Lightning-fast Method of Fundamental Solutions

JIONG CHEN, Inria, France
FLORIAN SCHÄFER, Georgia Institute of Technology, USA
MATHIEU DESBRUN, Inria / Ecole Polytechnique, France

Fig. 1. **Speeding up the method of fundamental solutions.** As our work allows for lightning-fast solving of dense linear equations from MFS and BEM through an efficient *inverse-Cholesky preconditioner*, boundary-type methods can now be leveraged to handle large-scale 3D problems as this illustrative scene demonstrates, with Laplace's equation (radiating colors from the tower), linear elasticity (e.g., deforming eagles through manipulating boxes), and Helmholtz equation (scattering of input waves on islands), each of these examples involving from $10K$ to $200K$ degrees of freedom, computed on a laptop.

The method of fundamental solutions (MFS) and its associated boundary element method (BEM) have gained popularity in computer graphics due to the reduced dimensionality they offer: for three-dimensional linear problems, they only require variables on the domain boundary to solve and evaluate the solution throughout space, making them a valuable tool in a wide variety of applications. However, MFS and BEM have poor computational scalability and huge memory requirements for large-scale problems, limiting their applicability and efficiency in practice. By leveraging connections with Gaussian Processes and exploiting the sparse structure of the inverses of boundary integral matrices, we introduce a variational preconditioner that can be computed via a sparse inverse-Cholesky factorization in a massively parallel manner. We show that applying our preconditioner to the Preconditioned Conjugate Gradient algorithm greatly improves the efficiency of MFS or BEM solves, up to four orders of magnitude in our series of tests.

CCS Concepts: • **Mathematics of computing** → **Solvers**; • **Computing methodologies** → **Computer graphics**.

Additional Key Words and Phrases: Method of Fundamental Solutions, inverse Cholesky factorization, preconditioning, Gaussian Processes

ACM Reference Format:
Jiong Chen, Florian Schäfer, and Mathieu Desbrun. 2024. Lightning-fast Method of Fundamental Solutions. *ACM Trans. Graph.* 43, 4, Article 77 (July 2024), 16 pages. https://doi.org/10.1145/3658199

1 INTRODUCTION

When dealing with a linear partial differential equation (PDE) with imposed values on the boundary of a domain $\Omega \subset \mathbb{R}^3$, the use of the Method of Fundamental Solutions (MFS) or of the Boundary Element Method (BEM), both based on the Green's function of the linear operator involved in the PDE, provides a way to solve the PDE with only surface-based degrees of freedom (DoFs) — instead of performing both a volumetric discretization of the 3D domain and the associated much-larger linear solve. Therefore, this "boundary integral" approach dramatically diminishes the dimensionality of the

Authors' addresses: J. Chen (jiong.chen@inria.fr): Inria Saclay, Palaiseau, France; F. Schäfer (florian.schaefer@cc.gatech.edu): Georgia Institute of Technology, Atlanta, USA; M. Desbrun (mathieu.desbrun@inria.fr): Inria Saclay/Ecole Polytechnique (IP Paris), Palaiseau, France.

problem and removes the pain of generating a fine volumetric mesh. Consequently, it has been used to approach a series of Computer Graphics (CG) problems with smaller counts of variables to solve, from elasticity [James and Pai 1999] and digital sculpting [De Goes and James 2017], to diffusion curves [Bang et al. 2023] and water waves [Schreck et al. 2019].

However, the resulting smaller matrices involved in the Boundary Integral Equations (BIE) from either BEM or MFS are *fully populated* and *ill-conditioned* after discretization of the boundary, posing significant challenges for *direct solvers*: just assembling all the entries of the dense matrix of the BIE can quickly fill up memory for large scale problems; worse, the cubic complexity of factorizing this dense matrix often renders direct solves impractical even for problems of moderate sizes. *Iterative solvers* do not face an easier situation either: the condition number of the BIE matrix often deteriorates as the number of DoFs increases. Even for symmetric positive-definite matrices, Conjugate Gradient (CG) solvers are at times dramatically slow to reach a low-enough error norm for large problems, while asymmetric BIE matrices require GMRES or BiStabCG-based solves which might even fail to converge due to a lack of guarantees for such matrices. As the dense nature of BIE matrices imposes high memory requirements and computational efforts in practice, stochastic approaches have gained popularity recently due to their grid-free nature and avoidance of global solves [Sawhney and Crane 2020; Sawhney et al. 2022; Sugimoto et al. 2023]: these Monte-Carlo methods are able to provide a quick preview of the solution *pointwise* and are arguably the only methods currently available to deal with problems with hundreds of thousands of DoFs. Yet, their slow convergence rate and redundancy in pointwise evaluations limit their applicability in graphics and simulation tasks for which evaluation of the solution is needed throughout the domain and accuracy is paramount, leaving only MFS and BEM as viable alternatives despite their inability to scale well.

This long-standing numerical difficulty is, in a sense, antithetical to the oft-observed problems in solving linear systems coming from the discretization of a linear differential equation: the latter involves larger-sized matrices (induced by a volume discretization) that are sparse (due to locality of derivatives) and for which a large variety of efficient preconditioners have been proposed [Zhu et al. 2010; Krishnan and Szeliski 2011; Chen et al. 2021b; Wu et al. 2022; Shao et al. 2022]. As mentioned above, Green's functions (i.e., solutions of the differential equation subject to a singularity load) allow us to write much more compact linear systems involving only boundary variables; but this time, the resulting matrices are *dense*. So traditional numerical preconditioning methods designed for sparse differential operators (such as multigrid, incomplete LU, or Cholesky factorization) are ineffective in this context.

Contributions. In this paper, we take inspiration from recent results in Gaussian Processes (whose covariance matrices share similarities with BIE matrices) to propose an efficient preconditioner for Boundary Integral Equations of the type $\boldsymbol{Ks}=\boldsymbol{b}$ for dense, positive, and symmetric matrices $\boldsymbol{K}$. We explain the roots of our variational multiscale massively-parallel preconditioner which consists in minimizing, for the resulting preconditioned matrix, the so-called Kaporin condition number, a variant of the traditional condition number that better predicts the number of preconditioned Conjugate Gradient iterations necessary to reach convergence. In practice, preconditioning is achieved by computing a sparse inverse-Cholesky factor of $\boldsymbol{K}$ in a massively-parallel way, followed by two sparse matrix-vector products per CG iteration (instead of back-substitutions in traditional incomplete Cholesky preconditioners) which thus involves a marginal overhead compared to the cost of a standard CG iteration — but drastically reducing the number of iterations needed to converge. We show that in various graphics applications, our preconditioner significantly accelerates BIE solves (we demonstrate more than four orders of magnitude for complex and large-scale problems), thus unlocking the scalability issue of the method of fundamental solutions. Finally, we discuss how the link between our contribution and Gaussian Processes provides direct uncertainty quantification of the results.

2 BACKGROUND AND RELATED WORKS

We begin our exposition with a brief review of MFS and BEM, before mentioning some of the practical CG applications that adopted these approaches and discussing the numerical efforts that have been proposed to speed up their solves.

2.1 Continuous roots of MFS/BEM

Suppose we want to find the function u in a domain $\Omega \subset \mathbb{R}^3$ satisfying a partial differential equation (PDE) of the form $\mathcal{L}u(\boldsymbol{x}) = 0$ where $\mathcal{L}$ is a linear operator, for Dirichlet boundary conditions $u(\boldsymbol{y})|_{\mathcal{M}} = b(\boldsymbol{y})$ where $\mathcal{M}$ is a (closed or open) 2-manifold within Ω, and $b(\cdot)$ is a given function over $\mathcal{M}$ that the solution u must match. Further, suppose we know the Green's functions $G(\boldsymbol{x}, \boldsymbol{y})$ of $\mathcal{L}$ representing a solution of the PDE subject to a singularity load at point $\boldsymbol{y}$, i.e., satisfying $\mathcal{L}G(\boldsymbol{x}, \boldsymbol{y}) = \delta(\boldsymbol{x}, \boldsymbol{y})$ where $\delta(\boldsymbol{x}, \boldsymbol{y})$ is the Dirac delta function. With the Green's functions of our linear PDE, we can solve for u via an *integral equation* on $\mathcal{M}$ rather than on the whole domain Ω due to the superposition principle, since the sum of multiple solutions of the PDE is still, itself, a solution. So we seek a density σ of singularity loads over $\mathcal{M}$ such that the integral of all the Green's functions at any point on $\mathcal{M}$ precisely matches the boundary conditions in order to get the correct solution, i.e., one has to find the density σ such that:

$$\int_{\mathcal{M}} G(\boldsymbol{y}, \boldsymbol{z})\sigma(\boldsymbol{z}) \, \mathrm{d}v_{\boldsymbol{z}} = b(\boldsymbol{y}) \quad \forall \boldsymbol{y} \in \mathcal{M}. \tag{1}$$

Once the density satisfying Eq. (1) (called the "boundary integral equation", BIE for short) is found, then the solution u at any point $\boldsymbol{x}$ in Ω can be expressed through a simple evaluation of the infinite sum of Green's functions through:

$$u(\boldsymbol{x}) = \int_{\mathcal{M}} G(\boldsymbol{x}, \boldsymbol{z})\sigma(\boldsymbol{z}) \, \mathrm{d}v_{\boldsymbol{z}}. \tag{2}$$

In other words, we solve a 3D problem via a *two-step approach*, where we need to find the **solution** to a fundamentally 2D problem — the BIE — before proceeding to the **evaluation** of the solution anywhere efficaciously. Notice that we used a Dirichlet problem above, but the same applies to Neumann boundary conditions, leading still to a boundary integral equation and an evaluation equation, but with altered expressions as they will now involve normal derivatives of the Green's function. Consequently, the integral formulations

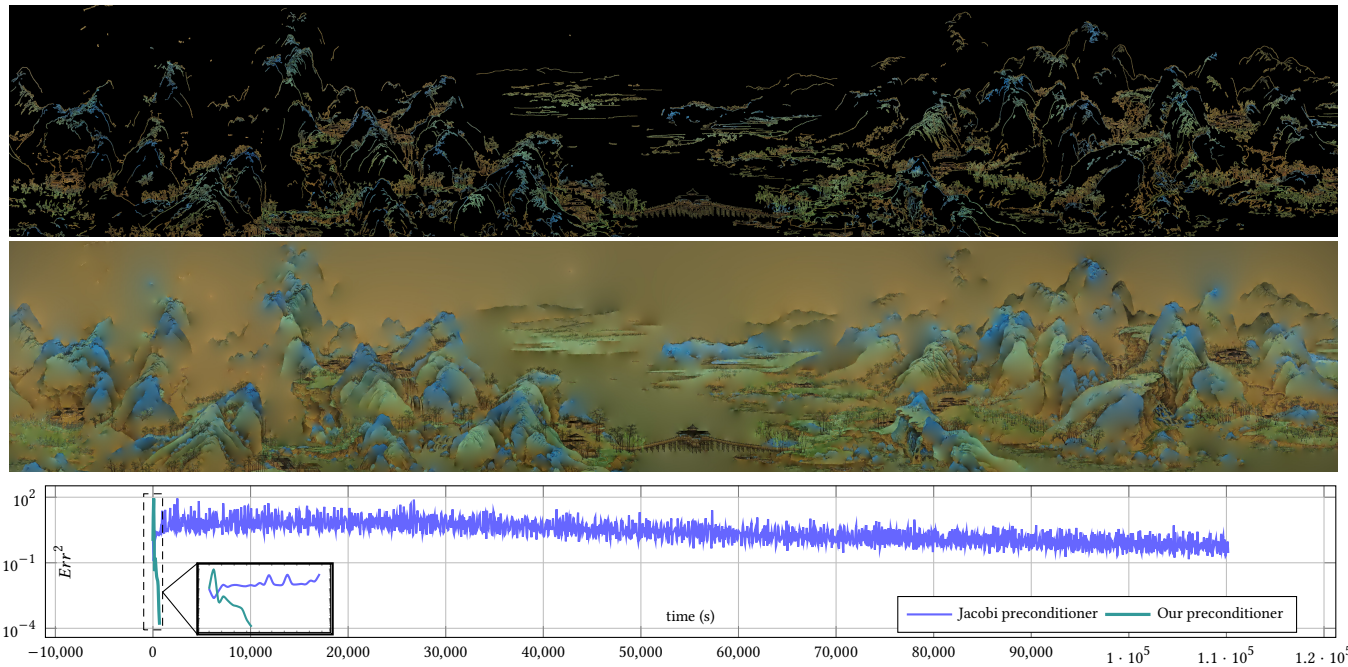


Fig. 2. **Large-scale pixel diffusion.** Here we use $1.35M$ boundary points and $8.4M$ target points, for a final image resolution of 6957×1207. Our algorithm only requires 9GB of memory to store $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ and $\boldsymbol{L}_{\mathcal{S}}$ for a sparsity parameter $\rho=6$. Timings for precomputation, assembly plus factorization for preconditioning, and PCG solve (for three channels) are resp. 112.4s, 11.3s, and 1775.8s, respectively. Only 9 PCG iterations were needed to achieve a relative error of around 0.01 for each color channel. Lastly, FMM takes 1049s to diffuse the colors in the final evaluation. Without our preconditioner, solving such a BIE would be extremely time-consuming and may not be able to even reach such a low error with a conventional preconditioner.

above are useful in many contexts. For instance, in elasticity, if the boundary function b represents the displacement field of the boundary $\mathcal{M}$ of a homogeneous body, the density σ satisfying Eq. (1) will represent the traction at the boundary, which then allows us to evaluate the displacements inside $\mathcal{M}$ via Eq. (2); similarly for electrostatics, where a density σ of point charges along $\mathcal{M}$ matching an input boundary electrostatic potential are enough to reconstruct the full potential field over the whole domain Ω.

2.2 Discretizing the BIE

In practice, we only have **boundary points** $\{\boldsymbol{y}_i\}_{i=1..B}$ on $\mathcal{M}$ which encode the (pointwise sampled or locally integrated) boundary conditions, and a(n often larger) set of **target points** $\{\boldsymbol{x}_i\}_{i=1..T}$ in Ω where the solution needs to be evaluated. For computational purposes, the boundary integrals above have to be discretized, so we also add **source points** $\{\boldsymbol{z}_i\}_{i=1..S}$ carrying a set of values $\{s_i\}_i$ representing the density function σ over $\mathcal{M}$.

Boundary Element Method. The BEM approach to formulate the discrete boundary integral equations assumes that the boundary points are part of a triangle mesh discretizing the surface $\mathcal{M}$, and so are the source points. Using for instance linear basis functions associated with each vertex (say, ϕ_i for $\boldsymbol{y}_i$, ψ_i for $\boldsymbol{z}_i$), a Galerkin finite-element discretization which writes the boundary function as $b(\boldsymbol{y})=\sum_i b_i\phi_i(\boldsymbol{y})$ with $b_i=b(\boldsymbol{y}_i)$ leads to a BIE of the form:

$$\sum_{j=1}^{S}\left(\iint_{\mathcal{M}\times\mathcal{M}}\phi_i(\boldsymbol{y})G(\boldsymbol{y},\boldsymbol{z})\psi_j(\boldsymbol{z})\,\mathrm{d}v_{\boldsymbol{y}}\mathrm{d}v_{\boldsymbol{z}}\right)s_j=\int_{\mathcal{M}}b(\boldsymbol{y})\phi_i(\boldsymbol{y})\mathrm{d}v_{\boldsymbol{y}}\quad\forall i. \tag{3}$$

Once the integrals in Eq. (3) are precomputed through careful quadratures [Duffy 1982], we do get a *linear system* (which we also call the BIE) of size $B\times S$, basically linking the unknown discrete source strengths $\{s_j\}_j$ to the known boundary values $\{b_i\}_i$. Be aware that this short summary of BEM is far from complete as there are multiple variants of BEM depending on the continuity of the solution or its derivatives across $\mathcal{M}$: we only explained the derivation of the "single-layer BEM for Dirichlet problems", but similar approaches can deal with the Neumann case, or even the "double-layer BEM" for Dirichlet or Neumann problems [Costabel 1987]. In our work, one only needs to know that any of these discretizations leads to a dense linear system to solve, involving Green's functions and/or their derivatives.

Method of Fundamental Solutions. The MFS can be easily explained at this point by revisiting the BEM derivation, and taking both the basis functions ϕ_i and ψ_j to be delta Dirac functions in Eq. (3); then the BIE simplifies down to a linear system too:

$$\sum_{j=1}^{S}G(\boldsymbol{y}_i,\boldsymbol{z}_j)\,s_j=b_i.\quad\forall i=1..B \tag{4}$$

Note that in this case, one does *not even need a mesh*: the point samples are enough. This simplicity of derivation has made this approach quite common in CG.

Generic BIE. As we have seen, there are various ways to derive the discrete notion of BIE. Whether the final (discrete) BIE to solve was derived through BEM or MFS, we will always denote it as

$$\boxed{\boldsymbol{K}\boldsymbol{s}=\boldsymbol{b}} \tag{5}$$

where $\boldsymbol{K}$ is a *dense* matrix of size $B \times S$, $\boldsymbol{s}$ is the vector of all sources' strength s_j to solve for, and $\boldsymbol{b}$ is the vector of all the given boundary values b_i. *Our paper focuses on how to solve this BIE efficiently through a few iterations of a Preconditioned Conjugate Gradient algorithm.*

Further Assumptions. Because we aim at applications in CG, we will consider the matrix $\boldsymbol{K}$ to be *symmetric*, as numerical solvers are typically more efficient in this case. This may be seen as a restriction: indeed, in the explanations we provided above, only the MFS-based BIE in Eq. (4) seems symmetric when the Green's functions are symmetric (which is always the case for isotropic cases) and the source samples and boundary samples are one and the same. However, it can be shown that the BEM-Dirichlet approach using single-layer potential from Eq. (3) does also lead to a symmetric BIE matrix when not only source samples and boundary samples coincide, but when their basis functions are the same as well — and the BEM-Neumann approach using double-layer potential which involves the second derivatives $\partial^2 G(\boldsymbol{x}, \boldsymbol{y}) / \partial n(\boldsymbol{x}) \partial n(\boldsymbol{y})$ of Green's functions, also results in a symmetric matrix in this case. For all other cases, $\boldsymbol{K}$ may not be symmetric, but one can still use a least-squares solution by solving $(\boldsymbol{K}^\mathsf{T}\boldsymbol{K})\boldsymbol{s} = (\boldsymbol{K}^\mathsf{T}\boldsymbol{b})$; so our generic Eq. (5) remains valid, where now $\boldsymbol{K}$ possibly comes from an inner product of Green's functions. We will thus assume in the remainder of this paper that boundary and source points are the same ($\boldsymbol{y}_i \equiv \boldsymbol{z}_i$), and that our matrix $\boldsymbol{K}$ is of size $B \times B$.

Since $\boldsymbol{K}$ derives from BEM/MFS, it will also typically be *positive-semidefinite* (PSD). Even if it is not the case when using, for instance, the Green's functions of the Helmholtz equation (which we will cover in Sec. 4.4), its least-squares version $\boldsymbol{K}^\mathsf{T}\boldsymbol{K}$ will be PSD.

Additionally, a typical issue with Green's functions $G(\boldsymbol{x}, \boldsymbol{y})$ is its singularity when $\boldsymbol{x} = \boldsymbol{y}$: the diagonal of matrix $\boldsymbol{K}$ may therefore have undefined terms involving $G(\boldsymbol{y}_i, \boldsymbol{y}_i)$. A typical approach to deal with this issue is to "regularize" the Green's functions. This is achieved by either finding a singularity-free function $G^\varepsilon(\cdot, \cdot)$ such that $\mathcal{L}G^\varepsilon$ is a smoothed-out Dirac functions [Cortez 2001], or by defining G^ε to match G almost everywhere except in a small disk of radius $\varepsilon \ll 1$ around the singularity to remove it. Another known approach is to avoid regularizing by slightly offsetting the source points so that they are not exactly on top of the boundary points; but this common approach renders the matrix asymmetric, hence requiring a least-squares solve or an ad-hoc solver. So we will assume that regularized Green's functions G^ε are used in this paper.

2.3 Evaluating the solution

In our discrete setup, once the source values $\{s_i\}_i$ are solved from the BIE (Eq. (5)), we can finally evaluate the solution values at each target point $\boldsymbol{x}_k$, through a simple matrix-vector multiplication basically: for MFS, one simply evaluates

$$u(\boldsymbol{x}_k) = \sum_{j=1}^{S} G(\boldsymbol{x}_k, \boldsymbol{y}_j)\, s_j \quad \forall k = 1..T. \tag{6}$$

This is just an evaluation, not a solve, but its computation can still be quite costly as, typically, none of the $G(\boldsymbol{x}_k, \boldsymbol{y}_j)$ terms are zero. However, due to the usual fast decay of Green's functions, this evaluation can be achieved very efficiently via the Fast Multipole Method (FMM) algorithm [Greengard and Rokhlin 1987], which drastically reduces the complexity of this large summation via an adaptive evaluation. One can also leverage an algebraic version of FMM in which the matrix $\boldsymbol{K}$ is partitioned, then approximated via a hierarchical matrix (or $\mathcal{H}$-matrix [Hackbusch 1999; Hackbusch and Khoromskij 2000]) via blockwise low-rank submatrices so as to accelerate matrix-vector multiplications.

2.4 Related Works in CG

Many graphics papers have leveraged the dimensionality reduction in the size of the variable count (and thus, of the matrix solve) that BEM, MFS, or even just Green's functions bring. In the case of animation, fast approaches to simulate deformable bodies [James and Pai 1999; Sugimoto et al. 2022], fluid [Da et al. 2016], or even ferrofluids [Huang and Michels 2020] have leveraged BEM on triangle meshes to accelerate computations. MFS was exploited in the case of wave animation [Schreck et al. 2019], polyhedral finite elements for elasticity [Martin et al. 2008], acoustics [James et al. 2006], as well as various pointset reconstruction methods [Carr et al. 2001; Zhong et al. 2019], while Green's functions (and MFS as well if boundary constraints are added) were key to the interactive sculpting work of De Goes and James [2017]. Various linear solvers were used to solve the BIE, typically Singular Value Decomposition (see, e.g., [Schreck et al. 2019]) or GMRES (see, e.g., [Bang et al. 2023]), while recent works have all adopted some form of FMM-based evaluation technique due to its broad applicability and efficacy (see, e.g., [Zhong et al. 2019; Bang et al. 2023]). However, we note that most resulting BIEs are *overconstrained*, as authors prefer to reduce the number of sources (and thus, only *approximate* the boundary conditions instead of fitting them exactly) so as to offer a faster solve: indeed, if a FMM evaluation is used, *the remaining bottleneck to a wider use of MFS/BEM methods is the BIE solve*, which is currently prohibitively expensive for very large matrices (i.e., of cubic complexity for direct solvers). Our work shows that having boundary and source points to be coinciding can be actually more efficient than just skimping on source points *if* the underlying structure of the matrix $\boldsymbol{K}$ (which is, in a sense, the inverse of the sparse linear matrix representing the linear operator of the PDE) is properly exploited. And while recent works never use more than 16,000 sources to guarantee acceptable timings, we will show how to scale a BIE solve up to millions of degrees of freedom.

2.5 Related works in preconditioning

Relative to the amount of work on preconditioning large sparse linear systems, there has been only a few contributions which tried to precondition BIE linear systems (i.e., involving dense, symmetric, and positive semi-definite matrices). Early works include [Steinbach and Wendland 1998] based on basis transformation and [Schippers 1985] which proposed to *customize* multigrid-based preconditioning methods for a few specific applications to efficiently solve BIE-like linear systems based on a case-by-case decomposition of the matrix $\boldsymbol{K}$ into two matrices and the use of quadrature evaluations to derive a relaxation scheme making the multigrid preconditioner efficient. When an $\mathcal{H}$-matrix is used for fast matrix-vector multiplication, preconditioning of the BIE matrix can be achieved more efficiently through LU factorization [Kriemann 2013] or through a nested GMRES-based construction [Amlani et al. 2019], with performance

improvements reported to be around a factor two; however, the precision of low-rank approximations used in an $\mathcal{H}$-matrix can notably limit efficiency, while pursing higher accuracy weakens the benefits of hierarchical matrices — thus requiring parameter tuning to achieve good results. Another BIE preconditioner was proposed in [Beatson et al. 1999], especially designed for RBF fitting.

Our approach, however, is general and can be applied to most PDEs (e.g., elliptic or parabolic, but even for some non-local partial derivative equations, as long as the Green's function decays rapidly). Our numerical method takes advantage of recent developments in Gaussian processes (GP), where the covariance matrices used in GP statistics are, in fact, the equivalent of the Green-derived matrices in our context. Early work by Vecchia [1988] approximated the likelihood function of a Gaussian distribution via a product of univariate conditional densities, each depending on a subset of the previously ordered variables. This was later observed to be equivalent to computing an approximate inverse-Cholesky factor of the covariance matrix [Katzfuss and Guinness 2021]; independently, Kaporin [1994] derived a closed-form expression of the approximate inverse-Cholesky preconditioner of a sparse matrix that minimizes, subject to sparsity constraints, a particular notion of condition number of the preconditioned system. Kaporin's approach turned out to be equivalent to minimizing a Frobenius norm objective under diagonal scaling constraint [Yeremin et al. 2000]. Schäfer et al. [2021a] then showed that Kaporin's preconditioner and the Vecchia approximation are equivalent and can be obtained by sparsity-constrained Kullback-Leibler (KL) minimization as well; they also proposed a reordering of rows and columns of the Green's functions and a sparsity pattern to create an end-to-end solver deducing the target values from boundary values with a provably log-linear complexity. In this paper, we leverage Kaporin's variational definition of the inverse-Cholesky factor but applied to dense matrices, and introduce a massively-parallel implementation of a *BIE preconditioner*. We show on various graphics applications that timings can be easily improved by one to four orders of magnitude, unlocking the potential of MFS/BEM to efficiently handle very large problems.

2.6 Overview

The remainder of this paper focuses on solving the Boundary Integral Equation (5) efficiently. After quickly reviewing the closed-form sparsity-constrained minimizer of Kaporin condition number on which our approach is based, we will detail the efficient construction of our preconditioner for $\boldsymbol{K}$, written as the product of a sparse approximation of the inverse-Cholesky factor and its transpose, so that a few iterations of Preconditioned Conjugate Gradient suffice in practice to solve the BIE (see Fig. 3). We will then show several applications of our fast BIE solver, demonstrating how its massively-parallel nature brings tremendous speedups in practice.

3 VARIATIONAL MULTISCALE PRECONDITIONER

We now present in detail how to compute, given a lower-triangular sparsity pattern $\mathcal{S}$, a sparse inverse-Cholesky factor $\boldsymbol{L}_\mathcal{S}$ of the $B\times B$ matrix $\boldsymbol{K}$ to approximate $\boldsymbol{K}^{-1}$: the resulting approximate factor $\boldsymbol{L}_\mathcal{S}$ can be used to form a low-cost, yet efficient preconditioner $\boldsymbol{L}_\mathcal{S}\boldsymbol{L}_\mathcal{S}^\mathsf{T}$ to greatly accelerate the convergence of PCG when solving Eq. (5).

3.1 Kaporin's variational inverse-Cholesky factor

While the real lower-triangular inverse-Cholesky factor $\boldsymbol{L}$ such that $\boldsymbol{L}\boldsymbol{L}^\mathsf{T}=\boldsymbol{K}^{-1}$ is slow to evaluate, one can always, for efficiency, look for an *incomplete* inverse-Cholesky factor $\boldsymbol{L}_\mathcal{S}$ which is a sparse approximation of $\boldsymbol{L}$ for a given choice of sparsity pattern $\mathcal{S}:=\{(i,j)\mid i\geq j \text{ and } (\boldsymbol{L}_\mathcal{S})_{ij}\neq 0\}$. A straightforward approach to find such a matrix $\boldsymbol{L}_\mathcal{S}$ to best approximate $\boldsymbol{K}^{-1}$ is to minimize $\|\boldsymbol{I}-\boldsymbol{L}_\mathcal{S}\boldsymbol{L}_\mathcal{S}^\mathsf{T}\boldsymbol{K}\|_F$ subject to the sparsity pattern $\mathcal{S}$, as this would provide an arguably optimal preconditioning of matrix $\boldsymbol{K}$. However, this Frobenius-norm minimization problem requires nonlinear solves that are more computationally intensive than the initial linear system we intend to solve for large scale problems.

Kaporin [1994] found a simple, closed-form solution of a different optimization problem which happens to be very useful in our context as it can be implemented in a massively-parallel fashion. First, he proved that what is now known as the Kaporin condition number κ_{Kap} of a matrix — that is, the *ratio between the algebraic and geometric means of the eigenvalues* of a matrix instead of the standard condition number κ evaluated as the ratio between its maximum and minimum eigenvalues — leads to a *tighter bound* than κ on the estimate of the iteration count needed by PCG to reach convergence within a given tolerance ϵ. Second, he proved that the matrix $\boldsymbol{L}_\mathcal{S}$ subject to a given sparsity pattern which minimizes the Kaporin condition number $\kappa_{\text{Kap}}(\boldsymbol{L}_\mathcal{S}\boldsymbol{L}_\mathcal{S}^\mathsf{T}\boldsymbol{K})$ can be expressed column-by-column in closed-form using inverses of small sub-blocks of matrix $\boldsymbol{K}$: if we denote the sparsity pattern of a column j of $\boldsymbol{L}_\mathcal{S}$ as $\mathcal{S}_j:=\{i\mid (i,j)\in\mathcal{S}\}$ (i.e., the row indices of the non-zero elements contained in the column j of a matrix with sparsity $\mathcal{S}$), then the j^{th} column $\boldsymbol{L}_{\mathcal{S}_j,j}$ of $\boldsymbol{L}_\mathcal{S}$ is expressed independently from the other columns as:

$$\boldsymbol{L}_{\mathcal{S}_j,j}=\frac{\boldsymbol{K}^{-1}_{\mathcal{S}_j,\mathcal{S}_j}\mathbb{e}_j}{\sqrt{\mathbb{e}_j^\mathsf{T}\boldsymbol{K}^{-1}_{\mathcal{S}_j,\mathcal{S}_j}\mathbb{e}_j}},\quad \forall j=1..B, \tag{7}$$

where $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ is the submatrix of $\boldsymbol{K}$ with row and column indices in $\mathcal{S}_j$, and $\mathbb{e}_j=(1,0,\ldots,0)^\mathsf{T}\in\mathbb{R}^{|\mathcal{S}_j|}$ is a unit vector of length $|\mathcal{S}_j|$. This Kaporin construction of $\boldsymbol{L}_\mathcal{S}$, which happens to achieve Vecchia's approximation [Vecchia 1988] and which has since then found two other variational interpretations (see Appendix A for details), thus provides an optimal preconditioner for PCG for a given sparsity, with the potential to drastically accelerate the solve of Eq. (5) if a good balance between sparsity and computational time to evaluate the inverse-Cholesky factor $\boldsymbol{L}_\mathcal{S}$ is found: too sparse a constrained pattern may be very fast to evaluate but may not end up conditioning the matrix $\boldsymbol{K}^{-1}$ sufficiently well to guarantee PCG convergence in a few iterations; better conditioning may come at a price of a far reduced sparsity, which will take longer to evaluate.

Note that Kaporin's work was mostly applied to inverting *sparse* matrices; since these matrices typically have dense inverses and inverse-Cholesky factors, the strength of this preconditioning was fundamentally limited. Recently, Schäfer et al. [2021a] showed that it can be used, instead, to approximate the sparse inverse-Cholesky factors of dense Green's matrices in log-linear cost, making it a promising approach for their inversion.

The remainder of this section tackles the evaluation of $\boldsymbol{L}_\mathcal{S}$ whose columns are expressed in Eq. (7). We will see that we can render this evaluation massively-parallel, since each column (or groups of

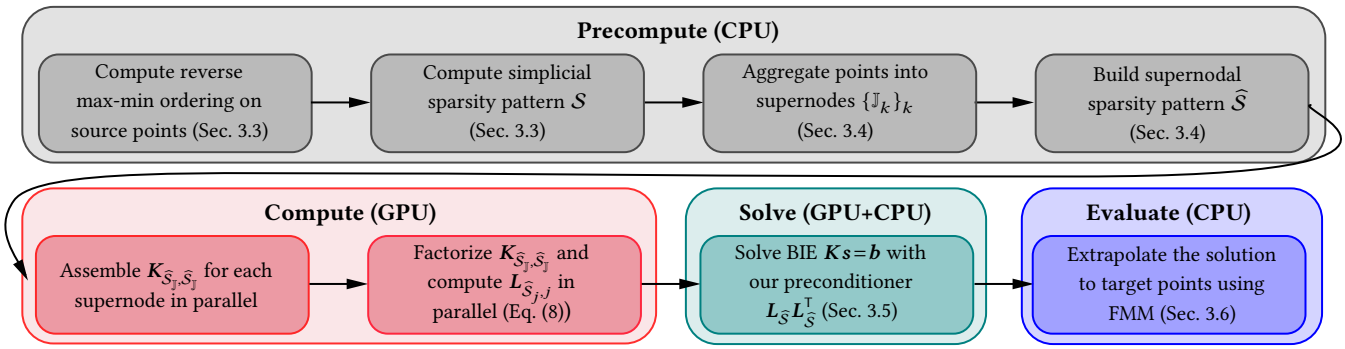


Fig. 3. **Overview.** In order to solve a generic boundary integral equation like Eq. (5), the construction of our approximate Cholesky factor $\boldsymbol{L}_{\mathcal{S}}$ for $\boldsymbol{K}^{-1}$ consists of a few CPU-based precomputations (such as constructing the ordering and the sparsity pattern), followed by the assembly on the GPU of submatrices of $\boldsymbol{K}$ to construct the factor $\boldsymbol{L}_{\mathcal{S}}$ in a massively-parallel manner; then $\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}$ is used as a preconditioner which allows for a Preconditioned Conjugate Gradient algorithm to converge in a few iterations; finally a FMM-based evaluation on a series of target 3D points is performed where the solution needs to be evaluated.

columns) can be evaluated independently and that the sub-blocks of $\boldsymbol{K}$ requiring inversions can be performed, themselves, in parallel. Notice also that only few elements of $\boldsymbol{K}$ are needed (those used in the relevant sub-blocks), so our construction will *not even require the full assembly of the matrix* $\boldsymbol{K}$, thus further saving time and memory by skipping many unnecessary evaluations of Green's functions.

3.2 Local Cholesky factorization for $\boldsymbol{L}_{\mathcal{S}_j,j}$

Each column of $\boldsymbol{L}_{\mathcal{S}}$ can be processed in parallel through Eq. (7) via a small, dense linear solve. Since $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ is symmetric and positive definite, we can factorize $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ using a classical Cholesky decomposition followed by two back-substitution to obtain $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j$. However, given the sparse structure of vector $\mathbb{e}_j$, the two back-substitutions can be replaced by one if one uses the *reverse* Cholesky factorization of $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$. More specifically, after assembling $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ based on the column sparsity $\mathcal{S}_j$, we can compute the decomposition $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}=\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{\mathsf{T}}$, where $\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}$ is an *upper* triangular matrix. We then tranform Eq. (7) into

$$\boxed{\boldsymbol{L}_{\mathcal{S}_j,j}=\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{-\mathsf{T}}\mathbb{e}_j,\quad \forall j=1..B} \tag{8}$$

to compute the j^{th} column of $\boldsymbol{L}_{\mathcal{S}}$. We note that computing this reverse decomposition does not incur extra cost: if $\text{chol}(\cdot)$ denotes the standard lower-triangular Cholesky factor, $\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}$ is found by

$$\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}=\boldsymbol{R}_j\,\text{chol}(\boldsymbol{R}_j\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{R}_j)\boldsymbol{R}_j, \tag{9}$$

where $\boldsymbol{R}_j$ is the *permutation matrix* of size $|\mathcal{S}_j|\times|\mathcal{S}_j|$ which reverses the indices from $1,2,...,|\mathcal{S}_j|$ to $|\mathcal{S}_j|,...,2,1$. Not only using Eq. (8) in lieu of Eq. (7) simplifies computations, but we will see in Sec. 3.4 that it also enables the reuse of the factor to compute other columns $\boldsymbol{L}_{\mathcal{S}_j,j}$ aggregated into a same supernode because the bottom-right $k\times k$ sub-matrix of $\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}$ (with $k<|\mathcal{S}_j|$) is also the reverse Cholesky factor of the bottom-right $k\times k$ sub-matrix of $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ for all j.

3.3 Ordering and sparsity pattern

For preconditioners based on incomplete matrix factorization, the ordering of the degrees of freedom (i.e., the orders of rows/column) and the choice of sparsity pattern play a crucial role on the quality of the results. In this work, we adopt the *reverse max-min ordering* $\boldsymbol{P}$

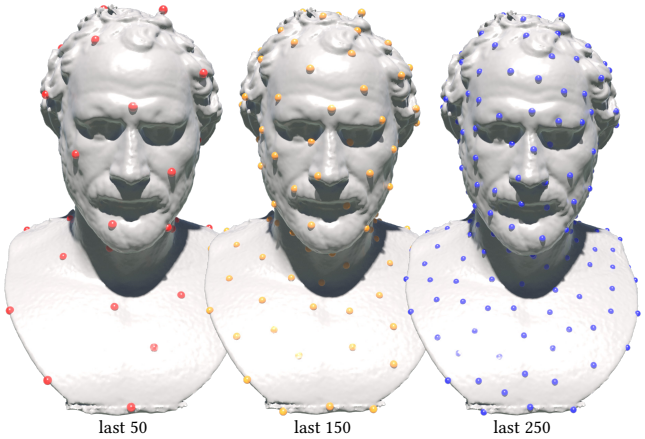


Fig. 4. **Reverse max-min ordering:** Geometrically, the reverse max-min order decomposes all DoFs (here, vertices of a mesh) into different spatial scales, with coarse-scale points (capturing low frequencies) at the end, and fine-scale points (handling high-frequency details) at the beginning.

to permute $\boldsymbol{K}$ and then compute $(\boldsymbol{P}^{\mathsf{T}}\boldsymbol{K}\boldsymbol{P})^{-1}=\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}$, as was proposed for ill-conditioned differential operators and Gaussian process regression in [Guinness 2018; Chen et al. 2021b; Schäfer et al. 2021a,b]. Generating the max-min ordering is implemented through farthest point sampling: starting from an arbitrary boundary point $\boldsymbol{y}_{i_0}$, we repeatedly pick the next boundary point in the ordering to be the farthest one from already-selected ones thus far, yielding a series of ordered indices as

$$i_k=\underset{q}{\text{argmax}}\ \min_{p\in\{0,k-1\}}\text{dist}(\boldsymbol{y}_q,\boldsymbol{y}_{i_p}), \tag{10}$$

where $\text{dist}(\cdot,\cdot)$ is the Euclidean distance. We then reverse this max-min ordering into $\boldsymbol{P}=\{i_{B-1},...,i_1,i_0\}$ to permute all DoFs. Note that while a brute force computation of the max-min ordering would be in $O(B^2)$, we instead apply the algorithm from [Schäfer et al. 2021b, Alg. 4.1] implemented in GPVecchia library [Zilber and Katzfuss 2021], which is in $O(B\log^2(B))$.

Geometrically, the reverse max-min reordering implies a multiresolution order of the boundary samples on $\mathcal{M}$ (see Fig. 4), where "coarse-scale" boundary points (appearing later in the ordering) are

first spread out over the domain, before "fine-scale" points (appearing earlier in the ordering) come in to fill in the voids (see [Chen et al. 2021b]), establishing a notion of length scale ℓ on all the points defined through $\ell_{i_k} = \min_{p\in\{0,k-1\}} \text{dist}(\boldsymbol{y}_{i_k}, \boldsymbol{y}_{i_p})$ which is monotonically increasing in the reverse max-min ordering. Since the k-th column of the inverse-Cholesky factors of a covariance matrix encodes conditional correlations of the k-th variable of the Gaussian process with those later in the ordering [Katzfuss and Guinness 2021; Schäfer 2021], the reverse max-min ordering ensures that the spatial locations associated with the k-th variable and its successors are roughly equally distributed in space. In this setting, the Green's functions of elliptic PDEs are known to be subject to the *screening effect* illustrated in Fig. 5, whereby conditioning on just a subset of the points leads to near-independence with more distant points, as long observed in the spatial statistics literature [Stein 2002]. Rigorous bounds on screening, implying exponential decay of the inverse Cholesky factors, were derived in [Schäfer et al. 2021a,b] based on prior work on operator-adapted wavelets and numerical homogenization (see [Owhadi and Scovel 2019; Altmann et al. 2021] for an overview on these topics). This screening effect thus motivates our choice of sparsity pattern: we construct $\mathcal{S}$ based on the length scale of each point, as was advocated for the preconditioning of differential operators in [Chen et al. 2021b], with the non-zero entries in the inverse-Cholesky factor specified as

$$\mathcal{S} := \left\{(i,j) | i \geq j \text{ and } \text{dist}(\boldsymbol{y}_i, \boldsymbol{y}_j) \leq \rho \min(\ell_i, \ell_j)\right\}, \qquad (11)$$

meaning that an element (i, j) in the sparsity is forced to be non-zero if $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are within each other's support radius scaled by ρ, a parameter which allows a tradeoff between preconditioning quality and computational cost. In practice, sparsity computation can be done in parallel for each column through a fast range search supported by k-d trees [Chen et al. 2021b]. Note that we refer to Eq. (11) as the *simplicial sparsity* to differentiate it from the supernodal sparsity which we will introduce next.

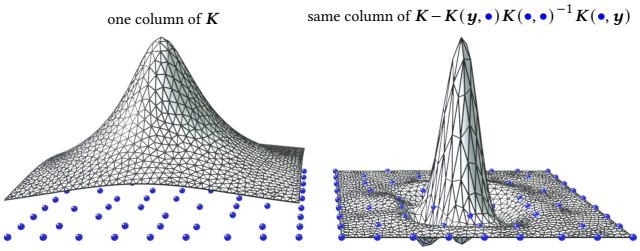


Fig. 5. **Screening effect.** The covariance matrix $\boldsymbol{K}$ features long-range correlations but the covariance conditioned on coarse (blue) samples given by its Schur complement is highly localized. Inverse Cholesky factors of $\boldsymbol{K}$ encode conditional correlations, thus screening causes their near-sparsity.

3.4 Aggregated factorization

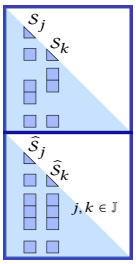
So far, we have seen how each column of $\boldsymbol{L}_{\mathcal{S}}$ only requires the assembly of small matrices $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ based on our chosen sparsity, which saves time and memory compared to assembling the whole matrix $\boldsymbol{K}$. However, notice that multiple columns of the reverse-Cholesky factors may require nearly the same dense sub-matrix $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$, hence leading to redundant computations in their factorizations. Reusing $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ and its Cholesky factorization $\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{U}^{\mathsf{T}}_{\mathcal{S}_j,\mathcal{S}_j}$ can be achieved by aggregating columns requiring nearly the same factorization into a *supernode* [Stein et al. 2004; Ferronato et al. 2015; Guinness 2018] to remove obvious duplicated evaluations: the factorization of an inclusive sub-matrix of $\boldsymbol{K}$ will thus serve all the columns of the supernode, since each column can extract the needed rows and columns from the factor.

Algorithm 1: Identifying supernodes

Data: Simplicial sparsity pattern $\mathcal{S}$, length scales $\{\ell_j\}_j$
Result: A set of supernodes $\{\mathbb{J}_k\}_k$

```
for j ← 1 to B do
    processed[j] ← false;
k ← 1;
for j ← 1 to B do
    if processed[j] then
        continue;
    J_k ← {};            // initialize a new supernode
    for i ∈ S_{ρ,j} do
        if !processed[i] and ℓ_i ≤ (3/2) ℓ_j ;  // nearby scales
        then
            processed[i] ← true;
            J_k ← J_k ∪ {i}
    k ← k + 1;
```

Aggregating columns into supernodes is done by merging spatially-close indices of similar scales (see pseudo-code in Alg.1) as they are more likely to share similar indices for their sub-matrix of $\boldsymbol{K}$. When the algorithm exits, it returns a set of supernodes $\{\mathbb{J}_k\}_k$, and each of them consists of certain number of indices which are not necessarily consecutive in the reverse max-min ordering — but all indices in the same supernode will share (part of) the same matrix extracted from $\boldsymbol{K}$: the supernode will perform a factorization of the submatrix of $\boldsymbol{K}$ with selected indices derived from the extended supernodal sparsity pattern $\widehat{\mathcal{S}}$. Simply speaking, the sparsity of a column $j \in \mathbb{J}$ is the union of all non-zero entries collected from all columns belonging to $\mathbb{J}$; that is, its sparsity is $\widehat{\mathcal{S}}_j$ is defined through (see inset)



$$\widehat{\mathcal{S}}_j := \{i \mid i \geq j \text{ and } \exists k \in \mathbb{J}, (i,k) \in \mathcal{S}\}, \forall j \in \mathbb{J}. \qquad (12)$$

We can then define the sparsity pattern of a supernode through:

$$\widehat{\mathcal{S}}_{\mathbb{J}} = \bigcup_{j\in\mathbb{J}} \widehat{\mathcal{S}}_j.$$

Fig. 6 illustrates the supernodal aggregation and the resulting supernodal sparsity pattern. We denote the submatrix of $\boldsymbol{K}$ within a supernode as $\boldsymbol{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}$, and its reverse Cholesky factor is reused for computing all columns $\boldsymbol{L}_{\widehat{\mathcal{S}}_{\mathbb{J}},j}, \forall j \in \mathbb{J}$. A direct implementation of the supernodal approach is given in Alg. 2, where each of the three steps can be massively parallelized. As Fig. 7 demonstrates, computing the inverse Cholesky factor scales almost linearly in problem size, and so do the memory cost to store the local matrix $\boldsymbol{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}$, and thus the global inverse Cholesky factor $\boldsymbol{L}_{\widehat{\mathcal{S}}}$.

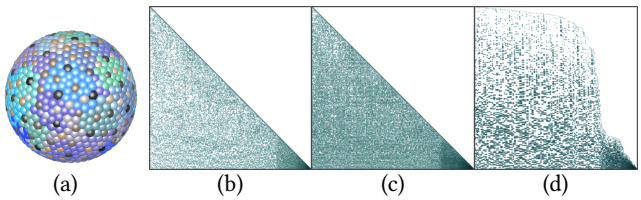
3.5 Preconditioning a BIE matrix $\boldsymbol{K}$

Now that we have a fast algorithm to evaluate $\boldsymbol{L}_{\mathcal{S}}$ for a given sparsity parameter ρ, and knowing that $\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}$ approximate $\boldsymbol{K}^{-1}$, we may be tempted to simply compute the solution to Eq. (5) via $\boldsymbol{b} = \boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{s}$, which is trivial to evaluate through two backsubstitutions. However, the accuracy of this direct solve may not good enough for a small ρ, while increasing ρ to get a denser, more accurate factorization leads to a quick growth in computational and memory costs (see Fig. 16 for an example). However, we saw that the Kaporin solution we computed is optimal in terms of the resulting condition number $\kappa_{\text{Kap}}(\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{K})$. So we can instead use $\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}$ as a preconditioner on a conjugate gradient solver to guarantee better efficiency in solving $\boldsymbol{K}\boldsymbol{s}=\boldsymbol{b}$. Moreover, since the preconditioner only needs to be applied to the residual $\boldsymbol{e}_k = \boldsymbol{K}\boldsymbol{s}_k - \boldsymbol{b}$ at iteration k of the conjugate gradient, we only need to apply two low-cost sparse matrix-vector products to evaluate $\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{e}_k$, hence the overhead spent on conditioning the linear system is quite limited. (Note that this is sharp contrast with the incomplete Cholesky preconditioner for differential operators from [Chen et al. 2021b], where two back-substitutions were needed to perform preconditioning.) In practice, we can use a relatively small ρ to already drastically reduce the amount of iterations: in Fig. 8, a value $\rho=6$ for a system of around $260K$ degrees of freedom leads to PCG convergence in exactly 7 iterations for relative errors below 0.03, compared to several hundreds of iterations for a simple Jacobi preconditioner.

Algorithm 2: Supernodal approach for $\boldsymbol{L}_{\widehat{\mathcal{S}}}$

Data: Supernodal sparsity patterns $\{\widehat{\mathcal{S}}_{\mathbb{J}}\}_{\mathbb{J}}$, Green's function $G(\cdot,\cdot)$, source points $\{\boldsymbol{z}_i\}_i$.
Result: Inverse Cholesky factor $\boldsymbol{L}_{\widehat{\mathcal{S}}}$ such that $\boldsymbol{K}^{-1} \approx \boldsymbol{L}_{\widehat{\mathcal{S}}}\boldsymbol{L}_{\widehat{\mathcal{S}}}^{\mathsf{T}}$

Function `AssembleLocalMFSMatrix()`:
  for each *supernode* $\mathbb{J}$ **do**
    $n_{\mathbb{J}} \leftarrow |\widehat{\mathcal{S}}_{\mathbb{J}}|$;
    $\boldsymbol{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}} \leftarrow \boldsymbol{0}_{n_{\mathbb{J}}\times n_{\mathbb{J}}}$;
    for $i\in\mathbb{J}$, $j\in\mathbb{J}$ **do**
      $\boldsymbol{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}[i,j] \leftarrow G(\boldsymbol{z}_i,\boldsymbol{z}_j)$;
Function `LocalCholeskyFactorize()`:
  for each *supernode* $\mathbb{J}$ **do**
    compute reverse Cholesky decomposition $\boldsymbol{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}} = \boldsymbol{U}_{\mathbb{J}}\boldsymbol{U}_{\mathbb{J}}^{\mathsf{T}}$;
Function `ComputeGlobalFactor()`:
  for $j \leftarrow 1$ **to** B **do**
    Find $\mathbb{J}$ such that $j\in\mathbb{J}$;
    $n_j \leftarrow |\widehat{\mathcal{S}}_j|$;
    $\boldsymbol{L}_{\widehat{\mathcal{S}}_j,j} \leftarrow (\boldsymbol{U}_{\mathbb{J}}[\text{-}n_j\text{:},\text{-}n_j\text{:}])^{-\mathsf{T}}\boldsymbol{e}_j$; `// python notation`



(a) (b) (c) (d)

Fig. 6. **Aggregated sparsity pattern.** The supernode clusters (each consisting of points with the same color) are shown on a sphere model with 1275 points (a), along with the simplicial sparsity pattern with $\rho=3$ (b) and the supernodal sparsity pattern (c). Reordered supernodal pattern (d) by placing columns within a supernode consecutively (for visualization purposes only).

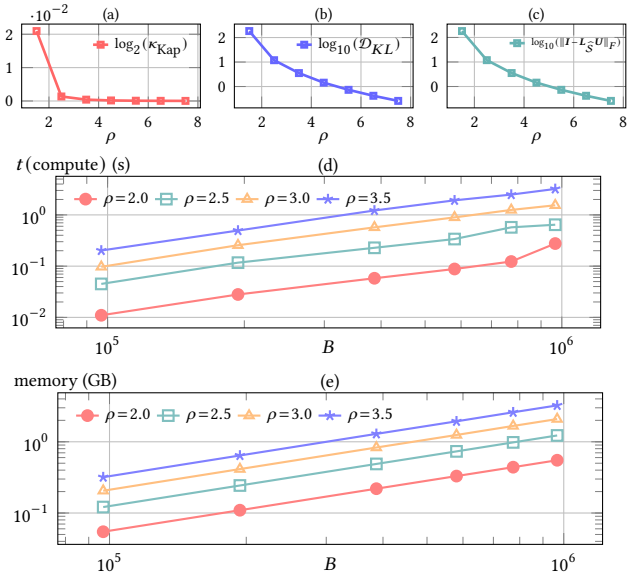


Fig. 7. **Asymptotic behaviors.** We test our algorithm in 3D for boundary points that are uniformly distributed over a square. In (a), (b) and (c), the three variational functionals presented in Appendix A can be seen decreasing as the sparsity parameter ρ increases (since the inverse-Cholesky factor gets less sparse) — and in particular, the Kaporin condition number drops very quickly. In (d) and (e), a quasi-linear growth in the number of boundary values B for both memory costs and time costs is witnessed for our GPU-based inverse-Cholesky preconditioner computation over a variety of sparsity levels, confirming the scalability of our construction.

3.6 Black-box FMM for matrix-vector product

Although our preconditioner can be constructed and applied efficiently, the run-time performance of PCG would be significantly slowed down for large problems if a full evaluation of the product $\boldsymbol{K}\,\boldsymbol{s}$ needs to be performed. However, there exists a large volume of work to speedup this dense matrix-vector product. Here, we exploit the Fast Multipole Method, already mentioned in Sec. 2, to accelerate dense matrix-vector products in PCG. We adopt the black-box fast multipole method implementation of [Wang et al. 2021] which only requires to provide a routine to evaluate the regularized Green's function G^{ε} between two points $\boldsymbol{x}$ and $\boldsymbol{y}$, without having to manually implement FMM operators such as multipole-to-local, multipole-to-multipole, etc. Their code constructs a low-rank approximation of Green's functions using Chebyshev polynomials for non-oscillatory kernels, further compressed through SVD to reduce the complexity of the matrix-vector product from quadratic to linear. Thus, it is well suited to our case for a range of Green's functions in both 2D and 3D. Note that this same FMM procedure (or alternatively, an $\mathcal{H}$-matrix) can be used for interpolating the solution to the target points via Eq. (6).

4 APPLICATIONS AND RESULTS

In this section, we first discuss about a few implementation details before testing our method on various CG-related applications: we show how our preconditioner significantly boosts the performance of solving Laplace's equations, linear elasticity and Helmholtz equations with boundary conditions. Finally, the link between MFS and Gaussian Processes is explored, enabling uncertainty quantification.

4.1 Implementation

Given boundary points $\{\boldsymbol{y}_i\}_i$ and target points $\{\boldsymbol{x}_k\}_k$, we rescale them so that their bounding box is unit. We perform a number of precomputations directly on the CPU, including computing the max-min order, identifying the supernodes, and constructing the sparsity patterns ($\mathcal{S}$ and its associated column sparsities $\mathcal{S}_j$). We start the column-wise construction of the sparse inverse-Cholesky factor and its assembly of local matrices $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ along with their factorizations entirely on GPU to leverage the massive parallelism they offer. During PCG iterations, we combine NVIDIA® cuSPARSE and cuBLAS for optimized linear algebra, and the CPU-based black-box FMM from [Wang et al. 2021] to accelerate matrix-vector products. The source code of our implementation is made available at *https://gitlab.inria.fr/geomerix/public/mfs-chol*.

All examples shown in this paper were run on a laptop (AMD Ryzen 7 5800H CPU with 8 cores and 16G RAM) with a NVIDIA GeForce RTX 3060 Laptop GPU with 6GB of RAM to prove scalability — except for Fig. 2 where an NVIDIA RTX A6000 GPU with 48GB of RAM was employed instead to handle our largest matrix size. Because all the Green's functions we use are isotropic, $G(\boldsymbol{x},\boldsymbol{y})$ is always expressed as a function of $r=\|\boldsymbol{x}-\boldsymbol{y}\|$. We thus use a simple regularization approach consisting in changing r in their expression to $r_\varepsilon=\sqrt{r^2+\varepsilon^2}$, with $\varepsilon=10^{-5}$ (any value in the range $[10^{-6},10^{-4}]$ provides visually similar results) — but other regularizations can be used. The only parameter in our approach is ρ to construct the sparsity pattern $\mathcal{S}$. Since we are dealing with boundary points sampling a hypersurface, we found that for 2D problems, ρ should be set relatively large (e.g., around 8), while 3D problems should use a smaller ρ (e.g., around 5) to offer a good balance between computational cost and preconditioning quality. Note that the resulting quality of our preconditioner depends not only ρ, but also on the spatial distribution of boundary points: if very dense regions of boundary points are present, ρ can be mildly decreased. Finally, we use the relative error $Err=\|\boldsymbol{K}\boldsymbol{s}-\boldsymbol{b}\|/\|\boldsymbol{b}\|$ to measure our solver's accuracy.

4.2 Laplace's equation

Laplace's equation $\Delta u=0$ with imposed boundary conditions has been a staple of CG applications. Functions satisfying this PDE are called harmonic, and the Green's function of the Laplace operator is

$$G(\boldsymbol{x},\boldsymbol{y})=\begin{cases}-\dfrac{1}{2\pi}\ln(r), & \text{in 2D}\\[2ex] \dfrac{1}{4\pi r}, & \text{in 3D}\end{cases} \tag{13}$$

where $r=\|\boldsymbol{x}-\boldsymbol{y}\|$. We demonstrate the effectiveness of our preconditioner on Laplace's equation by showing examples of "diffusion pixels", which can be regarded as an approximate meshless version

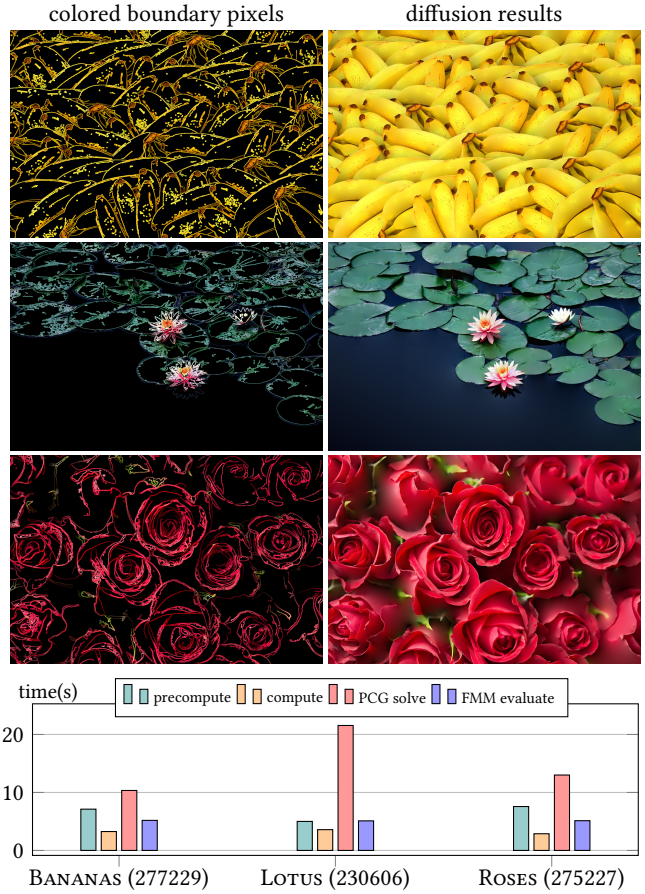


Fig. 8. **Diffusion pixels.** We provide stage-by-stage timings for our algorithm on three "pixel diffusion" examples. The number of input colored pixels (i.e., boundary points) for diffusion is shown between parentheses. The final image size is 1280×853, i.e., over 1 million target points to evaluate. Here, the sparsity pattern uses $\rho=6$. PCG time includes solving three RGB channels, each using 7 PCG iterations. Note that using fewer iterations would be even faster and the incurred error would be hardly noticeable.

of *diffusion curves* for image synthesis [Orzan et al. 2008; Sun et al. 2012]. For a number of given color pixels $\{\boldsymbol{y}_i\}_i$ with prescribed colors $\{\boldsymbol{b}_i\}_i$ (including three RGB channels as Dirichlet boundary conditions), we can solve for Eq. (5) via PCG to obtain the "color charge" of each source point $\boldsymbol{z}_i\equiv\boldsymbol{y}_i$. Finally, these color charges are extrapolated to diffuse the boundary colors to the whole image, which can be used to create vector graphics (as we could zoom in by computing more target points easily through FMM) or as a way to compress real-world photos. On each example in this paper, we take an input image, extract its "edge pixels" (and their colors) through a basic Canny edge-detector filter, and use these pixels and their four immediate neighbors (resp., their colors) as boundary points (resp., Dirichlet boundary colors) for a Laplacian-based BIE. The solution of our solve is thus a harmonic blending of these boundary colors, reproducing the original image well.

Reconstructing a high-resolution image from these boundary color pixels can easily result in a very large BIE (~$270,000^2$ for

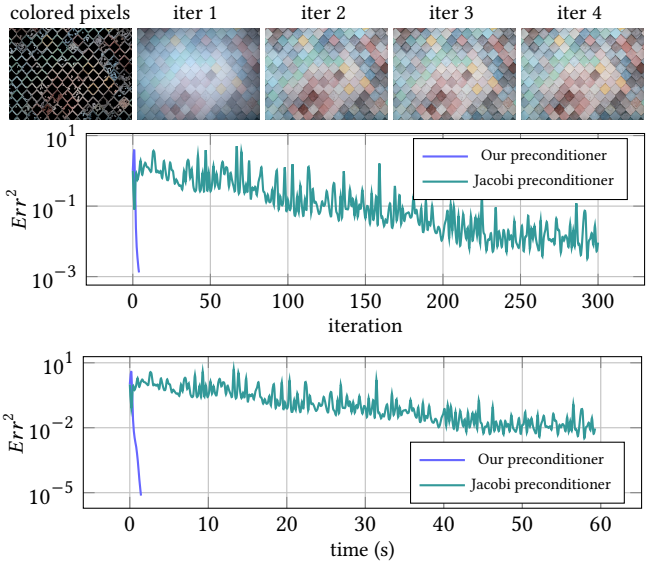


Fig. 9. **PCG convergence.** This example uses $109K$ colored pixels as Dirichlet boundary conditions. Applying our inverse-Cholesky based preconditioner (with $\rho = 4$) remarkably accelerates convergence compared to a Jacobi-based preconditioned solve, and the results are almost indistinguishable after only 2 iterations. Other preconditioners, such as Gauss-Seidel or SOR, are simply too costly to apply on dense matrices.

Fig. 8), and the boundary conditions (i.e., the prescribed colors) often involve a wide range of frequencies — both facts posing great challenges for traditional direct or iterative solvers. Our preconditioner resolves these difficulties remarkably well: in Fig. 8, we breakdown the time cost of our method on three 1280×853 images, with BIE matrices of sizes above $230K \times 230K$, and a number of target points over $1M$, for a total time of around 30 seconds to produce the final diffusion images on our laptop. For just 7 PCG iterations per RGB channel, the averaged relative error is between 10^{-3} to 10^{-2}. In Fig. 9, we compare the performance of PCG with our preconditioner and with a simple Jacobi (diagonal) preconditioner on a smaller example of size 640×480. Our method quickly reduces the error and the diffusion result is almost indistinguishable after only two iterations, while the Jacobi-based PCG takes two orders of magnitude more iterations to reach a comparable error level. We further tested our algorithm on an even larger image in Fig. 2, with $1.35M$ DoFs for the BIE solve and $8.4M$ target points to evaluate. Only 9 iterations were needed to reduce the error around 1% using only 9GB of GPU memory — while a simple Conjugate Gradient could not converge even after $1.5K$ iterations and more than one day. Finally, we tested a 3D diffusion case in Fig. 10. Since MFS is meshless, we do not need a manifold or singly-connected mesh, so our diffusion is based on a painted scalar field on a mesh with many holes, which emits its heat towards two grey walls. For around $240K$ DoFs and $524K$ target points (on the two walls lit by this lamp), it took less than 20 seconds in total to solve and evaluate the solution, with 7 PCG iterations needed to reach a relative error of less than 10^{-2}. Notice in comparison that recent papers related to diffusion images (e.g., [Bang et al. 2023]) typically limit their examples to have less than $16K$ DoFs for their BIE — and only approximate the boundary conditions due to a smaller amount of source points.

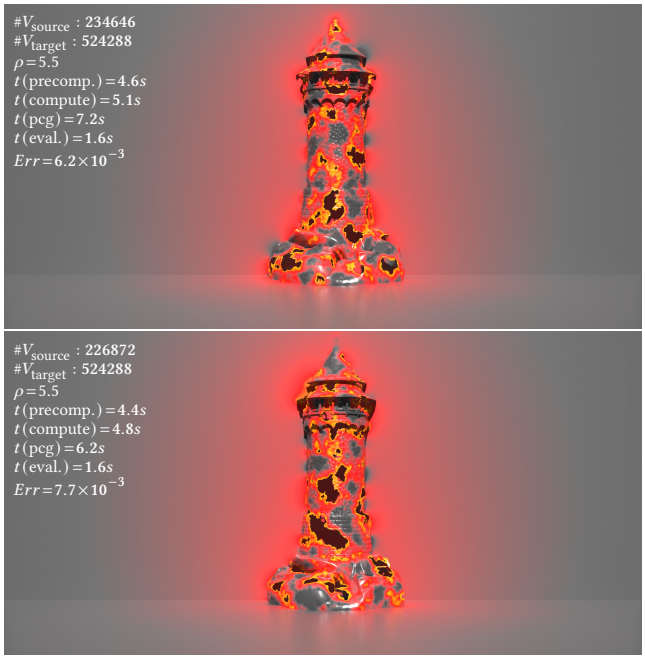


Fig. 10. **3D Laplacian.** We paint an intensity field on the Tower mesh ($0.2M$ points, with different irregular holes in the top and bottom examples) representing heat, which will radiate to the target planes ($0.5M$ points). The whole process takes less than 20 seconds for a relative error below 10^{-2}.

4.3 Linear elasticity

Our second example deals with linear elasticity. The equation for elastic deformation encoded via a displacement (vector) field u is:

$$\Delta u + \frac{1}{1-2\nu}\nabla(\nabla \cdot u) = 0,$$

where ν is the Poisson ratio, quantifying the incompressibility of the elastic material. The fundamental solutions to linear elasticity are known as Kelvinlets, written as

$$G(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} \dfrac{a-b}{r}\ln(1/r)\boldsymbol{I} + \dfrac{b}{r^2}\boldsymbol{r}\boldsymbol{r}^{\mathsf{T}}, & \text{in 2D} \\ \dfrac{a-b}{r}\boldsymbol{I} + \dfrac{b}{r^3}\boldsymbol{r}\boldsymbol{r}^{\mathsf{T}}, & \text{in 3D} \end{cases} \tag{14}$$

where $a = 1/2^{d-1}\pi$ and $b = a/4(1-\nu)$ in dimension $d = 2, 3$, while r still denotes the distance $\|\boldsymbol{x} - \boldsymbol{y}\|$. Kelvinlets have recently been applied to digital sculpting for real-time volumetric deformation [De Goes and James 2017]. However, enforcing positional constraints using Kelvinlets can significantly degrade performance: it involves solving a dense linear system, very much akin to the BIE in Eq. (5), via either Cholesky or LU decomposition, and further constraints can be added via rank-one updates. Our preconditioner applies directly to this case of multiple constraints in the Kelvinlet approach, greatly enhancing the efficiency with which one can solve this problem. In our results, we use a boundary in the shape of a box surrounding the initial object, and we provide a series of *displacement vectors* on boundary points, which we picked to be on a grid for each of the faces of the box, see Fig. 13 (top). The MFS-derived BIE is then solved to return forces at these boundary points, from which we derive the elastic deformation applied to a car through FMM-based

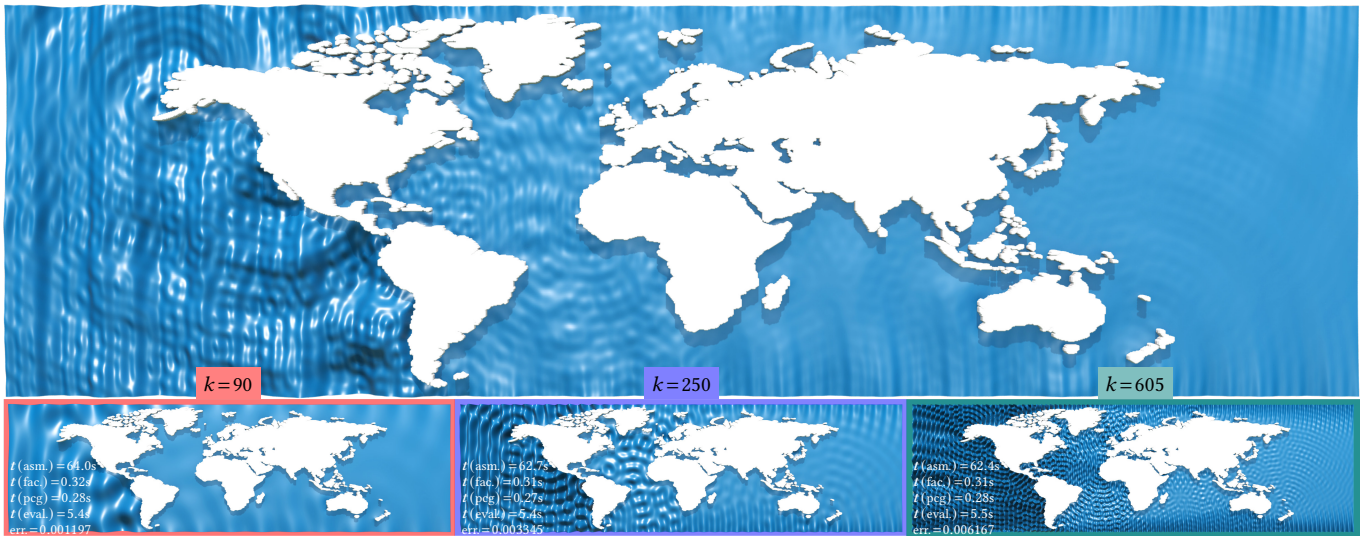


Fig. 11. **2D Helmholtz for various wave numbers.** We solve the scattering of input waves with $12.7K$ boundary points on the World Map model. For this least-squares case, assembling κ_{s_j,s_j} becomes the most computationally expensive step, while time cost for computing the Cholesky factor (with $\rho=8$) and PCG solve are almost negligible. We also observe that as k increases, the error grows larger due to the worsening condition number for high-frequency Helmholtz equations. Still, we only apply 15 iterations for PCG, giving an accurate enough result for this particular wave-propagation purpose.

evaluation. We used 14408 boundary points on this example, leading to a BIE matrix of size 43224×43224. Our PCG solver took less than 8 seconds to solve the BIE equation with an error below 5×10^{-3}. Since the car model has about $199K$ vertices, its deformation was evaluated in less than 10 seconds. The example of the eagle in Fig. 1 has $134K$ vertices, which took 5 seconds to deform based on the same BIE solution than the car.

4.4 Helmholtz equation

We discuss another possible application of our preconditioner, this time to a slightly more involved case to demonstrate the range of operators our approach can deal with. The Helmholtz equation has often been used in Computer Graphics, whether for acoustic transfer [James et al. 2006], or for the animation of linear water waves [Schreck et al. 2019] that are separable in space and time. For a complex-valued function u in space, it imposes

$$\Delta u + k^2 u = 0, \tag{15}$$

where $k\neq0$ is the wave number representing the frequency of the solution. The Green's functions are known to be of the form

$$G(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} \frac{\mathbb{i}}{4}H_0^{(1)}(kr), & \text{in 2D,} \\ \frac{\exp(\mathbb{i}kr)}{4\pi r}, & \text{in 3D,} \end{cases} \tag{16}$$

where $\mathbb{i}$ is the imaginary unit number, and $H_0^{(1)}$ is the zeroth-order Hankel function of the first kind (i.e., $H_0^{(1)}(x)=J_0(x)+\mathbb{i}Y_0(x)$ where J_0 and Y_0 are the zeroth-order Bessel function of the first and second kind respectively). In sharp contrast to the Laplace and elasticity cases described earlier, our method does not directly apply because Helmholtz's BIE matrix $\boldsymbol{K}$ is now complex and *not* Hermitian positive definite. As a consequence, we solve the *least-squares* BIE problem for the Helmholtz equation, as we mentioned in Sec. 2.

There are thus two major differences in terms of implementation that we need to address. First, in the BIE solve stage, we must *explicitly* store $\boldsymbol{K}$ (which we never had to do before), since evaluating each term $(\boldsymbol{K}^{\mathsf{H}}\boldsymbol{K})_{ij}$ would require to compute an inner product $\sum_{k=1}^{B} G(\boldsymbol{y}_k,\boldsymbol{z}_i)^{\mathsf{H}}G(\boldsymbol{y}_k,\boldsymbol{z}_j)$, bringing lots of redundant computations. Second, in the interpolation stage, we directly compute the matrix-vector product parallelized over each target points on GPU, as the black-box FMM does not perform well on oscillatory kernel functions in general due to its use of low-rank speedup. This Helmholtz case thus requires both more memory and a bit more time to evaluate the final target values. In our tests, we solve for an input standing plane wave propagating in a direction $\boldsymbol{d}$ being scattered by boundaries. The plane wave is decomposed into space and time parts, i.e., $u_{\text{in}}(\boldsymbol{x},t)=\bar{u}_{\text{in}}(\boldsymbol{x},k)\exp(-\mathbb{i}\,\omega_k t)$ where ω_k is the angular frequency and $\bar{u}_{\text{in}}(\boldsymbol{x},k)=\exp(\mathbb{i}\,\boldsymbol{d}^{\mathsf{T}}\boldsymbol{x})$. The Dirichlet-based BIE in frequency space is thus $\sum_{j=1}^{B} G(\boldsymbol{y}_i,\boldsymbol{z}_j)s_j+\bar{u}_{\text{in}}(\boldsymbol{y}_i,k)=0$. We can then evaluate

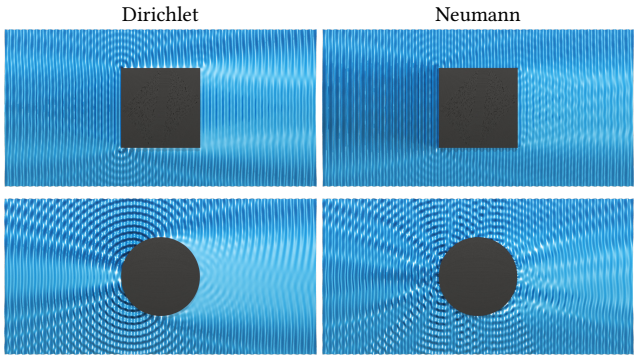


Fig. 12. **Dirichlet vs. Neumann conditions.** Both the Dirichlet (left) and Neumann problems (right) can be solved by MFS, using either the original Green's functions or its second-order normal derivatives. Both produce interesting reflections of input waves, here for a wave number set to $k=300$.

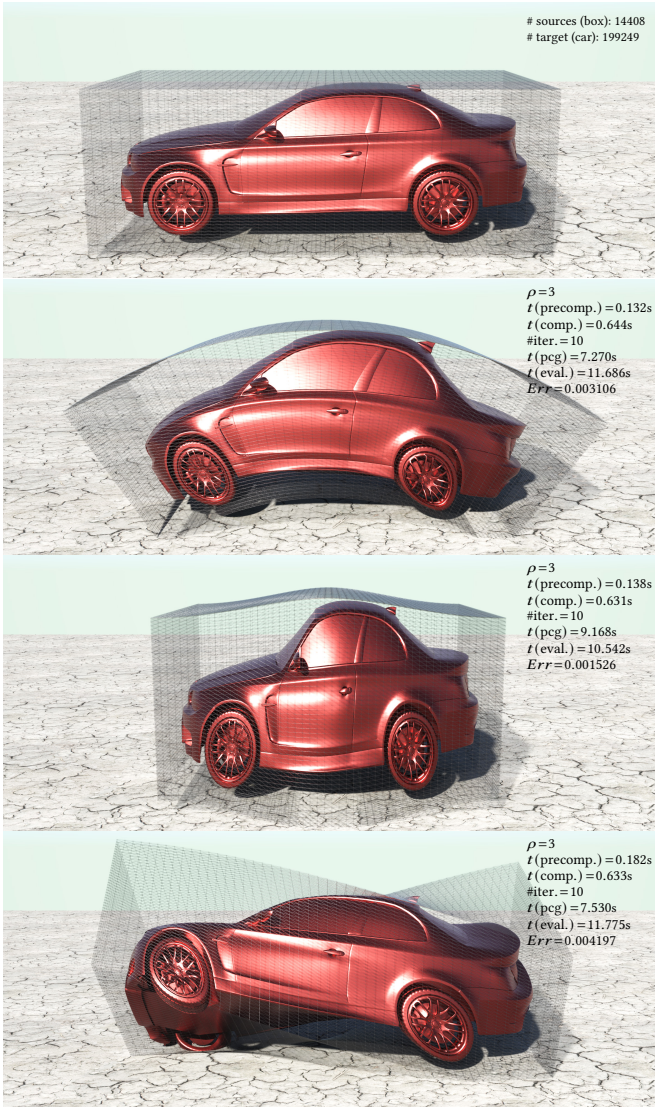
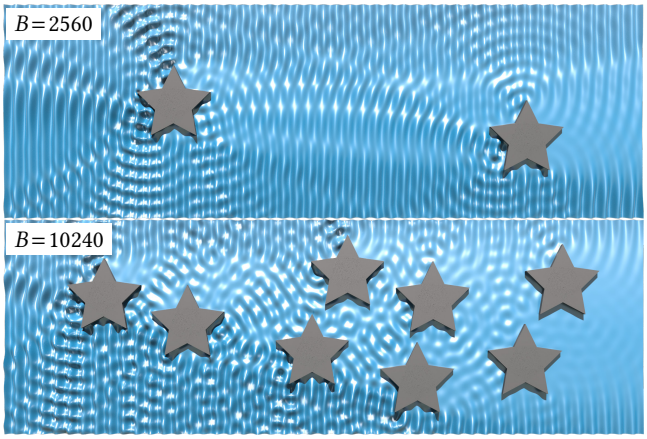


Fig. 13. **Densely constrained Kelvinlets.** While Kelvinlets [De Goes and James 2017] allow realtime sculpting of models, setting point constraints leads to dense linear solves, preventing fast results. With our preconditioner, we can constrain volumetric deformation using Kelvinlets far more efficiently: for this extreme case with over $14K$ constraints on a bounding box of a car, we solve the BIE allowing to compute the induced elastic deformation of the car in less than 8 s.

the wave solution $\bar{u}(\boldsymbol{x}, k)$ through matrix-vector evaluation (and further adding back the term $\bar{u}_{\text{in}}(\boldsymbol{x}, k)$), before reconstructing the wave $\bar{u}(\boldsymbol{x}, k)\exp(-\mathbb{i}\omega_k t)$ and taking only its real part as the wave height in time. Comparison between our preconditioner used to solve the least-squares BIE vs. an SVD solver from the Eigen library [Guennebaud et al. 2010] based on a recursive divide-and-conquer strategy is provided in Table 1. Even for small problems, we offer an order of magnitude speedup; as the problem size grows, our method gets to be three orders of magnitude faster. In Fig. 11, we tested our preconditioner on convoluted boundaries with $12.7K$ boundary points.

Table 1. **Comparison with SVD.** We compare our solver (for $\rho=8$) with a divide-and-conquer based SVD implemented in the Eigen library. We solve the Helmholtz equation with $k=300$, using different numbers of boundary points to scatter an input wave. Our method outperforms SVD in speed by one to three orders of magnitude, depending on the boundary size.

B	SVD		Ours					
	t(fac.)	t(slv.)	t(precomp.)	t(comp.)	#iters	t(pcg)	t(total)	Err
1280	4.864	0.003	0.004	0.419	15	0.005	0.427	0.000706
2560	33.757	0.011	0.007	0.715	15	0.013	0.735	0.000679
5120	261.454	0.045	0.013	1.270	15	0.048	1.331	0.004405
7680	911.212	0.156	0.023	3.478	15	0.099	3.600	0.003497
10240	2405.59	0.303	0.032	7.170	15	0.167	7.369	0.003665



Through profiling, we found out that most of the execution time was spent on assembling the terms $\boldsymbol{K}_{s_i,s_j}$, while computing the preconditioner itself and then iterating PCG to convergence took less than 1 s. in total. Since we parallelized evaluation using GPU, extrapolating the solution to $288K$ target points is still quite efficient. It is worth mentioning that the error of our PCG solve increases as the wave number grows, because the matrix $\boldsymbol{K}$ becomes more ill-conditioned: how to efficiently solve a *high-frequency* Helmholtz equation is still a very active research field in applied mathematics.

Solving the BIE system coming from a Neumann problem is equally efficient with our method. Similar to the Dirichlet variant we discussed above, we formulate the problem by computing the second-order normal derivatives of the Green's functions and write the boundary integral equation

$$\sum\nolimits_{j=1}^{B} \frac{\partial^2 G(\boldsymbol{y}_i, \boldsymbol{z}_j)}{\partial n_{\boldsymbol{z}_j} \partial n_{\boldsymbol{y}_i}} s_j + \frac{\partial \bar{u}_{\text{in}}(\boldsymbol{y}_i, k)}{\partial n_{\boldsymbol{y}_i}} = 0,$$

which amounts to a double-layer BEM for Neumann boundary conditions. Both variants result in interesting reflections of input waves as Fig. 12 depicts and as demonstrated in [Schreck et al. 2019].

4.5 Discussion

To conclude this section, we discuss a few more points to better assess our contributions and their consequences.

Uncertainty quantification. Investigating traditional graphics problems from a stochastic process point of view has gain interest recently [Sellán and Jacobson 2022]. Instead of seeking for a deterministic function as the solution, a stochastic approach focuses on the *distribution* of solutions, from which the uncertainty hidden in the solve process can be quantified. In many ways, the idea of Gaussian

Process fits well with the boundary value problem — computing the conditional mean for prediction based on observed data for training can directly map to interpolating the boundary sources to unknown target values as in our boundary value problem [Schäfer et al. 2021a]. To make the relation explicit, we can reinterpret the MFS method using Gaussian processes. Assume that the solution to a PDE is no longer deterministic but random, and respects a zero-mean Gaussian distribution on both boundary points and target points, i.e.,

$$\begin{bmatrix} u(\boldsymbol{x}) \\ u(\boldsymbol{y}) \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} \boldsymbol{K}(\boldsymbol{x},\boldsymbol{x}) & \boldsymbol{K}(\boldsymbol{x},\boldsymbol{y}) \\ \boldsymbol{K}(\boldsymbol{y},\boldsymbol{x}) & \boldsymbol{K}(\boldsymbol{y},\boldsymbol{y}) \end{bmatrix}\right). \tag{17}$$

In this sense, Green's functions can be seen as GP kernel functions tailored to PDEs. Next, from boundary values (observed data), we can "predict" the solution at targets (unobserved variables), which is computed through conditioning the Gaussian process:

$$\mu\left(f(\boldsymbol{x}) \mid \boldsymbol{y}, f(\boldsymbol{y})\right) = \boldsymbol{K}(\boldsymbol{x},\boldsymbol{y})\boldsymbol{K}(\boldsymbol{y},\boldsymbol{y})^{-1} f(\boldsymbol{y}), \tag{18}$$

which exactly reassembles the two stage of our MFS solve. The solution to a PDE by MFS can thus be seen as the most likely (or expected) solutions given the boundary value, and there could be other solutions which may occur with a smaller probability. To quantify the uncertainty of solution at a target $\boldsymbol{y}_i$, we can further compute its conditional variance based on the boundary data:

$$\sigma^2_{\boldsymbol{y}_i} = \boldsymbol{K}(\boldsymbol{y}_i,\boldsymbol{y}_i) - \boldsymbol{K}(\boldsymbol{y}_i,\boldsymbol{x})\boldsymbol{K}(\boldsymbol{x},\boldsymbol{x})^{-1}\boldsymbol{K}(\boldsymbol{x},\boldsymbol{y}_i). \tag{19}$$

We point out these are also the diagonal entries of the Schur complement of the BIE matrix. Knowing the conditional mean and variance for an unobserved point, not only its most expected solutions can be obtained, one can also measure that how likely the solution would be in a given range (see Fig. 14 for an example), and this could be useful to statistically determine critical regions when solving a PDE, helping to either exclude insignificant DoFs to reduce computational cost or refine sampling to improve the confidence of results. This interpretation can also be viewed as a special case of solving PDEs using GPs or radial basis functions [Fornberg and Flyer 2015; Cockayne et al. 2017; Chen et al. 2021a, 2023] that simplifies due to the radial basis/covariance function being the Green's function of the PDE to be solved. While we simply take Green's functions as the prior kernel function fed to the Gaussian process in this paper, to better quantify how reliable the solve is, prior kernels should be problem-adapted and should account for various sources of uncertainty in computation, for instance the discretization error or hidden noise in boundary data, so that we may develop robust filters for more reliable solutions.

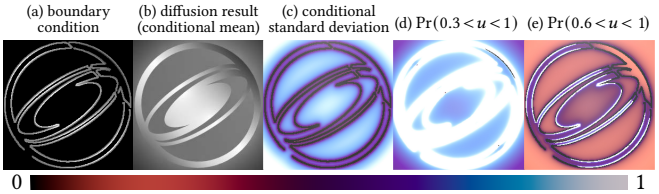


Fig. 14. **Uncertainty quantification.** The MFS solve and evaluation can be reinterpreted as conditioning of a Gaussian process where boundary values are the observed data (a), from which we can compute the solution as the conditional mean (b), as well as the conditional variance (c) to quantify the uncertainty of the solution. This enables us to evaluate spatial probability distributions where different solution ranges are expected (d,e).

GP vs. our approach. Let us also revisit the differences between previous GP-based works and our approach presented here to clarify our contributions. Clearly, Gaussian Processes are very similar to MFS/BEM in the sense that they start from training points (our boundary points) and evaluate prediction points (our target points). The recent works we relied on to choose a variational inverse-Cholesky factor [Guinness 2018; Schäfer et al. 2021a; Katzfuss and Guinness 2021] compute a GP conditional mean (our PDE solution) without separating solution and evaluation stages: it thus involves a larger covariance matrix $[\boldsymbol{K}(\boldsymbol{x},\boldsymbol{x}), \boldsymbol{K}(\boldsymbol{x},\boldsymbol{y}); \boldsymbol{K}(\boldsymbol{y},\boldsymbol{x}), \boldsymbol{K}(\boldsymbol{y},\boldsymbol{y})]$ of size $(B+T)\times(B+T)$, from which the conditional mean can be directly deduced from sub-matrices of its resulting inverse-Cholesky factor through a back-substitution after a matrix-vector operation. While they show that this is efficient for interpolation with Matérn covariance, their direct "training-to-prediction" approach with "prediction points first" adapted to our case introduces significant errors in the approximation. This is likely because our Green's functions decay slower than Matérn's and are more singular, and our regression problem is more extrapolatory than interpolatory in nature. Increasing ρ to mitigate this problem is not a practical solution in our problems due to the increase in the computational cost shown in Fig. 16. The "prediction points last" approach of [Schäfer et al. 2021a] promises significantly more accurate extrapolation, but is prohibitively expensive if there are too many target points — which is often the case in MFS. Thus, we use a procedure closer to Kaporin's original idea of variational preconditioning, this time applied to *dense matrices*, which achieves a better tradeoff between accuracy and cost by using PCG with FMM, providing a fast, controllable way to correct the approximate solve that the use of $\boldsymbol{L}_S\boldsymbol{L}_S^\mathsf{T}$ by itself would result in. Alternatively, we could have used a joint reverse max-min ordering on boundary and target points to compute efficient matrix-vector products with $\boldsymbol{K}$ using the approach of [Schäfer et al. 2021a], or used the FMM matrix-vector product to improve the computational efficiency of the "prediction points last" approach; we defer the exploration of these alternatives to future work.

$\mathcal{H}$-matrix based LU preconditioning. Because hierarchical matrices are data-sparse, their LU factorization can be somewhat performed more efficiently [Kriemann 2013]. Despite the superlinear behavior of regular LU in memory and execution time, we tested the "$\mathcal{H}$-LU" approach to preconditioning using an existing library [Kriemann 2024] for both the computation of the preconditioner and its use in the PCG. As Fig. 15 demonstrates (where the computational time of the preconditioning phase before PCG iterations is observed as the flat portion of the error curve, since no error decrease can be witnessed during these precomputations), the LU factorization leads to a great conditioning but at an unreasonable time cost. Using a more aggressive approximation for the hierarchical matrix can speed up factorization, but the significant drop in precision this creates prevents PCG iterations to converge as matrix-vector multiplications are performed with the $\mathcal{H}$-matrix.

5 CONCLUSIONS

In this paper, we provided a simple approach to significantly accelerate the Method of Fundamental Solutions and Boundary Element Methods by offering an efficient preconditioner to solve boundary integral equations through Preconditioned Conjugate Gradient.

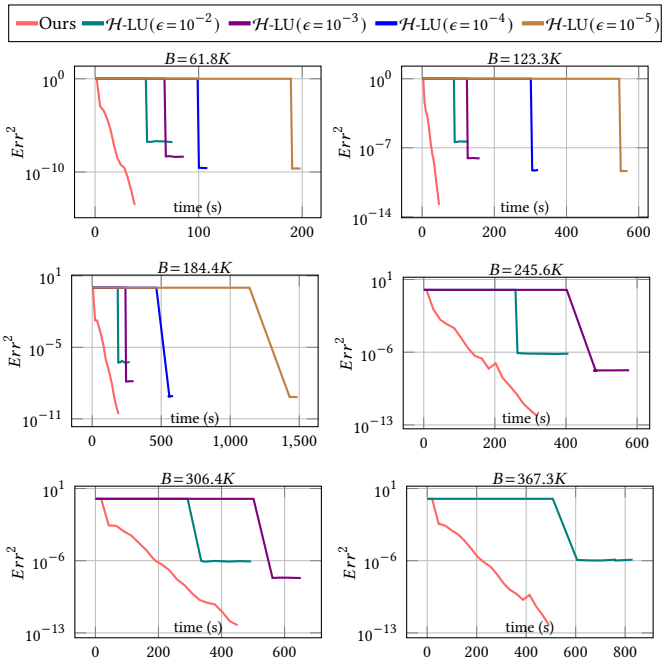


Fig. 15. **Comparison with $\mathcal{H}$-LU.** For this test, we use Poisson disk sampling to generate a number of points randomly within a unit bounding box. The right hand side vector is initialized with randomly 1 or -1 for each point. Using the HLIBpro library [Kriemann 2024], the quality and speed of $\mathcal{H}$-LU preconditioner is affected by the $\mathcal{H}$-arithmetic accuracy ϵ of its low-rank approximations. While $\mathcal{H}$-LU preconditioner can be moderately efficient with low accuracy, the convergence of PCG becomes essentially limited. Further increasing the accuracy quickly blows up the computational and memory costs, especially for large problems. As the system size goes above $200K$, $\mathcal{H}$-LU with $\epsilon=10^{-4}$ simply runs out of memory on our laptop; worse, the error of the linear solve is still stuck at a high level compared to ours, which shows no convergence issue (here, $\rho=2.5$).

While researchers often avoid dense matrices in the belief that no fast solver can handle them, we prove that this is not always the case by drawing inspiration from the Gaussian Process literature describing conditioning via constrained minimization: through the computation of a sparse approximation of the inverse-Cholesky factor of a large and dense SPD matrix of a BIE, we show that we can dramatically increase the efficiency of boundary solvers, without even requiring surface or volumetric meshes.

Limitations. While our GPU-based factorization step has never encountered breakdowns in all the examples we tried, it should be noted that the local Cholesky decompositions it performs could breakdown if the regularization parameter ε (Sec. 4.1) is chosen too large, for instance, 10 times larger than the minimal spacing between source points — as $\boldsymbol{K}$ may no longer be PSD. Thankfully, we want this ε to be tiny to stay as close as possible to the real Green's function, so this never happens in practical cases. However, this breakdown could also happen if there are points on top of each other, thus creating degenerate $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ terms. While one could simply filter these cases out in the first place, it could be interesting to explore LDL$^{\mathrm{T}}$ factorization to compute $\boldsymbol{L}_{\mathcal{S}_j,j}$ as it applies to indefinite matrices. While theoretical guarantees may not hold anymore, we may still see improved convergence in practice. Finally, using a least-squares matrix instead of an asymmetric BIE may sound like a bad idea as it squares the condition number. Yet, the efficacy of Kaporin's variational preconditioner almost compensates for this practice: as an example, we solved a least-square problem for Laplace's BIE with about $16K$ source points; compared to the original BIE, the least-squares version only needed two extra PCG iterations to converge below an error of 10^{-5}.

Future works. In this work, we tested our preconditioner on three linear elliptic PDEs, but this approach should be applicable to many other problems arising from geometry processing and physically-based simulation. For instance, applying our method to RBF-based surface reconstruction seems quite straightforward [Carr et al. 2001]. Also, we used a black-box FMM for accelerating matrix-vector products. Implementing PDE-specific FMM could further improve the run-time performance of both solution and evaluation stages. Alternatively, the $\mathcal{H}$-matrix representation could be potentially applicable for even faster matrix-vector product if the need for accuracy is less stringent, so we plan on evaluating if this algebraic version of FMM can further accelerate our approach. In the future, it could be very interesting to investigate efficient solvers for nonlinear PDEs using Gaussian processes as in [Owhadi 2023] to continue exploiting the connection we leveraged: indeed, we believe that analyzing and solving PDEs from a stochastic point of view could be helpful in reducing computational cost and improving reliability for virtual simulations of our real world.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their help on improving our exposition. Images used in Figs. 8 and 9 are courtesy of pixabay.com. The Demosthenes mesh for Figs. 4 and 16 is courtesy of Ryan Baumann, the Tower mesh for Figs. 1 and 10 is by jansentee3d via Thingiverse, and the Car mesh for Fig. 13 is courtesy of Mike Pan and Morgan McGuire. FS acknowledges support from the Office of Naval Research under award number N00014-23-1-2545 (Untangling Computation). MD acknowledges the support of Ansys, Adobe Research, and the MediTwin consortium, as well as a Choose France Inria chair from which JC was funded.

REFERENCES

Robert Altmann, Patrick Henning, and Daniel Peterseim. 2021. Numerical homogenization beyond scale separation. *Acta Numerica* 30 (2021), 1–86.

Faisal Amlani, Stéphanie Chaillat, and Adrien Loseille. 2019. An efficient preconditioner for adaptive Fast Multipole accelerated Boundary Element Methods to model time-harmonic 3D wave propagation. *Comput. Methods Appl. Mech. Eng.* 352 (2019), 189–210. https://doi.org/10.1016/j.cma.2019.04.026

Seungbae Bang, Kirill Serkh, Oded Stein, and Alec Jacobson. 2023. An Adaptive Fast-Multipole-Accelerated Hybrid Boundary Integral Equation Method for Accurate Diffusion Curves. *ACM Trans. Graph.* 42, 6, Article 215 (2023). https://doi.org/10.1145/3618374

Richard K. Beatson, Jon B. Cherrie, and Cameron Mouat. 1999. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Adv. Comput. Math.* 11 (1999), 253–270. https://doi.org/10.1023/A:1018932227617

Jonathan Carr, Richard Beatson, Jon Cherrie, T. Mitchell, W. Fright, Bruce McCallum, and T. Evans. 2001. Reconstruction & representation of 3D objects with radial basis functions. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 67–76. https://doi.org/10.1145/383259.383266

Jiong Chen, Florian Schäfer, Jin Huang, and Mathieu Desbrun. 2021b. Multiscale Cholesky preconditioning for ill-conditioned problems. *ACM Trans. Graph. (SIGGRAPH)* 40, 4 (2021), 1–13.

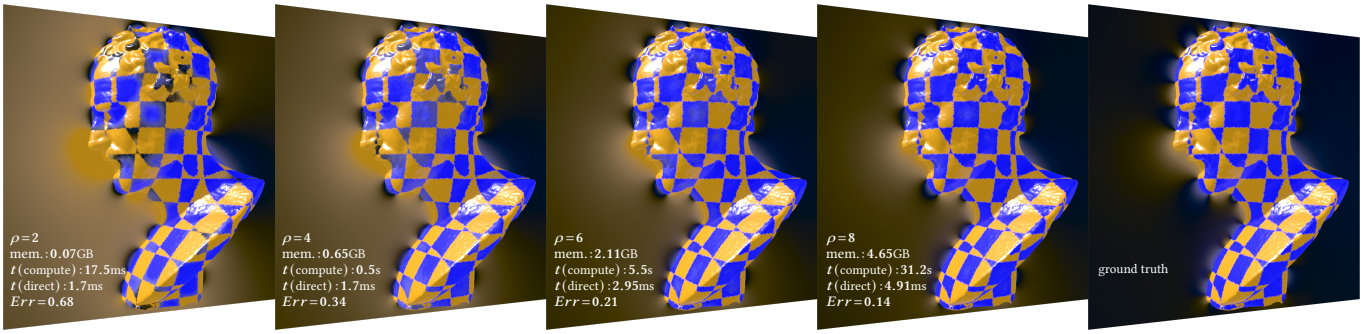


Fig. 16. **Inadequacy of the sparse inverse-Cholesky factor for direct solve.** Using $L_{S_j,j}L_{S_j,j}^{\mathsf{T}}$ to directly solve Eq. (5) is very fast, but its accuracy is rarely sufficient. Increasing the non-zeros of the sparsity pattern (by increasing ρ) does achieve better accuracy, but at the price of a rapid growth of memory consumption and computational cost. Thus we prefer using $L_{S_j,j}L_{S_j,j}^{\mathsf{T}}$ as a preconditioner for conjugate gradient as it turns out to be far more efficient.

Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew Stuart. 2021a. Solving and learning nonlinear PDEs with Gaussian processes. *J. Comp. Phys.* 447 (2021), 110668.

Yifan Chen, Houman Owhadi, and Florian Schäfer. 2023. Sparse Cholesky factorization for solving nonlinear PDEs via Gaussian processes. *arXiv:2304.01294* (2023).

Jon Cockayne, Chris Oates, Tim Sullivan, and Mark Girolami. 2017. Probabilistic numerical methods for PDE-constrained Bayesian inverse problems. In *AIP Conference Proceedings*, Vol. 1853.

Ricardo Cortez. 2001. The method of regularized Stokeslets. *SIAM J. Sci. Comput.* 23, 4 (2001), 1204–1225.

Martin Costabel. 1987. Principles of boundary element methods. *Computer Physics Reports* 6, 1 (1987), 243–274. https://doi.org/10.1016/0167-7977(87)90014-1

Fang Da, David Hahn, Christopher Batty, Chris Wojtan, and Eitan Grinspun. 2016. Surface-only liquids. *ACM Trans. Graph. (SIGGRAPH)* 35, 4 (2016), 1–12.

Fernando De Goes and Doug L James. 2017. Regularized Kelvinlets: sculpting brushes based on fundamental solutions of elasticity. *ACM Trans. Graph. (SIGGRAPH)* 36, 4 (2017), 1–11.

Michael G Duffy. 1982. Quadrature over a pyramid or cube of integrands with a singularity at a vertex. *SIAM journal on Numerical Analysis* 19, 6 (1982), 1260–1262.

Massimiliano Ferronato, Carlo Janna, and Giuseppe Gambolati. 2015. A novel factorized sparse approximate inverse preconditioner with supernodes. *Procedia Computer Science* 51 (2015), 266–275.

Bengt Fornberg and Natasha Flyer. 2015. Solving PDEs with radial basis functions. *Acta Numerica* 24 (2015), 215–258.

L. Greengard and V. Rokhlin. 1987. A fast algorithm for particle simulations. *J. Comput. Phys.* 73, 2 (1987), 325–348. https://doi.org/10.1016/0021-9991(87)90140-9

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen library. http://eigen.tuxfamily.org.

Joseph Guinness. 2018. Permutation and Grouping Methods for Sharpening Gaussian Process Approximations. *Technometrics* 60, 4 (2018), 415–429. https://doi.org/10.1080/00401706.2018.1437476

Wolfgang Hackbusch. 1999. A Sparse Matrix Arithmetic Based on $\mathcal{H}$-Matrices. Part I: Introduction to $\mathcal{H}$-Matrices. *Computing* 62 (04 1999), 89–108. https://doi.org/10.1007/s006070050015

Wolfgang Hackbusch and B. Khoromskij. 2000. A Sparse $\mathcal{H}$-Matrix Arithmetic, Part II: Application to Multi-Dimensional Problems. *Computing* 64 (01 2000).

Libo Huang and Dominik L Michels. 2020. Surface-only ferrofluids. *ACM Trans. Graph. (SIGGRAPH)* 39, 6 (2020), 1–17.

Doug L James, Jernej Barbič, and Dinesh K Pai. 2006. Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Trans. Graph. (SIGGRAPH)* 25, 3 (2006), 987–995.

Doug L. James and Dinesh K. Pai. 1999. ArtDefo: Accurate Real Time Deformable Objects. In *Proceedings of the Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. 65–72. https://doi.org/10.1145/311535.311542

I. E. Kaporin. 1994. New convergence results and preconditioning strategies for the conjugate gradient method. *Numer. Linear Algebra Appl.* 1, 2 (1994), 179–210. https://doi.org/10.1002/nla.1680010208

Matthias Katzfuss and Joseph Guinness. 2021. A General Framework for Vecchia Approximations of Gaussian Processes. *Statist. Sci.* 36, 1 (2021), 124 – 141. https://doi.org/10.1214/19-STS755

Liliya Yurievna Kolotilina and Aleksey Yuryevich Yeremin. 1993. Factorized sparse approximate inverse preconditionings I. Theory. *SIAM J. Matrix Anal. Appl.* 14, 1 (1993), 45–58.

Ronald Kriemann. 2013. $\mathcal{H}$-LU factorization on many-core systems. *Comput. Visual Sci.* 16 (2013), 105–117. https://doi.org/10.1007/s00791-014-0226-7

Ronald Kriemann. 2024. HLibPro. https://www.hlibpro.com/.

Dilip Krishnan and Richard Szeliski. 2011. Multigrid and multilevel preconditioners for computational photography. *ACM Trans. Graph. (SIGGRAPH)* 30, 6 (2011), 1–10.

Sebastian Martin, Peter Kaufmann, Mario Botsch, Martin Wicke, and Markus Gross. 2008. Polyhedral Finite Elements Using Harmonic Basis Functions. *Comput. Graph. Forum* 27, 5 (2008), 1521–1529.

Alexandrina Orzan, Adrien Bousseau, Holger Winnemöller, Pascal Barla, Joëlle Thollot, and David Salesin. 2008. Diffusion curves: a vector representation for smooth-shaded images. *ACM Trans. Graph. (SIGGRAPH)* 27, 3 (2008), 1–8.

Houman Owhadi. 2023. Gaussian process hydrodynamics. *Appl. Math. Mech.* (2023), 1–24.

Houman Owhadi and Clint Scovel. 2019. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design.* Vol. 35. Cambridge University Press.

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

Rohan Sawhney, Dario Seyb, Wojciech Jarosz, and Keenan Crane. 2022. Grid-free Monte Carlo for PDEs with spatially varying coefficients. *ACM Trans. Graph. (SIGGRAPH)* 41, 4 (2022), 1–17.

Florian Schäfer. 2021. *Inference, Computation, and Games.* Ph. D. Dissertation. California Institute of Technology.

Florian Schäfer, Matthias Katzfuss, and Houman Owhadi. 2021a. Sparse Cholesky Factorization by Kullback–Leibler Minimization. *SIAM J. Sci. Comput* 43, 3 (2021), A2019–A2046. https://doi.org/10.1137/20M1336254

Florian Schäfer, Timothy John Sullivan, and Houman Owhadi. 2021b. Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. *Multiscale Model. Simul.* 19, 2 (2021), 688–730.

H. Schippers. 1985. Multigrid methods for boundary integral equations. *Numer. Math.*, 46 (1985), 351–363. https://doi.org/10.1007/BF01389491

Camille Schreck, Christian Hafner, and Chris Wojtan. 2019. Fundamental solutions for water wave animation. *ACM Trans. Graph. (SIGGRAPH)* 38, 4 (2019), 1–14.

Silvia Sellán and Alec Jacobson. 2022. Stochastic Poisson Surface Reconstruction. *ACM Trans. Graph. (SIGGRAPH)* 41, 6 (2022), 1–12.

Han Shao, Libo Huang, and Dominik L Michels. 2022. A fast unsmoothed aggregation algebraic multigrid framework for the large-scale simulation of incompressible flow. *ACM Trans. Graph. (SIGGRAPH)* 41, 4 (2022), 1–18.

Michael L Stein. 2002. The screening effect in kriging. *The Annals of Statistics* 30, 1 (2002), 298–323.

Michael L Stein, Zhiyi Chi, and Leah J Welty. 2004. Approximating likelihoods for large spatial data sets. *J. R. Stat. Soc., B: Stat. Methodol.* 66, 2 (2004), 275–296.

Olaf Steinbach and Wolfgang L Wendland. 1998. The construction of some efficient preconditioners in the boundary element method. *Adv. Comput. Math.* 9 (1998), 191–216.

Ryusuke Sugimoto, Christopher Batty, and Toshiya Hachisuka. 2022. Surface-Only Dynamic Deformables using a Boundary Element Method. In *Comput. Graph. Forum*, Vol. 41. 75–86.

Ryusuke Sugimoto, Terry Chen, Yiti Jiang, Christopher Batty, and Toshiya Hachisuka. 2023. A Practical Walk-on-Boundary Method for Boundary Value Problems. *ACM Trans. Graph.* 42, 4, Article 81 (2023). https://doi.org/10.1145/3592109

Xin Sun, Guofu Xie, Yue Dong, Stephen Lin, Weiwei Xu, Wencheng Wang, Xin Tong, and Baining Guo. 2012. Diffusion curve textures for resolution-independent texture mapping. *ACM Trans. Graph. (SIGGRAPH)* 31, 4 (2012), 1–9.

Aldo V Vecchia. 1988. Estimation and model identification for continuous spatial processes. *J. R. Stat. Soc., B: Stat. Methodol.* 50, 2 (1988), 297–312. https://doi.org/10.1111/j.2517-6161.1988.tb01729.x

Ruoxi Wang, Chao Chen, Jonghyun Lee, and Eric Darve. 2021. PBBFMM3D: a parallel black-box algorithm for kernel matrix-vector multiplication. *J. Parallel and Distrib. Comput.* 154 (2021), 64–73.

Botao Wu, Zhendong Wang, and Huamin Wang. 2022. A GPU-based multilevel additive schwarz preconditioner for cloth and deformable body simulation. *ACM Trans. Graph. (SIGGRAPH)* 41, 4 (2022), 1–14.

Aleksey Yuryevich Yeremin, Liliya Yurievna Kolotilina, and A. A. Nikishin. 2000. Factorized sparse approximate inverse preconditionings – III. Iterative construction of preconditioners. *J. Math. Sci.* 101 (2000), 3237–3254. https://doi.org/10.1007/BF02672769

Deyun Zhong, Ju Zhang, and Liguan Wang. 2019. Fast Implicit Surface Reconstruction for the Radial Basis Functions Interpolant. *App. Sci.* 24, 9 (2019), 5335–5349. https://doi.org/10.3390/app9245335

Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph. (SIGGRAPH)* 29, 2 (2010), 1–18.

Daniel Zilber and Matthias Katzfuss. 2021. Vecchia–Laplace approximations of generalized Gaussian processes for big non-Gaussian spatial data. *Comput. Stat. Data Anal.* 153 (2021), 107081.

A VARIATIONAL FORMULATIONS

The preconditioner we present in this paper is the minimizer of the Kaporin condition number under sparsity constraints, but its expression in Eq. (7) can be also derived from two other variational formulations. We very briefly explain these different formulations.

Kaporin condition number. First, we prove Eq. (7) is a minimizer of κ_{Kap}. According to [Kaporin 1994], the Kaporin condition number for the $\boldsymbol{L}_{\mathcal{S}}$-preconditioned system of size $B\times B$ is defined as

$$\kappa_{\text{Kap}} = \frac{1}{B}\frac{\text{tr}(\boldsymbol{K}\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}})}{\det(\boldsymbol{K}\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}})^{\frac{1}{B}}}.$$

By exploiting the properties of matrix trace and matrix determinant and by defining $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j} = \boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{\mathsf{T}}$, we can expand κ_{Kap} as

$$\begin{aligned}
\kappa_{\text{Kap}} &= \frac{1}{\det(\boldsymbol{K})^{\frac{1}{B}}}\frac{\frac{1}{B}\sum_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}}{\left(\prod_j \boldsymbol{L}_{j,j}^2\right)^{\frac{1}{B}}} \\
&= \frac{1}{\det(\boldsymbol{K})^{\frac{1}{B}}}\frac{\frac{1}{B}\sum_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}}{\left(\prod_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}\right)^{\frac{1}{B}}}\left(\frac{\prod_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}}{\prod_j \boldsymbol{L}_{j,j}^2}\right)^{\frac{1}{B}} \\
&= \frac{1}{\det(\boldsymbol{K})^{\frac{1}{B}}}\frac{\frac{1}{B}\sum_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}}{\left(\prod_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}\right)^{\frac{1}{B}}}\left(\prod_j \frac{\left\|\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{\mathsf{T}}\boldsymbol{L}_{\mathcal{S}_j,j}\right\|}{\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\mathbb{e}_j}\right)^{\frac{2}{B}} \\
&= \frac{1}{\det(\boldsymbol{K})^{\frac{1}{B}}}\frac{\frac{1}{B}\sum_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}}{\left(\prod_j \boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}\right)^{\frac{1}{B}}}\left(\prod_j \frac{\left\|\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{\mathsf{T}}\boldsymbol{L}_{\mathcal{S}_j,j}\right\|\left\|\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j\right\|}{\left(\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{\mathsf{T}}\boldsymbol{L}_{\mathcal{S}_j,j}\right)^{\mathsf{T}}\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j}\right)^{\frac{2}{B}} \\
&\quad\times\left(\prod_j \frac{1}{\mathbb{e}_j^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j}\right)^{\frac{1}{B}} \geq \left(\frac{1}{\det(\boldsymbol{K})}\prod_j \frac{1}{\mathbb{e}_j\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j}\right)^{\frac{1}{B}}.
\end{aligned}$$

The inequality holds because of both the inequality of arithmetic and geometric means and of Cauchy-Schwarz inequality. The minimum is reached when $\boldsymbol{L}_{\mathcal{S}_j,j}$ and $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j$ is collinear and the values of $\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}$ are the same for all j. Thus, $\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}$ can be normalized to 1, which results in Eq. (7).

Constrained least-squares problem. [Kolotilina and Yeremin 1993] figured out that Eq. (7) (and equivalently, Eq. (8)) is also the minimizer of a constrained least-square problem, formulated as

$$\underset{\boldsymbol{L}_{\mathcal{S}}}{\text{argmin}}\left\|\boldsymbol{I}-\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{U}\right\|_F^2, \quad \text{s.t. } \text{diag}\left(\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{K}\boldsymbol{L}_{\mathcal{S}}\right)=\boldsymbol{1},$$

where $\boldsymbol{U}$ is the upper triangular Cholesky factor such that $\boldsymbol{K}=\boldsymbol{U}\boldsymbol{U}^{\mathsf{T}}$. To solve this constrained problem, we first write its Lagrangian as:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{L}_{\mathcal{S}},\{\lambda_j\}) &= \text{tr}\left((\boldsymbol{I}-\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{U})^{\mathsf{T}}(\boldsymbol{I}-\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{U})\right)+\sum_j\lambda_j\left(\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}-1\right) \\
&= B-2\sum_j\boldsymbol{L}_{j,j}\boldsymbol{U}_{j,j}+\sum_j\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}+\sum_j\lambda_j\left(\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}-1\right),
\end{aligned} \tag{20}$$

where λ_j's are B Lagrangian multipliers. By differentiating $\mathcal{L}$ w.r.t. $\boldsymbol{L}_{\mathcal{S}_j,j}$, the optimality condition reads

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{L}_{\mathcal{S}_j,j}} = -2\boldsymbol{U}_{j,j}\mathbb{e}_j+2(1+\lambda_j)\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}=\boldsymbol{0},$$

from which $\boldsymbol{L}_{\mathcal{S}_j,j}$ is solved and represented by

$$\boldsymbol{L}_{\mathcal{S}_j,j} = \frac{\boldsymbol{U}_{j,j}}{1+\lambda_j}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\mathbb{e}_j = \frac{\boldsymbol{U}_{j,j}^2}{1+\lambda_j}\boldsymbol{U}_{\mathcal{S}_j,\mathcal{S}_j}^{-\mathsf{T}}\mathbb{e}_j. \tag{21}$$

Finally, each Lagrange multiplier λ_j is found by substituting Eq. (21) into the constraint, yielding

$$\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}-1=\frac{\boldsymbol{U}_{j,j}^4}{(1+\lambda_j)^2}-1=0.$$

Thus, $\boldsymbol{U}_{j,j}^2/(1+\lambda_j)=\pm1$ and the minimizer Eq. (21) reduces to Eq. (8).

KL divergence. Recently, [Schäfer et al. 2021a] found that Eq. (7) also minimizes the KL divergence for two zero-mean Gaussian distributions of covariance $\boldsymbol{K}$ and $(\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}})^{-1}$ respectively. For these two covariance matrices, the KL divergence reduces to

$$\begin{aligned}
&\mathcal{D}_{KL}\left(\boldsymbol{K},\left(\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\right)^{-1}\right) \\
&= -\log\det\left(\boldsymbol{K}\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\right)+\text{tr}\left(\boldsymbol{K}\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\right)-B \\
&= -\log\det\left(\boldsymbol{K}\right)-\log\det\left(\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\right)+\text{tr}\left(\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}}\boldsymbol{K}\boldsymbol{L}_{\mathcal{S}}\right)-B \\
&= \sum_j\boldsymbol{L}_{\mathcal{S}_j,j}^{\mathsf{T}}\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}-2\sum_j\log\left(\boldsymbol{L}_{j,j}\right)+\text{const.}
\end{aligned} \tag{22}$$

Computing the optimality condition minimizing the KL divergence w.r.t. each column $\boldsymbol{L}_{\mathcal{S}_j,j}$ leads to

$$\frac{\partial\mathcal{D}_{KL}}{\partial\boldsymbol{L}_{\mathcal{S}_j}} = 2\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}\boldsymbol{L}_{\mathcal{S}_j,j}-\frac{2}{\boldsymbol{L}_{j,j}}\mathbb{e}_j=\boldsymbol{0}. \tag{23}$$

Again, Eq. (7) is the solution of Eq. (23). In fact, when $(\boldsymbol{L}_{\mathcal{S}}\boldsymbol{L}_{\mathcal{S}}^{\mathsf{T}})^{-1}$ approximates $\boldsymbol{K}$ well, the KL divergence is numerically close to the Frobenius norm, as illustrated in Fig. 7: the reason is that as the term $\boldsymbol{L}_{j,j}\boldsymbol{U}_{j,j}$ in Eq. (20) approaches 1, it will approximate $\log(\boldsymbol{L}_{j,j}\boldsymbol{U}_{j,j})+1$ well.