvers such as Conjugate Gradient and Multigrid are preferred.

In this work, we propose a novel framework that comprehends iterative methods for solving linear systems of equations stemming from Poisson Equations. Our method is inspired by the theory of Green's functions: integral equations obtained from the PDE and its corresponding boundary conditions. Once computed, they can provide the solution to the PDE by a simple convolution. However, Green's function methods are not adopted in practical applications because analytic solutions exist only in simple settings, and discretizing them is unpractical due to their wide kernel support. Our first contribution, thus, is a novel multi-level discrete Green's function formulation that is able to take advantage of a sparse and more efficient design. Our second contribution is to take advantage that Green's functions only depend on the boundaries of the domain, and train a neural network model to regress Green's functions from general boundary settings. Lastly, due to the spectral properties of our multi-level discrete Green's function, our method can be applied as an iterative solver on the error residual. These combined contributions create a linear system solver with unprecedented error convergence in 2-D, surpassing state of the art Conjugate Gradient and Multigrid methods.

2. Related Works

We revisit methods for solving linear system of equations arising from linear discrete PDEs in the following. Classical methods are extensively studied by LeVeque [1], so our discussion will focus on recent attempts involving learning-based methods and approximating Green's functions.

Learning Green's functions. The work of Alkhalifah et al. [2] proposed to represent Green's function of wave equations

through fully-connected neural networks. The network takes spatial coordinates and source locations as input and predicts the Green's function value for a given domain. The formulation is similar to the Physics-Informed Neural Networks (PINN) [3], where each specific PDE is represented by one fully connected network. Similarly to our work, Ichimura et al. [4] developed a neural network trained to fit local patches of the discretized Green's function. The training depends on the right-hand side and solution pair, thus results in a very large dataset, containing around 16.2 million samples. As the proposed Green's function is still an approximation of the ground-truth, the authors adopted it as a pre-conditioner for the linear system. On the contrary, we adopt the approximated Green's function as a multi-level iterative solver.

Green's functions were also explored in a multi-scale fashion to solve specific electromagnetic potentials. In a series of papers, Li et al. [5, 6, 7] proposed neural networks as non-linear neural operators to map coefficients of differential operators directly to solutions. However, the proposed neural operators always assume fixed forcing terms in the PDE, and these methods do not deal with complex boundary conditions. The work of Feliu-Faba et al. [8] decomposes Green's function of pseudo-differential operators through a nonstandard wavelet transform, training neural networks that jointly learn the mapping from Laplacian coefficients and wavelet parameters. This approach, however, does not take complex boundary conditions into consideration. Gin et al. [9] designs an autoencoder structure to map forcing terms from non-linear PDEs into a linear space for solving PDEs through Green's functions. Recent advances in learning Green's functions also involve applying them other domains such as solving Helmholtz and Sturm-Liouville problems [9], computing discrete Markov Chains [10], and quantum field theory [11].

Multi-Resolution Analysis and Sparse Approximate Inverses. Our work is inspired by several recent contributions on multi-resolution analysis and Sparse Approximate Inverses. Multiresolution matrix factorization (MMF) [12, 13, 14] extends classic multi-resolution analysis [15] to matrix representations. Haar wavelets (which act as ideal low pass filters) are the natural basis when constructing hierarchical representations of the Laplacian operator [12]. Similarly to our work, progressive mollification is applied to Kroenecker deltas for multi-scale analysis with diffusion wavelets [16]. Sparse Approximate Inverses (SPAI), on the other hand, are more general low rank matrix approximations for the inverse of a discrete matrix. These can be employed as preconditioners [17, 18, 19, 20] and as Multigrid smoothers [21, 22].

Solving Linear PDEs with Convolutional Neural Networks. More recent endeavors in the deep learning era leaned towards a direct mapping between image-represented boundary conditions of 2D Laplace equations and their solutions through convolutional neural networks (CNN). Barati Farimani et al. [23] trained a U-Net model by combining L^1 and adversarial losses, while Sharma et al. [24] adopted a weakly-supervised residual loss. Other works solved the Poisson equation for applied problems, including pressure projection in fluid simulation [25, 26], electric potentials [27] and particles simulation [28]. Differently from the previous work, Hsieh et al. [29] proposed to use neural networks to modify Jacobi-style iterative solvers. The neural network operates on the error term at each iteration, and is designed to be linear (without bias and non-linear activation functions) to guarantee convergence to the correct fix point solution. After supervised training, the network can moderately improve the convergence speed of a Jacobi and a Multigrid solver.

3. Background

We briefly review the linear partial differential equations (PDE) and their Green's function solutions in this section. The symbols used throughout the paper can be found in Table 1.

3.1. Linear Partial Differential Equations

Linear PDE solvers aim to find functions that satisfy a set of linear differential equations. Consider $\mathcal{F} = \{u : \Omega \in \mathbb{R}^k \rightarrow \mathbb{R}\}$ as a space of smooth scalar field functions in a domain of k dimensions, $\mathcal{A} : \mathcal{F} \rightarrow \mathcal{F}$ a linear differential operator and $u \in \mathcal{F}$ a candidate function that satisfies the equation $\mathcal{A}u(\mathbf{x}) = f(\mathbf{x})$, for $f \in \mathcal{F}$. In this paper, we assume $\mathcal{A} = \nabla^2 = \frac{\partial^2}{\partial x_1^2} + \cdots + \frac{\partial^2}{\partial x_n^2}$, which yields a Poisson equation of the form $\nabla^2 u(\mathbf{x}) = f(\mathbf{x})$. Solving a linear PDE involves finding a function $u \in \mathcal{F}$ that satisfies the above conditions.

The solution of a linear PDE depends on specified boundary conditions (BC). Assuming $\partial\Omega$ as the boundary of the domain with an oriented normal vector $\mathbf{n}$, homogeneous Dirichlet ($u(\mathbf{x}) = 0, \mathbf{x} \in \partial\Omega^{\mathcal{D}}$) and Neumann ($\frac{\partial u(\mathbf{x})}{\partial \mathbf{n}} = 0, \mathbf{x} \in \partial\Omega^{\mathcal{N}}$) are common boundary conditions. These conditions can model obstacles and domain boundaries. Thus, a Poisson equation with prescribed boundary conditions is formulated as

$$\begin{cases} \nabla^2 u(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in \Omega, \\ u(\mathbf{x}) = 0, & \mathbf{x} \in \partial\Omega^{\mathcal{D}}, \\ \frac{\partial u(\mathbf{x})}{\partial \mathbf{n}} = 0, & \mathbf{x} \in \partial\Omega^{\mathcal{N}}. \end{cases} \tag{1}$$

3.2. Green's Function

A *Green's function* $\mathcal{G}(\mathbf{x}, \mathbf{x}')$ of the linear differential operator $\mathcal{A}$ is defined as

$$\begin{cases} \mathcal{A}\mathcal{G}(\mathbf{x}, \mathbf{x}') = \delta(\mathbf{x} - \mathbf{x}'), & \mathbf{x} \in \Omega, \\ \frac{\partial \mathcal{G}(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{x}} = 0, & \mathbf{x} \in \partial\Omega^{\mathcal{N}}, \\ \mathcal{G}(\mathbf{x}, \mathbf{x}') = 0, & \mathbf{x} \in \partial\Omega^{\mathcal{D}}, \end{cases} \tag{2}$$

where δ is the Dirac delta function: $\delta(\mathbf{x}) = 0$ for $\mathbf{x} \neq 0$, $\int_{-\infty}^{\infty} \delta(\mathbf{x}) d\mathbf{x} = 1$; and $\mathbf{x}' \in \Omega$ is a fixed point. The Green's function is useful to solve inhomogeneous linear PDEs, since once computed for a specific operator, the PDE $\mathcal{A}u(\mathbf{x}) = f(\mathbf{x})$ is immediately solved by

$$u(\mathbf{x}) = \int_{\Omega} \mathcal{G}(\mathbf{x}, \mathbf{x}')\ f(\mathbf{x}')\ d\mathbf{x}', \tag{3}$$

for any arbitrary forcing function $f(\mathbf{x})$.

Several analytical solutions of Green's functions exist; for example, the three dimensional Green's function for the

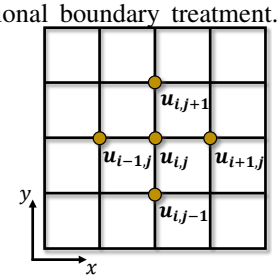
Laplace equation with Dirichlet conditions at the infinity is given by

$$\mathcal{G}_\infty(\mathbf{x}, \mathbf{x}') = -\frac{1}{4\pi} \cdot \frac{1}{|\mathbf{x} - \mathbf{x}'|}. \quad (4)$$

However, once more complex boundary conditions are introduced, there are no known closed form Green's function formulation for the general case. The goal of this paper, thus, is to efficiently approximate $\mathcal{G}(\mathbf{x}, \mathbf{x}')$ given a certain boundary configuration. This will provide us a solution operator of any Poisson equation without relying on the forcing function $f(\mathbf{x})$.

3.3. The Discretized Setting

Since there are no trivial analytical solutions for linear PDEs in the general case, its common to solve them numerically. To do so, the first step is to discretize the operator $\mathcal{A}$ spatially. Among grid-based methods, regular (Cartesian), curvilinear, or unstructured discretizations are common [1], and each of which can be potentially coupled with distinct approximation schemes such as finite differences, finite volumes or finite elements. In this paper we will focus on approximating Green's functions for regular grids in 2-D with embedded objects discretized by fully filled cells [30], and no fractional boundary treatment. We notice, however, that all the aforementioned discretizations will yield a discrete linear matrix and the discussion presented here does not limit itself to the particular choice of regular grids coupled with finite differences.



The 2-D Laplacian operator discretized with a second order approximation at regular grid node $\mathbf{x}_{i,j}$ is given by

$$\nabla^2 u(\mathbf{x}_{i,j}) \approx \frac{1}{h^2}(\mathbf{u}_{i+1,j} + \mathbf{u}_{i,j+1} + \mathbf{u}_{i-1,j} + \mathbf{u}_{i,j-1} - 4\mathbf{u}_{i,j}), \quad (5)$$

where h is the grid spacing and $\mathbf{u}_{i+1,j}, \mathbf{u}_{i,j+1}, \mathbf{u}_{i-1,j}, \mathbf{u}_{i,j-1}, \mathbf{u}_{i,j}$ are discrete grid values at different locations (inset). Notice that this discretization of Laplacian operator is compact, since each node only interacts with its immediate neighbors. Extending this relationship for all grid nodes yields the following linear system:

$$\mathbf{Au} = \mathbf{f}. \quad (6)$$

Here we use $\mathbf{A}$ to discretize the negative of Laplacian operator. This system is sparse with 5 non-zero entries per row/column. Boundary conditions can change the stencil of the discrete Laplacian kernel (Equation (5)) and the structure of matrix $\mathbf{A}$. Moreover, if boundary conditions are compatible, the matrix $\mathbf{A}$ is symmetric positive definite.

It's straightforward to see that in the discrete case, the Green's function is simply the inverse of the Laplacian matrix, since discretizing Equation (2) yields

$$\mathbf{GA} = \mathbf{I}, \quad (7)$$

where $\mathbf{G}$ is the discretized Green function for the operator $\mathbf{A}$. Therefore, for an arbitrary discrete forcing term $\mathbf{f}$, the solution of the PDE is

$$\mathbf{u} = \mathbf{Gf}. \quad (8)$$

Table 1. Summary of symbols

$\mathcal{A}, \mathcal{G}$	Differential operator and its Green's function
$u(\mathbf{x}), f(\mathbf{x})$	Solution and forcing function of the PDE
$\mathbf{A}, \mathbf{G}$	Discretized Laplacian and its Green's function
$\mathbf{u}, \mathbf{f}$	Ground-truth solution and right-hand side vectors
$\hat{\mathbf{u}}$	Approximated solution vector
$\mathbf{r}$	Residual vector $\mathbf{r} = \mathbf{f} - \mathbf{A}\hat{\mathbf{u}}$
$\partial\Omega^{\mathcal{N}}, \partial\Omega^{\mathcal{D}}$	Neumann and Dirichlet boundary
ℓ	Level index for Multi-level Green's Function
i, j	Grid position index
N_ℓ	Number of nodes of the regular grid at level ℓ
m_ℓ, n_ℓ	Sizes (x and y direction) of the regular grid at level ℓ
$\mathbf{U}_\ell^q, \mathbf{D}_\ell^q$	Upsampling and Downsampling operators
$\mathbf{G}_\ell^*$	Residual Green's function at level ℓ (Eq. 9)
$\mathbf{G}_\ell$	Green's function at level ℓ (Eq. 13)
$\mathbf{I}_\ell$	Downsampled identity matrix at level ℓ
k_ℓ	Size of Green's Function Kernel at level ℓ
$\mathbf{u}_\ell$	Intermediate solution vector at level ℓ
$\mathbf{f}_\ell$	Downsampled right-hand side vector at level ℓ
ϕ_ℓ	SDF of solid obstacles at level ℓ
$\mathcal{M}_\ell, \Theta$	MLP at level ℓ and its parameters

4. Neural Green's Function For Laplacian Systems

The most straightforward way to find a Green's function solution for a Poisson Equation is to invert its discrete matrix representation $\mathbf{G} = \mathbf{A}^{-1}$. However, this is computationally inefficient as $\mathbf{G}$ is dense. Even if the Green's function for discrete Laplacian is known a-priori, the cost for computing the solution by dense matrix multiplication $\mathbf{u} = \mathbf{Gf}$ is $O(N^2)$ given a regular grid with N nodes, which is sub-par when compared with state-of-the-art iterative solvers. This approach, therefore, is not employed in practice.

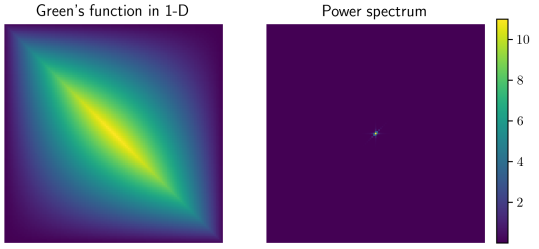


Fig. 1. Left: a Green's function for the one dimensional Laplacian operator. Right: the power spectrum of the Fourier decomposition of the Green's function.

The dense Green's function for the Laplacian operator, however, has a sparse counterpart in the frequency space: consider its one-dimensional representation plotted for Dirichlet boundary conditions in Figure 1. Despite being dense in the original space, its power spectrum shows that it is sparse on the frequency space. This property suggests that there is another representation, besides the standard matrix format, in which the Green's function can be compactly discretized. Previous works on Sparse Approximate Inverses (SPAI) employed Wavelets [20] to also make the $\mathbf{G}$'s representation more efficient. We

therefore propose a multi-level compact representation for the Green's function in the next section.

4.1. A Multi-level Green's Function Representation

Our multi-level Green's approximation (MLGA) relies on lower resolution grids that are progressively up-sampled until a target resolution:

$$\begin{aligned}\mathbf{G} &= \mathbf{U}_1^L\mathbf{G}_1^*\mathbf{D}_L^1 + \mathbf{U}_2^L\mathbf{G}_2^*\mathbf{D}_L^2 + \ \dots\ + \mathbf{U}_{L-1}^L\mathbf{G}_{L-1}^*\mathbf{D}_L^{L-1} + \mathbf{G}_L^* \\ &= \sum_{\ell=1}^{L-1}\mathbf{U}_\ell^L\mathbf{G}_\ell^*\mathbf{D}_L^\ell + \mathbf{G}_L^*, \end{aligned} \tag{9}$$

where $\mathbf{G}_1^*$ and $\mathbf{G}_L^*$ are matrices representing coarsest and finest discretizations respectively; and $\mathbf{U}_\ell^q$ ($\mathbf{D}_\ell^q$) is upsampling (downsampling) operator that maps a vector from discretization level ℓ (of N_ℓ nodes) to discretization level q. We implement the upsampling (downsampling) operators through composing a series of ratio-2 operators; e.g., the upsampling operator $\mathbf{U}_\ell^L$ is composed as

$$\mathbf{U}_\ell^L = \mathbf{U}_{L-1}^L\mathbf{U}_{L-2}^{L-1}\cdots\mathbf{U}_\ell^{\ell+1} = \prod_{q=\ell}^{L-1}\mathbf{U}_q^{q+1}. \tag{10}$$

Similar to the Multigrid method [31], the ratio-2 upsampling and downsampling operators are represented by 3×3 full-weighting kernels. The downsample kernels are constructed by computing the tensor product $\mathbf{D} = \mathbf{B}\otimes\mathbf{B}$ between one-dimensional stencils. We choose $\mathbf{B}$ to be a linear stencil defined as

$$(\mathbf{B}\mathbf{u})_i = \frac{1}{4}\mathbf{u}_{i-1} + \frac{1}{2}\mathbf{u}_i + \frac{1}{4}\mathbf{u}_{i+1}. \tag{11}$$

The upsample operator is chosen to be the scaled transpose of the downsample operator $\mathbf{U} = 4\mathbf{D}^{\mathrm{T}}$ [31]. Boundary conditions are easily handled by manipulating the operand: a Dirichlet condition imposes that values on the boundaries are set to 0, while a Neumann condition requires extrapolation of values to the boundary in the direction of the derivative. When a boundary cell has multiple Neumann conditions applied to it, the extrapolation simply averages contributions from different directions. Moreover, we notice that linear 3×3 downsampling kernels can introduce aliasing; however this was not an observed problem in our experiments.

4.2. Enforcing Sparsity

Simply approximating the matrix inverse with the formulation shown in Equation (9) is not enough for an efficient representation. Therefore, similarly to SPAI approaches [14], our goal is to find a sparse approximation of the Green's function $\hat{\mathbf{G}}$, which can be expressed through the following optimization:

$$\hat{\mathbf{G}} = \arg\min_{\mathbf{G}} \|\mathbf{G}\mathbf{A} - \mathbf{I}\|_2^2, \tag{12}$$

where $\hat{\mathbf{G}}$ is sparse. One of the contributions of this paper is to show that the multi-level approximation can sparsely and efficiently represent the Green's Function by solving Equation (12) independently per level. To show that, we first define $\mathbf{G}_\ell$ as

$$\mathbf{G}_\ell = \sum_{q=1}^{\ell-1}\mathbf{U}_q^\ell\mathbf{G}_q^*\mathbf{D}_\ell^q + \mathbf{G}_\ell^* \ = \ \mathbf{U}_{\ell-1}^\ell\mathbf{G}_{\ell-1}\mathbf{D}_\ell^{\ell-1} + \mathbf{G}_\ell^*. \tag{13}$$

The equation above states that the approximation at the ℓ-th level is defined by the summing the previous upsampled level $\mathbf{G}_{\ell-1}$ and a residual matrix $\mathbf{G}_\ell^*$ defined at ℓ.

Similarly to the Galerkin approximation, our derivation assumes that coarser levels should solve a Laplace system progressively downsampled from the finest level. Defining $\mathbf{A}_\ell = \mathbf{D}_L^\ell\mathbf{A}\mathbf{U}_\ell^L$ and using the Green's definition of Equation (13), the sparse optimization can be written as

$$\hat{\mathbf{G}}_\ell = \arg\min_{\mathbf{G}_\ell} \|\mathbf{G}_\ell\mathbf{A}_\ell - \mathbf{I}_\ell\|_2^2, \tag{14}$$

where $\mathbf{I}_\ell = \mathbf{D}_L^\ell\mathbf{I}\mathbf{U}_\ell^L$ is the downsampled identity matrix.

Directly optimizing Equation (14) is memory inefficient, since the approximation in a level depends on the previous coarser one. By reformulating Equation (14), we can solve for the mismatch of the downsampled identity between grid resolutions of adjacent levels $\mathbf{I}_\ell - \mathbf{U}_{\ell-1}^\ell\mathbf{D}_\ell^{\ell-1}\mathbf{I}_\ell$, and rewrite the optimization to be level-independent as

$$\hat{\mathbf{G}}_\ell^* = \arg\min_{\hat{\mathbf{G}}_\ell^*} \left\|\hat{\mathbf{G}}_\ell^*\mathbf{A}_\ell - \left(\mathbf{I}_\ell - \mathbf{U}_{\ell-1}^\ell\mathbf{D}_\ell^{\ell-1}\mathbf{I}_\ell\right)\right\|_2^2. \tag{15}$$

Notice that this method only works because it is bound to a target finest grid resolution; therefore, it cannot be reused for distinct grids with varying number of nodes/levels. The full derivation from Equation (14) to Equation (15) is presented in the supplemental material.

Lastly, the per-level $\hat{\mathbf{G}}_\ell^*$ Green's approximation is mostly sparse. That happens because each row (a 2-D kernel converted to an array) has a *compact* support due to the predominance of lower frequencies in Green's function of elliptical operators. The kernel with compact support is centered around the point of evaluation as illustrated in Figure 2. This assumption requires that contributions from distant nodes are inherently modelled by interpolation of Green's Functions from coarser levels. Scenarios that violate this assumption will be discussed in Section 6.

We choose to represent residual Green's function approximations $\hat{\mathbf{G}}_\ell^*$ through spatially varying convolutions, i.e. sliding-window of compact kernels $\mathbf{G}_\ell^*(i_\ell, j_\ell)$ that vary at each position. Here $1 \le i_\ell \le m_\ell$, $1 \le i_\ell \le n_\ell$, with m_ℓ and n_ℓ being the sizes in x and y direction of the regular grid at level ℓ, and $N_\ell = m_\ell n_\ell$. We can conveniently write them as sparse matrix-vector multiplications: the values of $\mathbf{G}_\ell^*(i_\ell, j_\ell)$ represent the non-zero values at row $(i_\ell N_\ell + j_\ell)$ in the matrix $\mathbf{G}_\ell^* \in \mathbb{R}^{N_\ell\times N_\ell}$. For coarser levels we adopt kernels of sizes $k_\ell \times k_\ell$ to cover most of the domain since they are still relatively cheap to compute. As the discretization progresses to more refined levels, the kernel size progressively decreases until it covers only a small neighbourhood (e.g., for all our examples the finest level has kernel size 5×5).

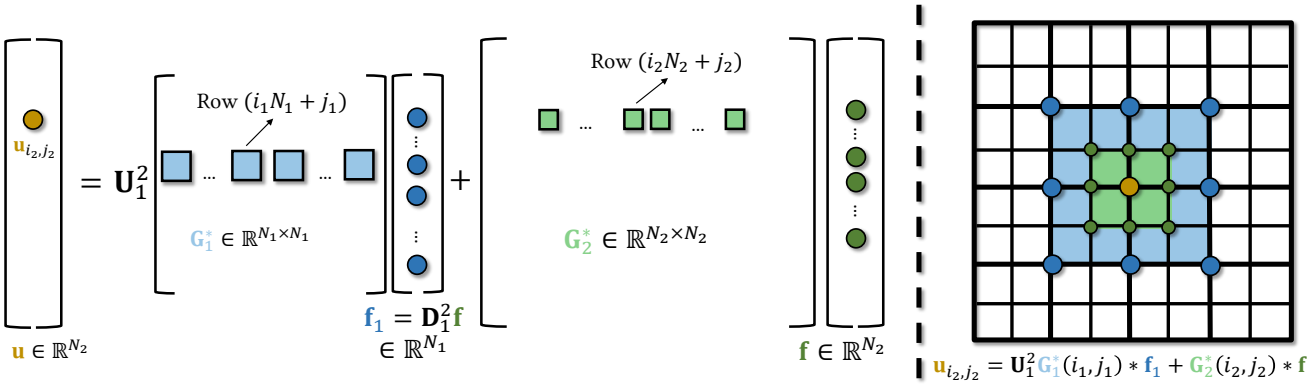


Fig. 2. Multi-level Green's function ($L = 2$). Left: sparse matrix represented by multi-level Green's function $\mathbf{G}_1^*, \mathbf{G}_2^*$ multiplied with vector $\mathbf{f}$. Right: spatially varying convolutional kernel represented by $\mathbf{G}_1^*, \mathbf{G}_2^*$ convolved with $\mathbf{f}$. Here $*$ represents the 2-D convolution operation. Both representations are equivalent to each other, with the same values shown in the same color.

The number of operations needed to for sparse matrix-vector multiplication depends on the number of non-zero matrix entries; the total number of non-zero elements in $\mathbf{G}_\ell^*$ is $\sum_{\ell=1}^{L} k_\ell^2 N_\ell$. The number of operations needed for upsampling and downsampling operators implemented by 3×3 kernels is $\sum_{\ell=1}^{L-1} 3^2 N_\ell$. Therefore, the total number of operations for multiplying the multi-level Green's function with a vector is $k_L^2 N_L + \sum_{\ell=1}^{L-1} \left(k_\ell^2 N_\ell + 2 \cdot 3^2 N_\ell\right)$. Thus, as long as $k_l^2 << N_l$ in higher resolution levels, evaluating Equation (8) with the proposed multi-level Green's approximation requires significantly less operations when compared with it the dense Green's counterpart, which requires $(N_L)^2$ operations.

4.3. Solving the Poisson System Iteratively

Once the optimization problem in Equation (15) is solved, the final approximation of the Green's function can be obtained by combining all $\hat{\mathbf{G}}_\ell^*$ in Equation (9). However, naively computing this matrix-vector multiplication is memory inefficient, since there is no need to upsample the per-level residual Green's function $\hat{\mathbf{G}}_\ell^*$ to level L. Instead, Equation (8) can be implemented through applying $\hat{\mathbf{G}}_\ell^*$ on the right-hand side vector $\mathbf{f}$ in a level-by-level fashion. Starting with level $\ell = 1$, a first approximation of the solution is obtained $\mathbf{u}_0 = \hat{\mathbf{G}}_1^*(\mathbf{D}_L^1 \mathbf{f})$. At the following levels ℓ, we apply $\hat{\mathbf{G}}_\ell^*$ on the corresponding right-hand side vector to get a error correction term $\mathbf{e}^\ell = \hat{\mathbf{G}}_\ell^*(\mathbf{D}_L^\ell \mathbf{f})$. This correction is then added to the upsampled coarser level, which yields the current level's solution $\mathbf{u}_\ell = \mathbf{U}_{\ell-1}^\ell \mathbf{u}_{\ell-1} + \mathbf{e}_\ell$. The process continues until the finest level L.

As the approximated Green's function $\hat{\mathbf{G}}$ is not exact due to cut-offs introduced by compact kernels, directly applying it on the right-hand side vector usually does not provide a solution that is precise enough. Thus, we devise an iterative fashion of applying the Green's function to refine the prediction. After getting a first approximation of the result as $\hat{\mathbf{u}} = \mathbf{u}_L$, a residual $\mathbf{r} = \mathbf{f} - \mathbf{A}\hat{\mathbf{u}}$ is computed. Then, the approximated Green is applied on this residual and added back to $\hat{\mathbf{u}}$ to obtain the next approximation: $\hat{\mathbf{u}} \leftarrow \hat{\mathbf{u}} + \hat{\mathbf{G}}\mathbf{r}$. The process is repeated until a user-defined residual tolerance or maximum number of iteration is reached. The presented method has similarities with a Multigrid solver, but the relaxation step is substituted by using our approximated Green's kernels. The whole solving process is summarized in Algorithm 1.

Input: $\{\hat{\mathbf{G}}_\ell^* : \ell = 1, 2, \ldots, L\}$, right-hand side vector $\mathbf{f}$, system matrix $\mathbf{A}$, maximum number of iteration N_{itr}, residual tolerance ε.
Initialization: $\mathbf{r} \leftarrow \mathbf{f}$; $\hat{\mathbf{u}} \leftarrow 0$;
for $I = 1$ **to** N_{itr} **do**
  $\mathbf{u}_1 \leftarrow \hat{\mathbf{G}}_1^*(\mathbf{D}_L^1 \mathbf{r})$;
  for $\ell = 2$ **to** L **do**
    $\mathbf{e}_\ell \leftarrow \hat{\mathbf{G}}_\ell^*(\mathbf{D}_L^\ell \mathbf{r})$;
    $\mathbf{u}_\ell \leftarrow \mathbf{U}_{\ell-1}^\ell \mathbf{u}_{\ell-1} + \mathbf{e}_\ell$;
  end
  $\hat{\mathbf{u}} \leftarrow \hat{\mathbf{u}} + \mathbf{u}_\ell$;
  $\mathbf{r} \leftarrow \mathbf{f} - \mathbf{A}\hat{\mathbf{u}}$;
  if $\|\mathbf{r}\|_\infty < \varepsilon$ **then**
    break;
  end
end
Output: $\hat{\mathbf{u}}$

Algorithm 1: Using $\mathbf{G}_\ell^*$ to solve Poisson Equation.

4.4. Learning Green's Functions

The optimization in Equation (15) depends on the specific system matrix $\mathbf{A}$ that it was solved for, which restricts its application to a Laplace operator under specific boundary conditions. Assuming that the Laplacian operator has homogeneous coefficients, we observe that the kernel weights of $\mathbf{G}_\ell^*(i, j)$ can be determined solely by the boundary conditions in a given domain. Therefore, we can obtain Green's functions for more general configurations by training a Multilayer Perceptron (MLP) to directly map boundary conditions into kernel weights.

Contrary to position-based features that were used in previous implicit approaches [32], we can adopt features that are based on distances of an evaluation point relative to all domain boundaries. Including all these distances (or, alternatively, the whole domain as input) as features of the neural network, however, would result in a large parametrization space that is challenging to generalize, since it would contain features that would

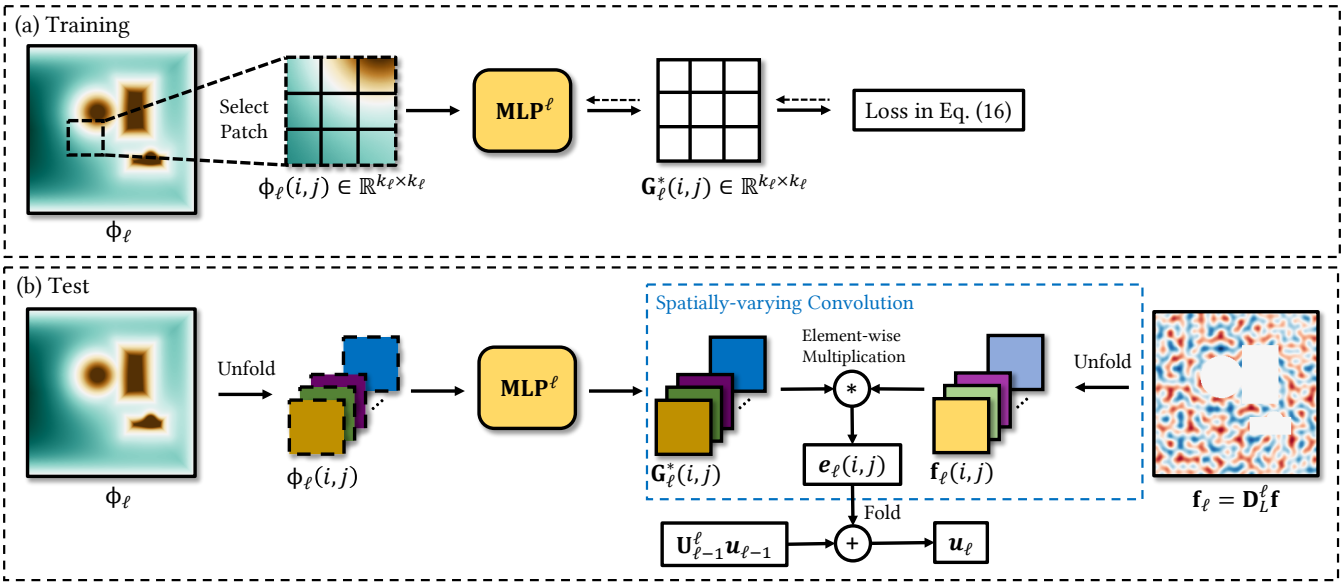


Fig. 3. Pipeline illustration for training and test time. During training (a), given the SDF ϕ_ℓ of objects representing the interior boundary setting at level ℓ, several patches are randomly selected to form an input mini-batch. The MLP at this level processes the input SDF batch and outputs a batch of corresponding residual Green's function kernels. These kernels are then used to incur the loss defined in Equation (16). Dashed lines represent the back-propagation process. At test time (b), sliding local patches are extracted from the SDF ϕ_ℓ. These patches represent the input MLP; the MLP's output corresponds to the residual Green's function kernels. The kernels are used to perform a spatially-varying convolution on the right-hand-side to get the error correction term $\mathbf{e}_\ell$. The error correction term is added to the up-sampled coarser solution from level $\ell - 1$ to get the current level's solution $\mathbf{u}_\ell$. The *Unfold* operation extracts all local patches centered at (i, j) of size $k_\ell \times k_\ell$: $\phi_\ell \to \{\phi_\ell(i,j)\}^{(m_\ell, n_\ell)}_{(i,j)=(0,0)}$. The *Fold* operation is the inverse of *Unfold*, and it combines all local patches back into a complete field.

only weakly correlate with the outputs of our Neural Green's function. Instead, we rely on a feature vector that takes patches of Signed Distance Functions (SDFs) from the point of evaluation to the domain boundaries. These functions are smooth, and once coupled with their derivatives - which are automatically encoded by the patch information - they can represent non-local information of the underlying geometries embedded on the domain.

The neural network that maps the boundary conditions to Green's function kernels at level ℓ is defined as $\mathcal{M}^\Theta_\ell : \phi_\ell(i,j) \in \mathbb{R}^{k_\ell \times k_\ell} \to \mathbf{G}^*_\ell(i,j) \in \mathbb{R}^{k_\ell \times k_\ell}$. This mapping takes a SDF patch ϕ_ℓ centered at (i, j) and maps it to a Green's kernel $\mathbf{G}^*_\ell(i,j)$ of the same size. Since the mapping is defined continuously by the signed distance functions, the mapping function $\mathcal{M}^\Theta_\ell$ is well represented by a MLP. The MLP flattens the input SDF patch into a vector and outputs a vector of the same shape. The output vector is then reshaped into a $k_\ell \times k_\ell$ patch and used as $\mathbf{G}^*_\ell(i,j)$.

We define separate MLPs for each level. The training of each level is also performed independently according to the objective defined in Equation (15). The only difference lies in the parameters to be optimized:

$$\hat{\Theta} = \arg\min_\Theta \sum_r \sum_{i,j} \left\| \mathcal{M}^\Theta_\ell \left(\phi^r_\ell(i,j)\right) \mathbf{A}_\ell - \left(\mathbf{I}_\ell - \mathbf{U}^\ell_{\ell-1} \mathbf{D}^{\ell-1}_\ell \mathbf{I}_\ell\right) \right\|^2_2, \quad (16)$$

where r denotes the training example index. The multiplication of $\mathcal{M}^\Theta_\ell(\phi_\ell(i,j))$ and matrix $\mathbf{A}$ is a sparse vector-matrix multiplication. In practice, we do not sequentially take all patches from a boundary setting to form a batch during training, but rather randomly pick (i, j) from different boundary settings.

At test time, we extract all sliding local patches from ϕ_ℓ to form a batch $\{\phi_\ell(i,j) : \forall 1 \le i \le m_\ell, \forall 1 \le j \le n_\ell\}$. The whole batch is used as input to the current level's MLP to get predictions of the corresponding residual Green's function kernels $\{\mathbf{G}^*_\ell(i,j) : \forall 1 \le i \le m_\ell, \forall 1 \le j \le n_\ell\}$. This process is performed for all levels $\ell = 1, \ldots, L$ to obtain the full multi-level Green's function representation. Given $\mathbf{G}^*_\ell(i,j)$ for all levels, we can use the procedure defined in Algorithm 1 to solve for arbitrary right-hand side. An overall illustration of the training and test pipeline can be seen in Figure 3.

5. Experiments and Results

We evaluate the ability of our approximated Green's function to solve the Poisson equations (Algorithm 1) by comparing its performance with classical linear solvers. Random right-hand side vectors $\mathbf{f}$ are generated with either Gaussian noise (Figure 5, top row) or Perlin noise [33] (Figure 5, bottom row). Classical linear solvers evaluated include Jacobi, Conjugate Gradient (CG), Multigrid (MG), and Multigrid Preconditioned Conjugate Gradient (MGPCG). The residual L^1-norm of different solvers is plotted against the number of multiply-add operations, as a comparison based on the number of iterations would be biased by the computational cost required at each iteration step.

5.1. Implementation Details

We implement all our models, as well as classical (Conjugate Gradient (CG), Jacobi, Multigrid, and Multigrid Preconditioned Conjugate Gradient) solvers in PyTorch [34]. All experiments are run on a NVIDIA GeForce RTX 2080Ti GPU with 11 GB of dedicated memory.

Multi-level representation. Grids at all levels are discretized with a resolution of $N_l = (2^{\ell+1} + 1) \times (2^{\ell+1} + 1)$ nodes, where $\ell = 1, \ldots, L$ is the index of the level. This means that we always fix the coarsest level $\ell = 1$ to have 5×5 nodes. The kernel sizes of each level k_ℓ are determined heuristically. For example, setups with the resolution of $N = 129^2$ employ different kernels sizes for each level as $k_1 = 9, k_2 = 11, k_3 = 9, k_4 = 7, k_5 = 5, k_6 = 5$. The total number of multiply-add operations in this setting when applying the multi-level Green's function to the right-hand side is $\sim 7.1 \times 10^5$, while the operations needed for applying the dense Green's function is $\sim 2.8 \times 10^8$. Equation (15) is solved as an unconstrained optimization problem with a Quasi-Newton (L-BFGS) optimizer. We choose a history size of 10 to approximate the Hessian, and the Armijo criteria is used for the line search algorithm, which is limited to 10 steps. For all tests the same hyper-parameters are used for the L-BFGS algorithm.

Neural networks. The model in Section 4.4 uses distinct MLPs for each level. The MLPs of the coarser levels (up to $\ell = 3$) have 6 fully-connected layers, while rest of the other levels have 12 layers. Each hidden layer in the MLP has 256 channels, and uses ReLU activations. The input layer is skip connected to the 4th layer. Dropout is applied in all layers during training. Each MLP is trained with an Adam Optimizer, with initial learning rate set to 10^{-4}. The learning rate progressively decays every 300 epochs by a ratio of 0.5 until convergence. MLPs of different levels are trained independently in parallel. The training takes around 24 hours for each level.

Multigrid Settings. We implement a geometric Multigrid solver [31] with weighted Jacobi ($\omega = \frac{2}{3}$) as the smoothing operator. We use 8 pre-smoothing steps, 16 post-smoothing steps for all levels, and 20 smoothing steps for the coarsest level. When used as preconditioner for Conjugate Gradient, we change the parameters to 2 pre-smoothing steps, 2 post-smoothing steps and 4 smoothing steps for the coarsest level. The same set of parameters is used for all experiments.

5.2. Representing Green's Function for a Single Scene

We evaluate our method in two parts: first, we validate that our multi-level representation can adequately represent a Green's function for a fixed boundary configuration. Next, we measure the capacity of the MLPs to predict Green's functions for arbitrary obstacles. The purpose of our first experiment is to validate that the choice of our structure of multi-level convolution is sufficient to accurately and sparsely approximate the discrete Green's function (the dense inverse of the $\mathbf{A}$ matrix). To this end, we fix the boundary conditions and directly optimize for the values of $\hat{\mathbf{G}}^*_\ell$ for all ℓ (Equation (15)). We note that the discretization of the Green's function can be exactly represented with our multi-layer model if we allow the radius of the convolution to span the size of the whole domain. Although this would be unpractical for large domains, this allows us to compute a set of ground truth kernels $\mathbf{G}^*_\ell$ at low resolution (33×33).

Figure 4 shows the comparisons of ground-truth $\hat{\mathbf{G}}^*_\ell(i, j)$ (b) and approximated $\mathbf{G}^*_\ell(i, j)$ (c) kernels at different locations for level $\ell = 3$, with the difference between them shown on the right-most column. This difference does not exceed 3% of the magnitude for the ground-truth kernel, which indicates that the most dominant values are well represented by the compact kernels $\hat{\mathbf{G}}^*_\ell(i, j)$. In Figure 4(e), we show that applying our kernel iteratively on the residual (Section 4.3) results in an algorithm that outperforms state-of-the-art solvers.

Additional examples for a grid resolution of 257×257 nodes are shown in Figure 5. The MGPCG solver (top row) outperforms our model for the simplest case of boundary conditions (Dirichlet exterior boundaries, no interior boundaries). However, when more complex boundaries are present (bottom row, mixed Dirichlet and Neumann exterior and interior boundaries), our model starts to outperform all classical solvers. We chose two different types of randomly generated right-hand side vectors on the top and bottom rows to showcase that, once optimized, our Green's function representation is oblivious to the right-hand side function. Distinct functions do not influence the convergence curves shown on the right, and additional examples are demonstrated in the supplemental material. We also compare the wall-clock time across resolutions for different solvers in Table 2. Although our method saves only about 2× the number of multiply-add operations compared to MGPCG, its intrinsic parallel nature enables it to reach a speedup of up to 12× at all resolutions.

Table 2. Runtime comparison of our method with CG and MGPCG solvers for different grid resolutions. All methods are set to stop at a residual of 5×10^{-4} in L^1 norm. We run all experiments on a NVIDIA GeForce RTX 2080Ti GPU.

Solver	33×33	65×65	129×129	257×257
CG [ms]	90	189	390	708
MGPCG [ms]	128	197	299	425
Ours [ms]	**13**	**16**	**26**	**33**

5.3. Predicting Green's Function for Various Boundary Geometries

We further evaluate the performance of MLPs to predict Green's function kernels based on general boundary conditions of arbitrary scenes. Note that the training does not rely on either the solution or the right-hand-side vectors. It only requires SDF values for all levels and the corresponding discrete Laplacian operator $(\phi_1, \ldots, \phi_L, \mathbf{A})$.

Dataset generation. We randomly place spheres and rectangles of random sizes into the scene to represent interior Neumann boundaries. Spheres and rectangles keep canonical orientations for simplicity, but are allowed to overlap in order to create more complex shapes. SDF values are computed at different discretization levels from the parameters of the spheres and rectangles to obtain $\phi_1, \ldots, \phi_L$. The voxelized obstacle setting and its corresponding discrete Laplacian $\mathbf{A}$ are computed

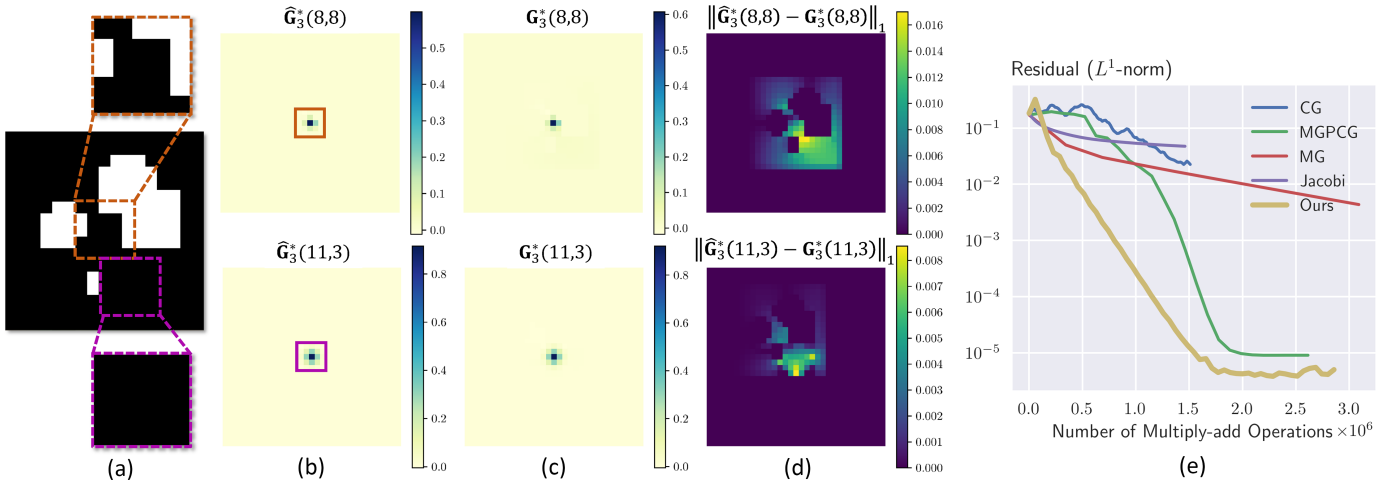


Fig. 4. We compare our truncated Green's function kernels (b) with the ground-truth (c). Both approaches are shown in the colored-line squares, the values outside them are zero paddings. A 33×33 regular grid is used (obstacles shown in (a)), and two kernels are extracted at the third level $\ell = 3$ (out of 4), at positions $(i, j) = (8, 8)$ (top), and $(11, 3)$ (bottom). In particular, our (5×5) convolution kernel (b) is compared to the (33×33) ground truth kernel (c) and the difference is shown in (d). Note that the error is only about $1 - 3\%$, showing that most of the dominant values in the dense $\mathbf{G}_\ell^*$ can be captured by the compact kernels. (e) shows our approximated residual Green's function can be used iteratively to solve the Poisson equation and outperforms classical solvers in terms of multiply-add operations.

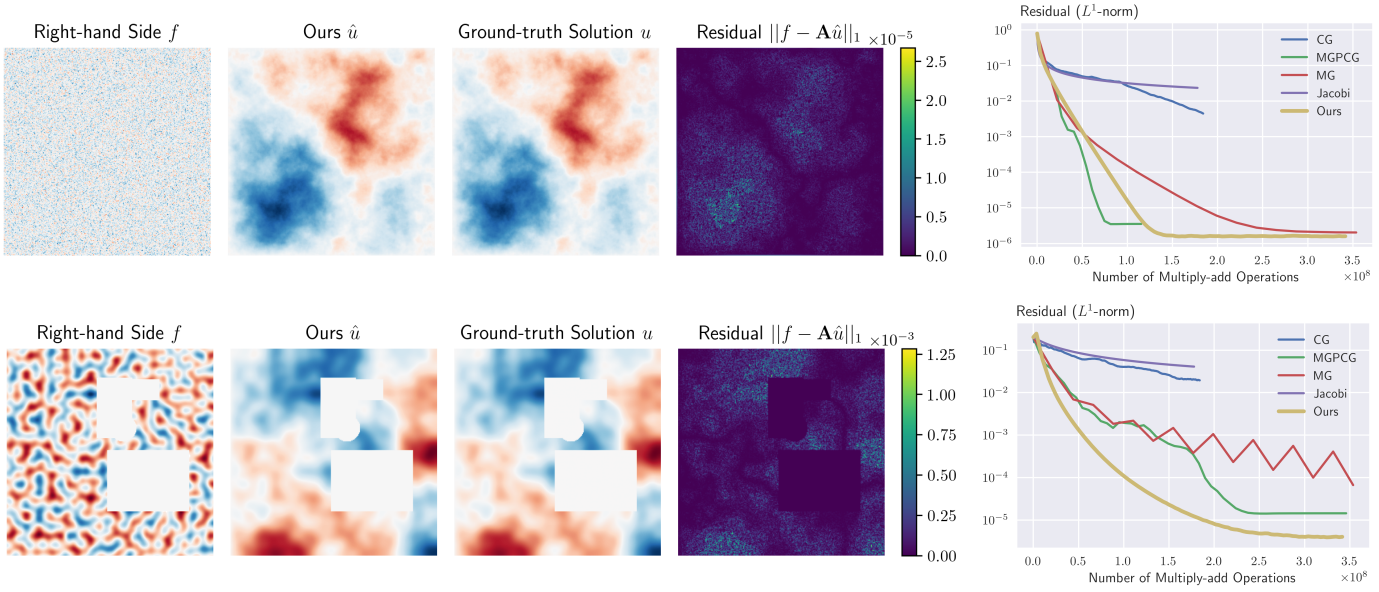


Fig. 5. Solving the Poisson Equation using multi-level Green's function optimized on single scenes in a resolution of 257×257. The scene in the top row has Dirichlet exterior boundaries and no interior boundaries. The scene in the bottom row has Dirichlet exterior boundary on the left side of the scene and Neumann exterior boundary on the other sides. The interior objects shown in white all have Neumann boundaries. Our method is inferior to MGPCG for the simplest case (top row), but outperforms all competing solvers when more complex boundary conditions exist (bottom row).

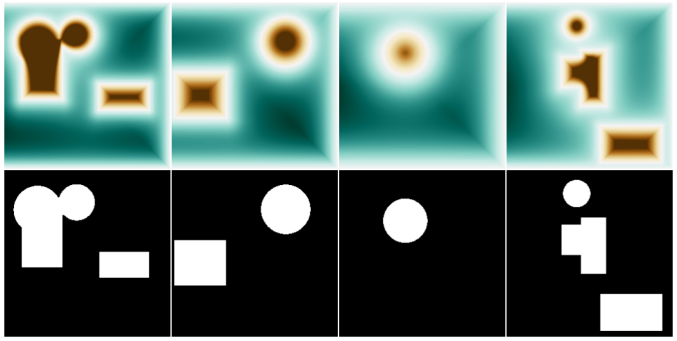


Fig. 6. Boundary condition samples from the training dataset. The top row shows the SDF (plotted in BrBG colormap: brown color for negative values, white color for zero and aqua for positive values) of the objects representing interior boundaries. The bottom row shows the corresponding binary map of the solution region (black) and out-of-boundary region (white). The left exterior boundary of the scene is assumed to be Dirichlet boundary, while the right, top and bottom exterior boundary are assumed to be Neumann.

at the finest discretization level. The discrete Laplacian is not stored for coarser levels, instead, they are downsampled relative to the finest resolution on the fly during training. We generate 1,000 scenes for the dataset (see examples in Figure 6), but the number of training samples is larger as we randomly select local patches. Moreover, 10 scenes are generated in a similar fashion (randomly placing spheres and rectangles) for creating the testing dataset. This means that none of the test sampeles is seen during training. The left side of the exterior boundary of all scenes is assumed to be Dirichlet while the other sides are assumed to be Neumann.

Predicting on new boundary settings. We train MLPs for grids with 129×129 nodes and show test results evaluated by solving Poisson equations in Figure 7 (a,b). Similar to the optimization results in Figure 5, the Green's kernel produced by the MLPs can outperform classical solvers in terms of residual convergence.

Fine tuning model output. Most of the test examples perform similarly well as in Figure 7 (a,b) For some examples with more complex interior boundary geometry though, inaccurate kernel predictions can result in slow convergence or even divergence of the residual, as is shown in Row (c) of Figure 7. The predicted kernels are suitable approximations in most of the regions, except the few grid points next to the interior Neumann boundaries. To fix this issue, we run a post-processing step on the failure cases. We solve the optimization in Equation (15) with the predicted Green's function kernels from MLPs as a starting value. The results of the fixed kernels are shown in Row (d) of Figure 7. The initialization reduces the number of iteration needed for the optimizations from 20 to 10, and reduces the runtime of optimization from $\sim$ 35 s to $\sim$ 13 s.

Ablation tests. To evaluate the results variation over multiple training runs, we train the model with different random seeds for initialization (Figure 8 (b, d)). Both seeds used for initialization show similar results in terms of both accuracy and convergence (Figure 8 (e)). We also tried using spatial gradients of the SDF patch as extra input channels (Figure 8 (c)). The plot (e) shows that incorporating the spatial gradient results in a slightly inferior convergence rate. As the spatial gradient is inherently contained in the input SDF patch, we argue that using it as extra channels is redundant.

5.4. An Example Application

Our multi-level Green's function can be used to replace classical solvers for Poisson Equations. We showcase an application to a fluid (smoke) simulation in Figure 9, where a Poisson solver is necessary at the pressure projection step [30]. In particular, the pressure projection equation is

$$\begin{cases} \nabla^2 p(\mathbf{x}) = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{u}(\mathbf{x}), & \mathbf{x} \in \Omega, \\ \frac{\partial p(\mathbf{x})}{\partial \mathbf{n}} = 0, & \mathbf{x} \in \partial\Omega^N, \\ p(\mathbf{x}) = 0, & \mathbf{x} \in \partial\Omega^D, \end{cases} \tag{17}$$

where $\partial\Omega^N$ and $\partial\Omega^D$ are fluid-solid and fluid-air interfaces, respectively. This step is the computational bottleneck in fluid solvers; therefore, it is crucial that its solution is computed efficiently. We restrict our study to a 2-D, 257×257 scene, with solid obstacles as shown in the top-left inset images in the Figure. The multi-level Green's function approximations are obtained by optimizing kernels for this single scene. We compare our results for several frames (top), to a MGPCG solver (bottom). Both solvers are set to stop at residual tolerance of 10^{-4} in L^∞ norm. The resulting density field of the smoke are similar between the two solvers. The simulation takes 104 ms per frame when using our Multi-level Green's function, and takes 576 ms per frame when using MGPCG solver.

6. Conclusions and Discussions

A multi-level Green's functions for 2-D Poisson Equations was presented as an alternative representation to standard SPAIs. Our novel optimization scheme is level-independent, which makes its evaluation efficient and memory-bound. Moreover, we show that our representation can be used to solve the discrete linear system by iteratively applying it on the residual of the error.

The locality property of the Laplace operator – its evaluation only depends on the neighborhood of the evaluated position – is the key for our method's efficiency. The Green's function, on the other hand, exhibits a *sparsity pattern in the frequency domain*: the low-frequency global information is crucial to reconstruct the solution of the current position, while magnitudes of high-frequencies are often small (Figure 1). Our designed multi-level Green's function approximation takes advantage of this property by representing low-frequency information through coarser grids, while high-frequency information is localized in finer grids. Therefore, our method can potentially be applied to other types of elliptical PDE as the differential operators are typically local.

By taking advantage that Green's function of the Poisson Equations only depends on the boundary conditions, we show that neural networks can be trained to Green's functions for general boundary settings. Once trained or optimized, our model can surpass state-of-the-art linear system solvers for certain settings in terms of convergence rate and runtime. Lastly, tested

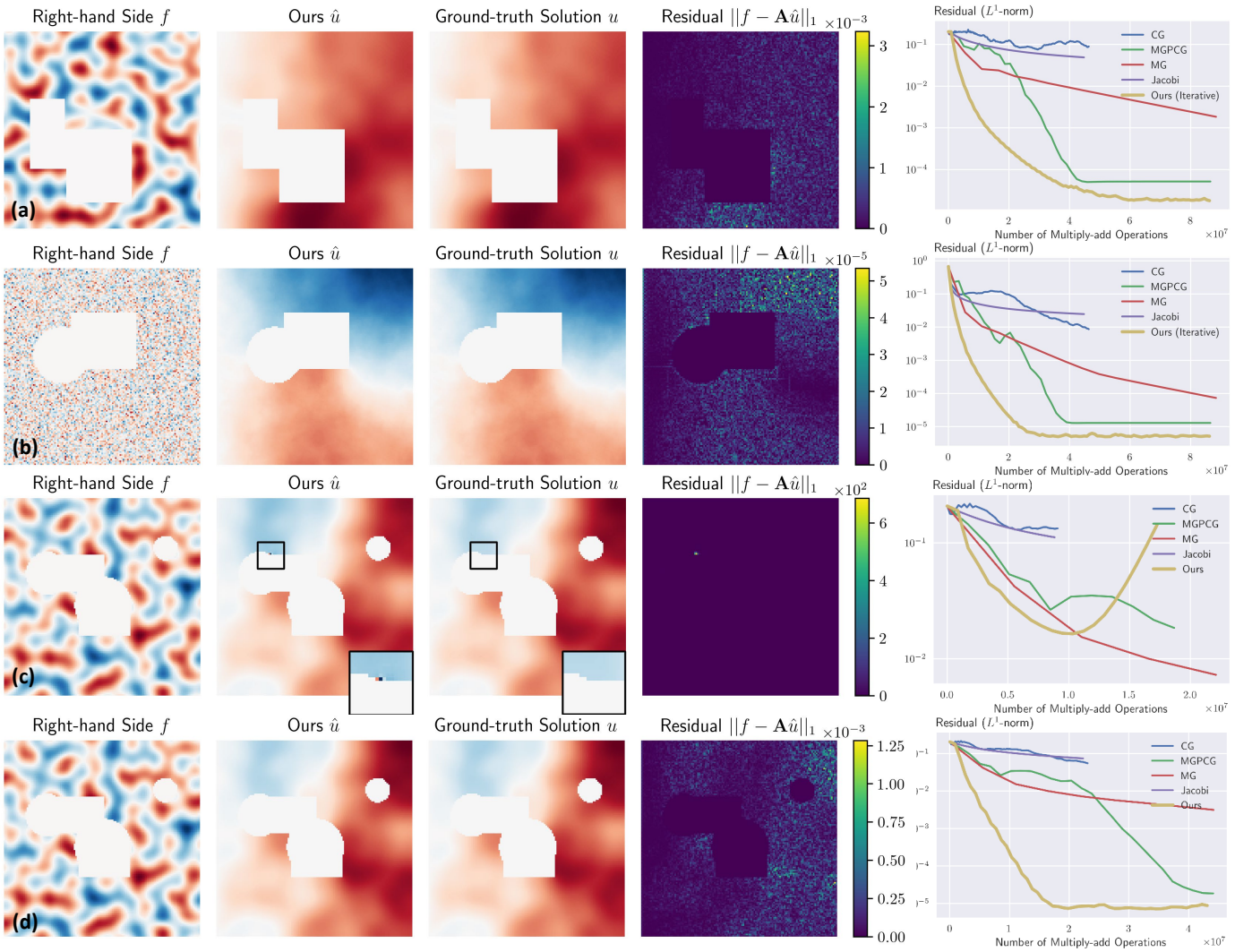


Fig. 7. Solving a Poisson Equation using the multi-level Green's function obtained by inferring the MLP. All scenes have Dirichlet exterior boundary on the left side of the scene and Neumann exterior boundary on the other sides. The interior objects (in white) are modelled by Neumann conditions. Rows (a) and (b): Multi-level Green's function outperforms other competing solvers in terms of residual convergence. Row (c) shows a divergent result due an inaccurate kernel prediction for a position that is simultaneously close to two objects (inset image). After post-processing problematic kernels through optimization (Row (d)), our method outperforms other solvers.

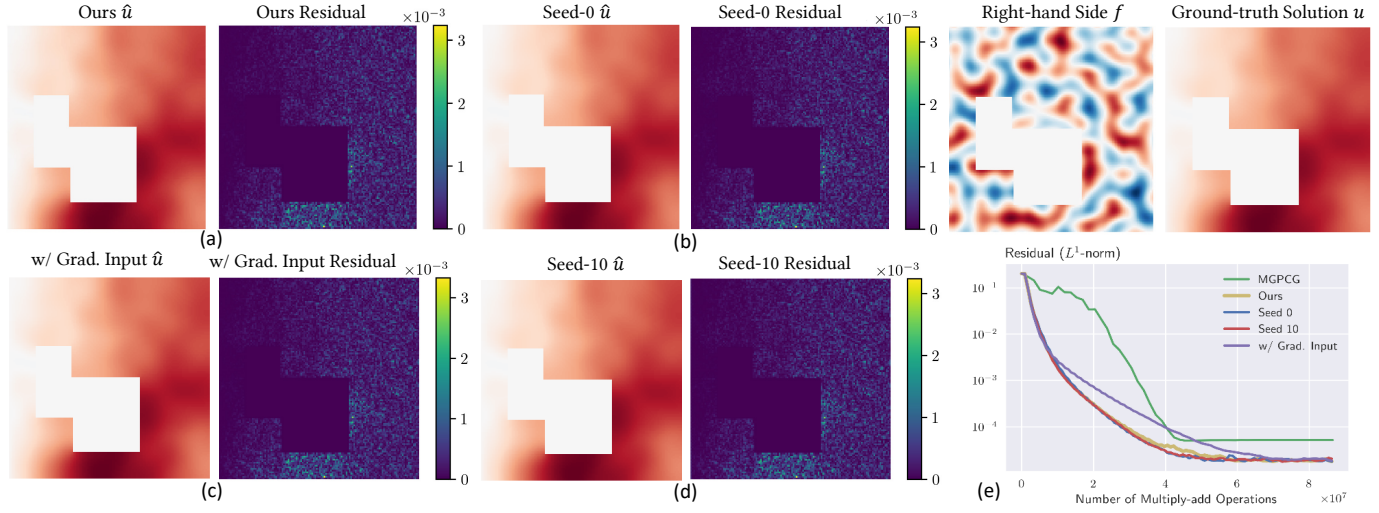


Fig. 8. Ablation study for evaluating initialization seeds and SDF gradients as local features. The MLP is trained with different random initialization seeds (b, d). To evaluate alternative local features, the spatial gradients of SDF values are used as extra input channels (c). All three variations show similar/inferior results and convergence (e) to our original model (a).

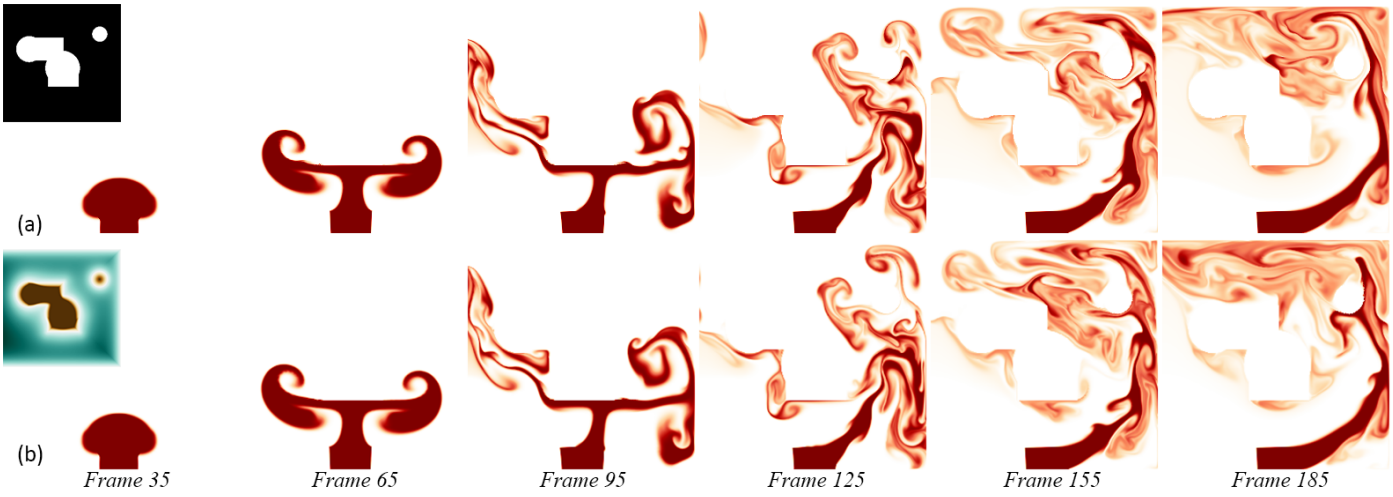


Fig. 9. Using the multi-level Green's function (a) and a MGPCG solver (b) to solve the projection step of a fluid solver with a grid resolution 257×257. The Neumann interior boundary settings and the SDF of the interior objects are shown as inset images in (a) and (b) respectively. The left exterior boundary is set to be Dirichlet while the other sides are set to be Neumann. The residual tolerance is set to be 10^{-4} in L^{∞} norm for both solvers. The simulation takes 104 [ms] per frame using multi-level Green's function, and takes 576 [ms] per frame using MGPCG solver. Both wall clock time count exclude initialization and optimization time.

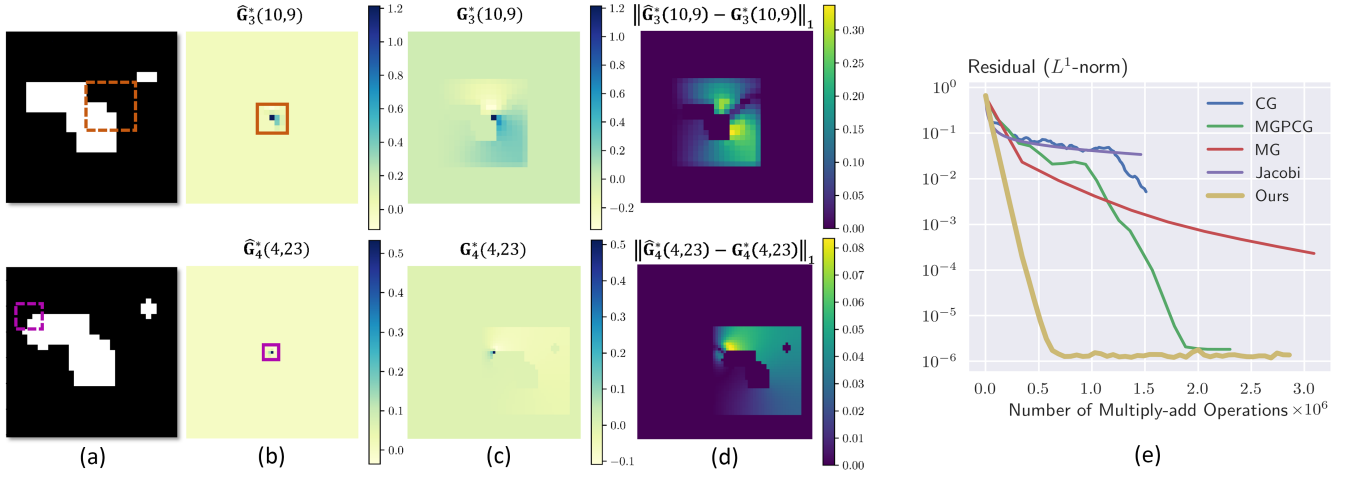


Fig. 10. We compare our truncated Green's function kernels (b) with the ground truth (c). We use a 33×33 scene with 4 levels (obstacles shown in (a)), and extract two kernels, at positions $(i, j) = (10, 9)$ (level $\ell = 3$, top), and $(4, 23)$ (level $\ell = 4$, bottom). In particular, our (5×5) convolution kernel (b) is compared to the (33×33) ground truth kernel (c) and the difference is shown in (d). (e) shows our approximated residual Green's function can be used iteratively to solve the Poisson equation and outperforms classical solvers in terms of multiply-add operations.

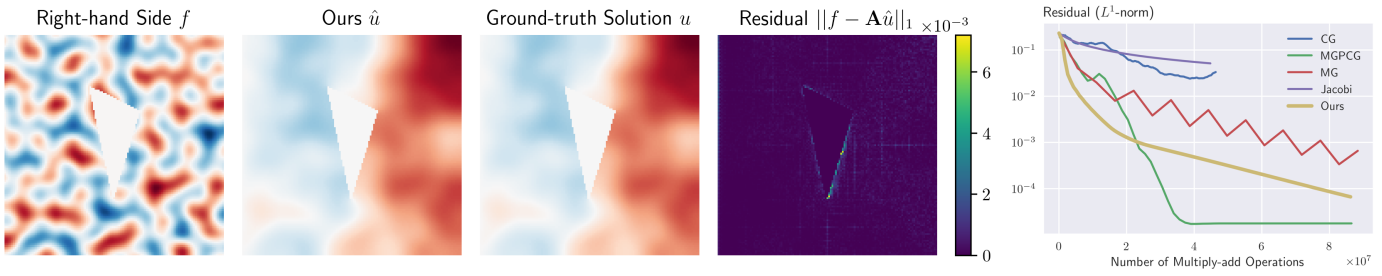


Fig. 11. Solving Poisson Equations using multi-level Green's function from the MLP output. The MLP is tested on an unseen triangular shape from training. The model performance is inferior to those in Figure 7.

our Green's function representation to replace pressure solvers in fluid simulation, and achieve a speedup of ~ 5x compared to state of the art classical solvers.

The up-front cost to get our approximated multi-level Green's function can pay off when the system is solved for multiple different right-hand sides (single scene optimization). Examples of such applications include Poisson matting [35], in which a Poisson equation of different right-hand side is solved in each iteration; Poisson image editing [36], where the boundary condition changes when a different region of blending is selected; grid-based fluid simulations [30], as the right-hand side computed from the advected velocity change at each time step.

6.1. Limitations

Inaccurate MLP kernel predictions. As shown in Figure 7 (c), our trained model underperforms on some individual patches around solid obstacles. This small potion of inaccurate kernels can result in slow convergence or even divergence if the SPD property is locally violated. We suspect that this behavior might come from SDF patches that are not well represented on the original dataset and the limited capability of the simple MLPs architectures employed. Therefore an improved network design and better sampling of the input examples during training may relieve this issue. Furthermore, when testing our model on scenes with unseen shapes (e.g., triangles) representing interior Neumann boundaries, our model also demonstrates inferior performance (Figure 11). We notice, however, that our method can be further fine-tuned for these scenarios by performing additional optimization iterations through Equation (15) (Figure 7 (d)).

Kernels without compact support. We found in experiments that ground-truth residual Green's kernels $\mathbf{G}^*_\ell(i, j)$ can have a non-compact support in some cases, as shown in Figure 10. The Figure shows the comparison of the approximated residual Green's kernel $\hat{\mathbf{G}}^*_\ell(i, j)$ (b) and $\mathbf{G}^*_\ell(i, j)$ (c), computed similarly as in Figure 4), at different spatial locations ((10, 9), top) and ((4, 23), bottom) for levels $\ell = 3$ and $\ell = 4$ respectively. The absolute difference between $\mathbf{G}^*_\ell(i, j)$ and $\hat{\mathbf{G}}^*_\ell(i, j)$ is shown on the right. Compared to the kernels in Figure 4, ground-truth kernels in this example have more values outside the compact range defined. However, our approximated kernels can still perform well when evaluating its convergence of solving Poisson Equations (Figure 10 (e)). We suspect the non-compact kernels may have a larger effect on the convergence rate in higher resolutions, and additional experiments for fine-tuning and ablation are needed.

6.2. Future Work

We plan to extend our work to support higher resolutions and 3-D settings. Moreover, our method could be applied to handle Poisson equations on arbitrary meshes, which would require more sophisticated numerical methods and efficient implementation of the downsampling and upsampling operators. Lastly, our method could offer significant speed-ups when dealing with varying boundaries (moving solids or liquids simulations). These scenarios were not explored in this paper; however, we believe that the method can be extended to handle such cases, as long as the training dataset represents moving boundaries accurately. These extensions would greatly increase the application of the proposed approach, since solving the Poisson equation is a widely pervasive problem.

References

[1] LeVeque, RJ. Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems. SIAM; 2007.

[2] Alkhalifah, T, Song, C, bin Waheed, U. Machine learned Green's functions that approximately satisfy the wave equation 2020;:2638–2642doi:10.1190/segam2020-3421468.1.

[3] Raissi, M, Perdikaris, P, Karniadakis, GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics 2019;378:686–707. URL: https://doi.org/10.1016/j.jcp.2018.10.045. doi:10.1016/j.jcp.2018.10.045.

[4] Ichimura, T, Fujita, K, Hori, M, Maddegedara, L, Ueda, N, Kikuchi, Y. A Fast Scalable Iterative Implicit Solver with Green's function-based Neural Networks 2020;:61–68doi:10.1109/ScalA51936.2020.00013.

[5] Li, Z, Kovachki, N, Azizzadenesheli, K, Liu, B, Bhattacharya, K, Stuart, A, et al. Multipole Graph Neural Operator for Parametric Partial Differential Equations 2020;URL: http://arxiv.org/abs/2006.09535. arXiv:2006.09535.

[6] Li, Z, Kovachki, N, Azizzadenesheli, K, Liu, B, Bhattacharya, K, Stuart, A, et al. Fourier Neural Operator for Parametric Partial Differential Equations 2020;URL: http://arxiv.org/abs/2010.08895. arXiv:2010.08895.

[7] Li, Z, Kovachki, N, Azizzadenesheli, K, Liu, B, Bhattacharya, K, Stuart, A, et al. Neural Operator: Graph Kernel Network for Partial Differential Equations 2020;URL: http://arxiv.org/abs/2003.03485. arXiv:2003.03485.

[8] Feliu-Faba, J, Fan, Y, Ying, L. Meta-learning Pseudo-differential Operators with Deep Neural Networks 2019;URL: http://arxiv.org/abs/1906.06782http://dx.doi.org/10.1016/j.jcp.2020.109309. doi:10.1016/j.jcp.2020.109309. arXiv:1906.06782.

[9] Gin, CR, Shea, DE, Brunton, SL, Kutz, JN. Deepgreen: deep learning of green's functions for nonlinear boundary value problems. Scientific reports 2021;11(1):1–14.

[10] Chung, , Yau, ST. Discrete Green ' s theorem. Context 2000;(1):2–5.

[11] Frasca, M, Khurshudyan, AZ. General representation of nonlinear Green's function for second order differential equations nonlinear in the first derivative. arXiv 2018;:1–15arXiv:1806.00274.

[12] Kondor, R, Teneva, N, Garg, V. Multiresolution Matrix Factorization. In: Xing, EP, Jebara, T, editors. Proceedings of the 31st International Conference on Machine Learning; vol. 32 of *Proceedings of Machine Learning Research*. Bejing, China: PMLR; 2014, p. 1620–1628. URL: https://proceedings.mlr.press/v32/kondor14.html.

[13] Ithapu, VK, Kondor, R, Johnson, SC, Singh, V. The Incremental Multiresolution Matrix Factorization Algorithm 2017;URL: http://arxiv.org/abs/1705.05804. arXiv:1705.05804.

[14] Mudrakarta, PK, Kondor, R. A generic multiresolution preconditioner for sparse symmetric systems. 2017. arXiv:1707.02054.

[15] Mallat, S. A theory for multiresolution signal decomposition: the wavelet representation. IEEE Transactions on Pattern Analysis and Machine Intelligence 1989;11(7):674–693. URL: http://ieeexplore.ieee.org/document/192463/. doi:10.1109/34.192463.

[16] Coifman, RR, Maggioni, M. Diffusion wavelets. Applied and Computational Harmonic Analysis 2006;21(1):53–94. URL: https://linkinghub.elsevier.com/retrieve/pii/S106352030600056X. doi:10.1016/j.acha.2006.04.004.

[17] Chan, TF, Tang, WP, Wan, WL. Wavelet sparse approximate inverse preconditioners. BIT Numerical Mathematics 1997;37(3):644–660. URL: http://link.springer.com/10.1007/BF02510244. doi:10.1007/BF02510244.

[18] Benzi, M, Meyer, CD, Tůma, M. A Sparse Approximate Inverse Preconditioner for the Conjugate Gradient Method. SIAM Journal on Scientific Computing 1996;17(5):1135–1149. URL: http://epubs.siam.org/doi/10.1137/S1064827594271421. doi:10.1137/S1064827594271421.

[19] Benzi, M, Cullum, JK, Tuma, M. Robust Approximate Inverse Preconditioning for the Conjugate Gradient Method. SIAM Journal on Scientific Computing 2000;22(4):1318–1332. URL: http://epubs.siam.org/doi/10.1137/S1064827599356900. doi:10.1137/S1064827599356900.

[20] Bridson, R, Tang, WP. Multiresolution Approximate Inverse Preconditioners. SIAM Journal on Scientific Computing 2001;23(2):463–479. URL: http://epubs.siam.org/doi/10.1137/S1064827500373784. doi:10.1137/S1064827500373784.

[21] Tang, WP, Wan, WL. Sparse Approximate Inverse Smoother for Multigrid. SIAM Journal on Matrix Analysis and Applications 2000;21(4):1236–1252. URL: http://epubs.siam.org/doi/10.1137/S0895479899339342. doi:10.1137/S0895479899339342.

[22] Bröker, O. Sparse approximate inverse smoothers for geometric and algebraic multigrid. Applied Numerical Mathematics 2002;41(1):61–80. URL: https://linkinghub.elsevier.com/retrieve/pii/S0168927401001106. doi:10.1016/S0168-9274(01)00110-6.

[23] Barati Farimani, A, Gomes, J, Pande, VS. Deep Learning the Physics of Transport Phenomena. arXiv 2017;94305. arXiv:1709.02432.

[24] Sharma, R, Farimani, AB, Gomes, J, Eastman, P, Pande, V. Weakly-supervised deep learning of heat transport via physics informed loss. arXiv 2018;arXiv:1807.11374.

[25] Tompson, J, Schlachter, K, Sprechmann, P, Perlin, K. Accelerating Eulerian Fluid Simulation With Convolutional Networks 2016;URL: http://arxiv.org/abs/1607.03597. doi:10.1145/1143844.1143891. arXiv:1607.03597.

[26] Xiao, X, Zhou, Y, Wang, H, Yang, X. A Novel CNN-Based Poisson Solver for Fluid Simulation. IEEE Transactions on Visualization and Computer Graphics 2020;26(3):1454–1465. doi:10.1109/TVCG.2018.2873375.

[27] Tang, W, Shan, T, Dang, X, Li, M, Yang, F, Xu, S, et al. Study on a Poisson's equation solver based on deep learning technique. 2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium, EDAPS 2017 2018;2018-Janua:1–3. doi:10.1109/EDAPS.2017.8277017. arXiv:1712.05559.

[28] Zhang, Z, Zhang, L, Sun, Z, Erickson, N, From, R, Fan, J. Solving poisson's equation using deep learning in particle simulation of PN junction. arXiv 2018;:2019–2022.

[29] Hsieh, JT, Zhao, S, Eismann, S, Mirabella, L, Ermon, S. Learning neural pde solvers with convergence guarantees. arXiv 2019;(i):1–14. arXiv:1906.01200.

[30] Stam, J. Stable fluids. Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999 1999;:121–128doi:10.1145/311535.311548.

[31] McAdams, A, Sifakis, E, Teran, J. A parallel multigrid Poisson solver for fluids simulation on large grids. Eurographics/ACM SIGGRAPH Symposium on Computer Animation 2010;:10URL: https://dl.acm.org/citation.cfm?id=1921438. doi:10.2312/SCA/SCA10/065-073.

[32] Mildenhall, B, Srinivasan, PP, Tancik, M, Barron, JT, Ramamoorthi, R, Ng, R. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis 2020;URL: http://arxiv.org/abs/2003.08934. arXiv:2003.08934.

[33] Perlin, K. Image Synthesizer. Computer Graphics (ACM) 1985;19(3):287–296. doi:10.1145/325165.325247.

[34] Paszke, A, Gross, S, Massa, F, Lerer, A, Bradbury, J, Chanan, G, et al. PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 2019;32(NeurIPS). arXiv:1912.01703.

[35] Sun, J, Jia, J, Tang, CK, Shum, HY. Poisson matting. In: ACM SIGGRAPH 2004 Papers. 2004, p. 315–321.

[36] Pérez, P, Gangnet, M, Blake, A. Poisson image editing. ACM Transactions on Graphics 2003;22(3):313–318. URL: https://dl.acm.org/doi/10.1145/882262.882269. doi:10.1145/882262.882269.