Contents lists available at ScienceDirect

Computers & Graphics

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

Neural Green's Function for Laplacian Systems – Supplymental Materials

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. Optimization Objective for Multi-level Green's Function Representation

1.1. Derivation for level ℓ

We show that $\hat{\mathbf{G}}_\ell^* = \arg\min_{\mathbf{G}_\ell^*} \left\| \mathbf{G}_\ell^* \mathbf{A}_\ell - \left(\mathbf{A}_\ell - \mathbf{U}_{\ell-1}^\ell \mathbf{D}_\ell^{\ell-1} \mathbf{I}_\ell \right) \right\|_2^2$ (Equation (15) in the main text) is a sufficient condition for $\hat{\mathbf{G}}_\ell = \arg\min_{\mathbf{G}_\ell} \|\mathbf{G}_\ell \mathbf{A}_\ell - \mathbf{I}_\ell\|_2^2$ (Equation (14) in the main text) given the definition $\mathbf{G}_\ell = \mathbf{U}_{\ell-1}^\ell \mathbf{G}_{\ell-1} \mathbf{D}_\ell^{\ell-1} + \mathbf{G}_\ell^*$ (Equation (13) in the main text)

We write down the optimized $\hat{\mathbf{G}}_\ell$ for both levels ℓ (finer) and $\ell - 1$ (coarser):

$$\hat{\mathbf{G}}_{\ell-1} \mathbf{A}_{\ell-1} - \mathbf{I}_{\ell-1} = \Sigma_{\ell-1}, \tag{1}$$

$$\hat{\mathbf{G}}_\ell \mathbf{A}_\ell - \mathbf{I}_\ell = \Sigma_\ell. \tag{2}$$

Here $\Sigma_{\ell-1} \simeq 0$ and $\Sigma_\ell \simeq 0$ are the residuals after the optimization of $\mathbf{G}_{\ell-1}$ and $\mathbf{G}_\ell$. Since we are only concerned with two levels, we use $\mathbf{U} = \mathbf{U}_{\ell-1}^\ell$ and $\mathbf{D} = \mathbf{D}_\ell^{\ell-1}$ to simplify the notations.

We now bring the definitions $\mathbf{A}_{\ell-1} = \mathbf{D}_L^{\ell-1} \mathbf{A} \mathbf{U}_{\ell-1}^L = \mathbf{D} \mathbf{A}_\ell \mathbf{U}$, $\mathbf{I}_{\ell-1} = \mathbf{D}_L^{\ell-1} \mathbf{I} \mathbf{U}_{\ell-1}^L = \mathbf{D} \mathbf{I}_\ell \mathbf{U}$ and $\mathbf{G}_\ell = \mathbf{U} \mathbf{G}_{\ell-1} \mathbf{D} + \mathbf{G}_\ell^*$ into (1) and (2), and get

$$\hat{\mathbf{G}}_{\ell-1} (\mathbf{D} \mathbf{A}_\ell \mathbf{U}) - \mathbf{D} \mathbf{I}_\ell \mathbf{U} = \Sigma_{\ell-1}, \tag{3}$$

$$\left(\mathbf{U} \hat{\mathbf{G}}_{\ell-1} \mathbf{D} + \hat{\mathbf{G}}_\ell^* \right) \mathbf{A}_\ell - \mathbf{I}_\ell = \Sigma_\ell. \tag{4}$$

We further organize the above equations by removing all the parentheses and get

$$\hat{\mathbf{G}}_{\ell-1} \mathbf{D} \mathbf{A}_\ell \mathbf{U} - \mathbf{D} \mathbf{I}_\ell \mathbf{U} = \Sigma_{\ell-1}, \tag{5}$$

$$\mathbf{U} \hat{\mathbf{G}}_{\ell-1} \mathbf{D} \mathbf{A}_\ell + \hat{\mathbf{G}}_\ell^* \mathbf{A}_\ell - \mathbf{I}_\ell = \Sigma_\ell. \tag{6}$$

Since the first term in both equations look alike, we can try to eliminate them by combining the two equations. To do that, as $\mathbf{U} \neq \mathbf{0}$ we can left multiply $\mathbf{U}_{\ell-1}^\ell$ in (5) and right multiply $\mathbf{U}_{\ell-1}^\ell$

in (6).

$$\mathbf{U}\hat{\mathbf{G}}_{\ell-1}\mathbf{D}\mathbf{A}_\ell\mathbf{U} - \mathbf{U}\mathbf{D}\mathbf{I}_\ell\mathbf{U} = \mathbf{U}\Sigma_{\ell-1}, \tag{7}$$

$$\mathbf{U}\hat{\mathbf{G}}_{\ell-1}\mathbf{D}\mathbf{A}_\ell\mathbf{U} + \hat{\mathbf{G}}^*_\ell\mathbf{A}_\ell\mathbf{U} - \mathbf{I}_\ell\mathbf{U} = \Sigma_\ell\mathbf{U}. \tag{8}$$

Now we subtract (8) by (7), and get

$$\left(\hat{\mathbf{G}}^*_\ell\mathbf{A}_\ell - \mathbf{I}_\ell + \mathbf{U}\mathbf{D}\mathbf{I}_\ell\right)\mathbf{U} = \Sigma_\ell\mathbf{U} - \mathbf{U}\Sigma_{\ell-1} \simeq 0 \tag{9}$$

Since $\mathbf{U} \neq \mathbf{0}$, $\left(\hat{\mathbf{G}}^*_\ell\mathbf{A}_\ell - \mathbf{I}_\ell + \mathbf{U}\mathbf{D}\mathbf{I}_\ell\right) \simeq 0$ is a sufficient condition for (9) to hold. Note that from (1) and (2) to (9), sufficiency and necessity always holds, so $\hat{\mathbf{G}}^*_\ell = \arg\min_{\mathbf{G}^*_\ell} \left\|\mathbf{G}^*_\ell\mathbf{A}_\ell - \left(\mathbf{I}_\ell - \mathbf{U}^\ell_{\ell-1}\mathbf{D}^{\ell-1}_\ell\mathbf{I}_\ell\right)\right\|^2_2$ is a sufficient condition for $\hat{\mathbf{G}}_\ell = \arg\min_{\mathbf{G}_\ell} \|\mathbf{G}_\ell\mathbf{A}_\ell - \mathbf{I}_\ell\|^2_2$.

1.2. An alternative two-level derivation

We now consider the two level case ($L = 2$) for a simpler derivation. We want that the level $\ell = 2$ approximates the matrix inverse as:

$$\hat{\mathbf{G}}_2\mathbf{A} - \mathbf{I} = 0 \tag{10}$$

with $\hat{\mathbf{G}}_2$ depending on the coarser level $\ell = 1$ as

$$\hat{\mathbf{G}}_2 = \mathbf{U}\hat{\mathbf{G}}_1\mathbf{D} + \hat{\mathbf{G}}^*_2 \tag{11}$$

To make this discretization level-independent, we enforce the coarser level $\ell = 1$ to solve the downsampled version of Laplace system:

$$\hat{\mathbf{G}}_1\left(\mathbf{D}\mathbf{A}\mathbf{U}\right) = \mathbf{D}\mathbf{U} \tag{12}$$

By substituting (12) into (11), we get

$$\left(\mathbf{U}\left[\mathbf{D}\mathbf{U}(\mathbf{D}\mathbf{A}\mathbf{U})^{-1}\right]\mathbf{D} + \hat{\mathbf{G}}^*_2\right)\mathbf{A} - \mathbf{I} = 0 \tag{13}$$

This equation holds:

$$\hat{\mathbf{G}}^*_2\mathbf{A} + \mathbf{U}\mathbf{D} - \mathbf{I} = 0 \tag{14}$$

2. Further Implementation Details

2.1. Multi-level Green's function

Each level ℓ of our multi-level Green's function approximation $\hat{\mathbf{G}}^*_\ell$ is essentially a sparse matrix that can be efficiently implemented by any framework supporting sparse computations. To fully utilize the Pytorch framework for its automatic differentiation as well as the parallel nature of convolutional operations, we implement $\hat{\mathbf{G}}^*_\ell$ through spatially varying convolutions, i.e. sliding window of compact kernels $\mathbf{G}^*_\ell(i_\ell, j_\ell)$ of size $k_\ell \times k_\ell$ that vary at each position (i_ℓ, j_ℓ). When applied on the right-hand side at level ℓ, the sliding window is multiplied with the corresponding values in the right-hand side:

$$\mathbf{u}_{i_\ell,j_\ell} = \sum_{i,j\in\mathcal{N}(i_\ell,j_\ell)} \left[\mathbf{G}^*_\ell(i_\ell, j_\ell)\right]_{i,j}\mathbf{f}_{i,j}, \tag{15}$$

where $\mathcal{N}(i_\ell, j_\ell)$ denotes the neighourhood of (i_ℓ, j_ℓ). Note that the above equation is similar to a normal 2-D convolution operation, except that the convolutional kernel varies at each position. Upsampling and downsampling operators $\mathbf{U}^\ell_{\ell-1}$, $\mathbf{D}^{\ell-1}_\ell$ are implemented in the same way through spatially-varying convolutions. The only difference is that the kernel values for upsampling and downsampling operators are fully determined by the linear stencil (Equation (11) in the main text) while kernel values for Green's functions are obtained through either optimization (Equation (15) in the main text) or neural network feed-forward evaluation $\mathbf{G}^*_\ell(i_\ell, j_\ell) = \mathcal{M}^\Theta_\ell(\phi_\ell(i_\ell, j_\ell))$.

2.2. Dataset generation

To generate the training dataset, we first randomly place spheres and rectangles as described in in Section 5.3 in the main text to generate the scene setting. As we only have primitive shapes in the scene, the corresponding SDF is easily computed at each discretization level $(\phi_1, ..., \phi_L)$. When two or more primitive shapes overlap, we compute the union of the two or more SDF through smooth blending [1] with $\alpha = 1$. The discrete Laplacian matrix of the scene $\mathbf{A}(= \mathbf{A}_L)$ is then computed based on the voxelized obstacle represented boundary conditions at level L, and stored in the dataset. To compute $\mathbf{A}L$, we follow Bridson [2] (Chapter 5) to adapt the Laplace stencil according to the boundary conditions of the nearby cells. The Laplacian matrices of other levels ℓ is downsampled from $\mathbf{A}_\ell = \mathbf{D}^\ell_L\mathbf{A}_L\mathbf{U}^L_\ell$ on the fly during training. At each training iteration for level ℓ, we load $(\phi_\ell, \mathbf{A})$ from the dataset, evaluate the MLP, compute the loss in Equation (16) in the main text, and back-propagate to update the weights Θ of the MLP.

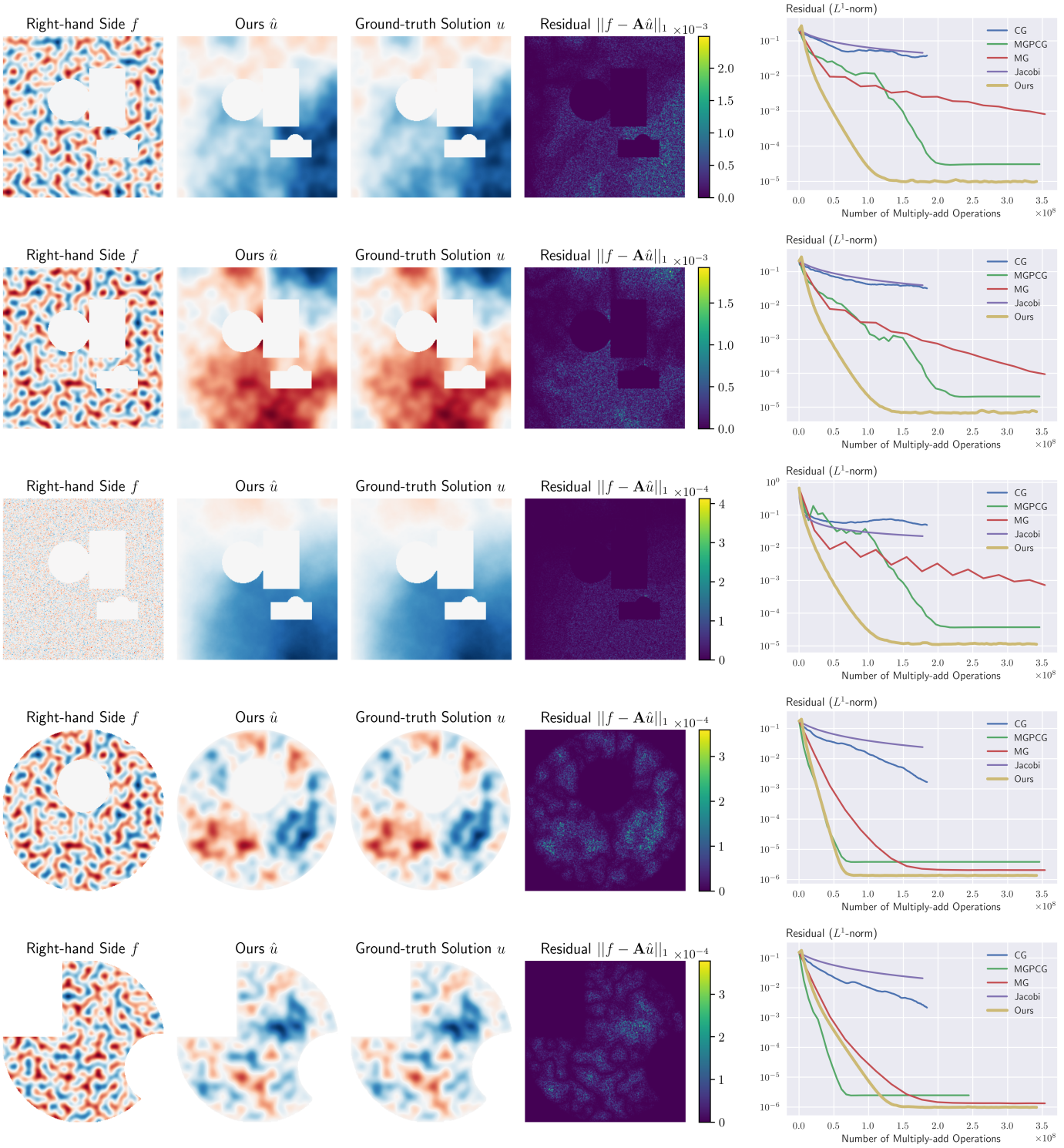

Fig. 1. Solving Poisson Equation using multi-level Green's function optimized on single scenes in resolution $n = 257$. Top 3 rows: The same Green's function is applied to three different right-hand side vectors. In all three cases, multi-level Green's function outperforms other competing solvers in terms of convergence of residual. Bottom 2 rows: the Green's function is optimized for a sphere shaped scene (Row 4) and a more irregular shaped scene (Row 5). All boundary in both scenes are Dirichlet boundaries. The convergence of Green's function is similar to or slightly worse than MGPCG, but better than other competing solvers.

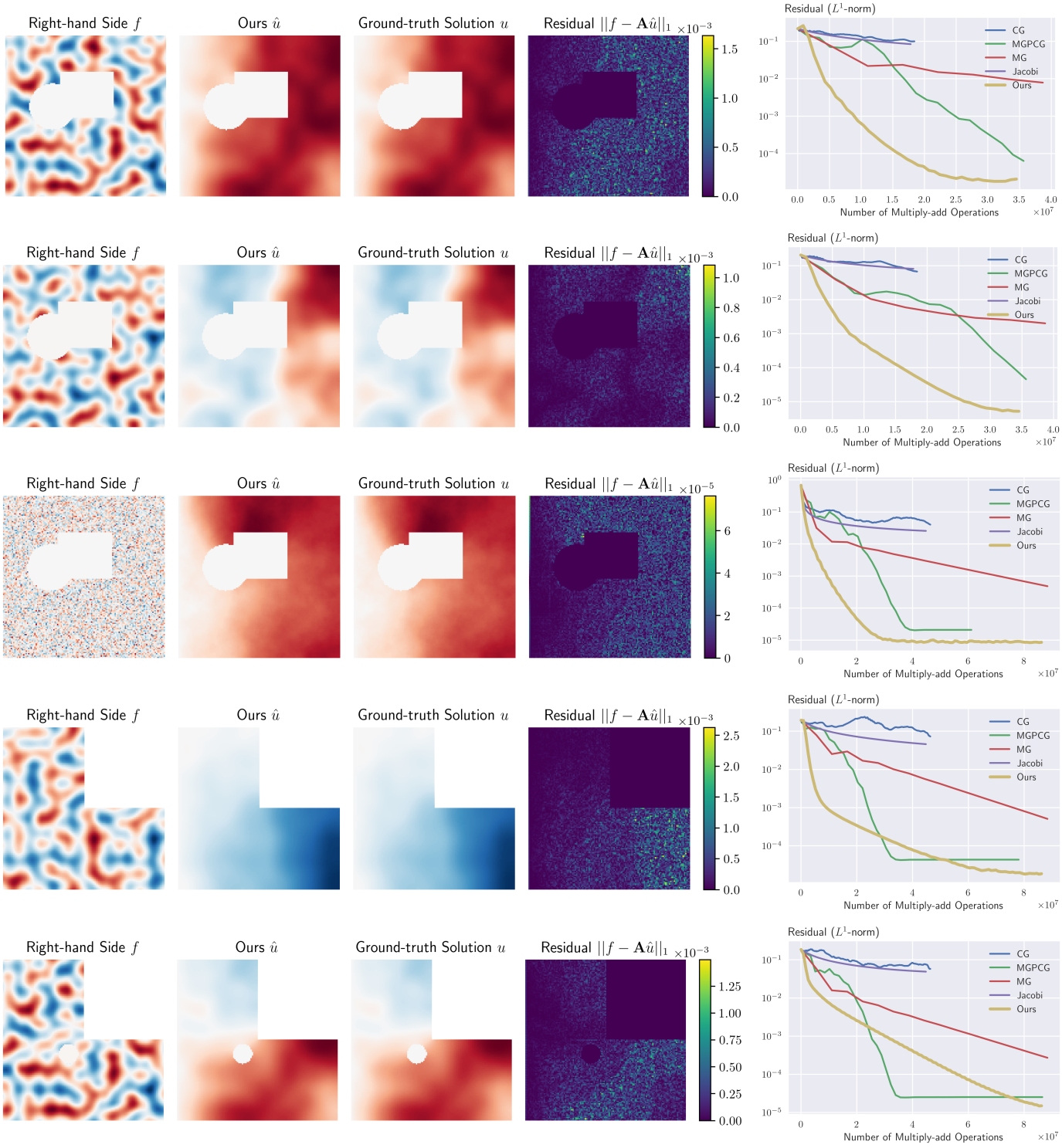

Fig. 2. Solving Poisson Equation using multi-level Green's function from MLP output in resolution $n = 129$. Top 3 rows: the same Green's function is applied to three different right-hand side vectors. In all three cases, multi-level Green's function outperforms other competing solvers in terms of convergence of residual. Bottom 2 rows: the MLP is evaluated on L-shaped scenes. The scene has Dirichlet boundary on the left side, and Neumann boundary on other sides. The interior boundary in Row 5 is also Neumann boundary. The convergence of Green's function is slightly worse than MGPCG, but better than other competing solvers.

To generate the test set, we use the same procedure as in the training dataset for generating $(\phi_1, ..., \phi_L, \mathbf{A})$. We additionally generate 5 right-hand side vectors for each scene through either Perlin noise [3] or Gaussian noise. To get the ground-truth solution vector, we use the Conjugate Gradient solver to solve for the Poisson equation of different right-hand side vectors separately with a tolerance of 10^{-12} in L^∞ norm.

3. Extended Results

We show extended results of multi-level Green's function for solving Poisson Equations.

3.1. Applying the same Green's function on different right-hand side vectors

One of the major advantage of using Green's function to solve linear PDE is that the Green's function can be used for arbitrary forcing term. We show our multi-level Green's function representation has the same properties. Both Green's function optimized for a single scene (Figure 1, top 3 rows), and the output Green's function kernels from MLP (Figure 2, top 3 rows) are agnostic to right-hand side vectors. Once optimized or trained, the Green's function can be used for solving Poisson Equations with arbitrary right-hand side, without changing the convergence properties.

3.2. Irregular exterior boundaries

We further evaluate the multi-level Green's function on more irregular domains. In Figure 1 (bottom 2 rows), Green's function is optimized for a sphere shaped scene and a more irregular shaped scene in Dirichlet boundary conditions. In both scenes, Green's function solvers converge similarly or slightly worse than MGPCG, but perform better than other solves. In Figure 2 (bottom 2 rows), the same MLP is used to evaluate a L-shaped scene with mixed Dirichlet and Neumann boundary conditions, which never appeared in the training set. The Green's function solver is slightly inferior to MGPCG in these cases at lower tolerance regions, but better than other solvers.

References

[1] Pasko, AA, Savchenko, VV. Blending operations for the functionally based constructive geometry 1994;.
[2] Bridson, R. Fluid simulation for computer graphics. CRC press; 2015.
[3] Perlin, K. Image Synthesizer. Computer Graphics (ACM) 1985;19(3):287–296. doi:10.1145/325165.325247.