西北工业大学

数字图像处理–论文翻译

原论文标题：Lightning-fast Method of Fundamental Solutions

张引

计算机学院

计算机科学与技术

学号：2022302728

2024 年 11 月

Northwestern Polytechnical University

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

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.

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

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 $\boldsymbol{K}\boldsymbol{s}=\boldsymbol{b}$ 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

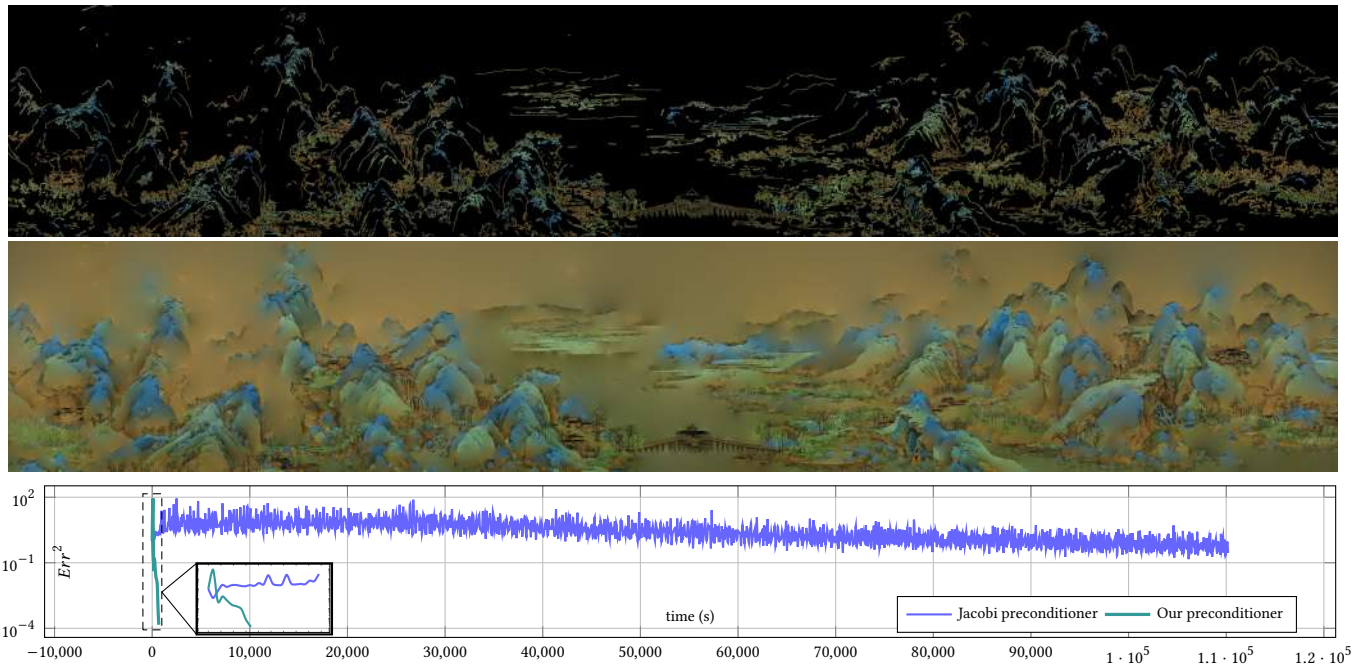


Fig. 2. **Large-scale pixel diffusion.** Here we use $1.35M$ boundary points and $8.4M$ target points, for a final image resolution of 6957×1207. Our algorithm only requires 9GB of memory to store $\boldsymbol{K}_{s_j,s_j}$ and $\boldsymbol{L}_S$ for a sparsity parameter $\rho=6$. Timings for precomputation, 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:

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

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:

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

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

$$\boxed{\boldsymbol{K}\boldsymbol{s}=\boldsymbol{b}} \quad (5)$$

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:

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

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 $\mathbb{e}_j=(1,0,\dots,0)^\mathsf{T}\in\mathbb{R}^{|\mathcal{S}_j|}$ 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

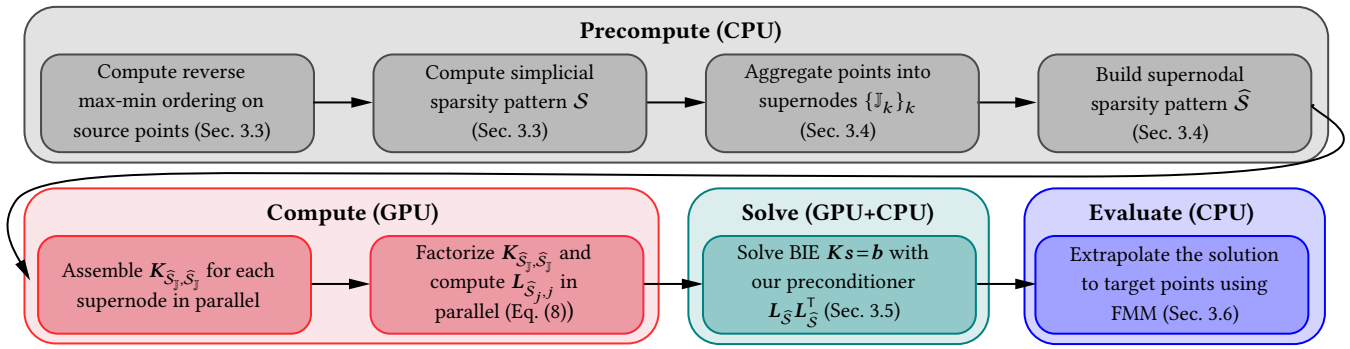


Fig. 3. **Overview.** In order to solve a generic boundary integral equation like Eq. (5), the construction of our approximate Cholesky factor $\boldsymbol{L}_{\mathcal{S}}$ for $\boldsymbol{K}^{-1}$ consists of a few CPU-based precomputations (such as constructing the ordering and the sparsity pattern), 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 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* $\boldsymbol{P}$

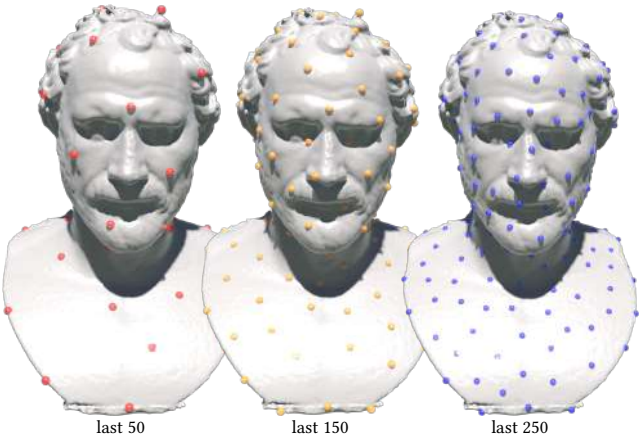


Fig. 4. **Reverse max-min ordering:** Geometrically, the reverse max-min order decomposes all DoFs (here, vertices of a mesh) into different spatial scales, with coarse-scale points (capturing low frequencies) at the end, and fine-scale points (handling high-frequency details) at the beginning.

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 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.

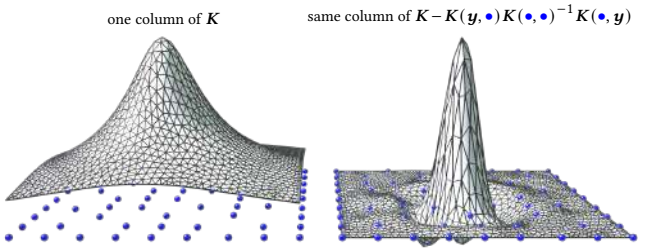


Fig. 5. **Screening effect.** The covariance matrix $\boldsymbol{K}$ features long-range correlations but the covariance conditioned on coarse (blue) samples given by its Schur complement is highly localized. Inverse Cholesky factors of $\boldsymbol{K}$ encode conditional correlations, thus screening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

3.4 Aggregated factorization

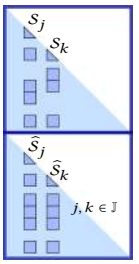
So far, we have seen how each column of $\boldsymbol{L}_{\mathcal{S}}$ only requires the assembly of small matrices $\boldsymbol{K}_{\mathcal{S}_j,\mathcal{S}_j}$ based on our chosen sparsity, which saves time and memory compared to assembling the whole matrix $\boldsymbol{K}$. However, notice that multiple 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 $\boldsymbol{L}_{\widehat{\mathcal{S}}_j,j}, \forall j \in \mathbb{J}$. 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 $\boldsymbol{L}_{\widehat{\mathcal{S}}}$.

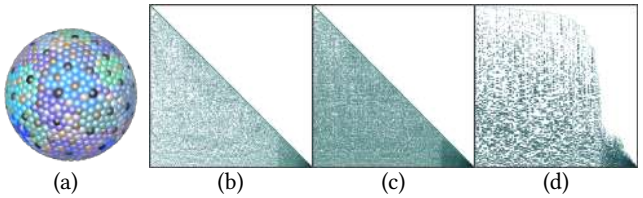
3.5 Preconditioning a BIE matrix K

Now that we have a fast algorithm to evaluate L_S for a given sparsity parameter ρ, and knowing that $L_S L_S^\intercal$ approximate K^{-1}, we may be tempted to simply compute the solution to Eq. (5) via $\boldsymbol{b} = L_S L_S^\intercal \boldsymbol{s}$, 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 $\kappa_{\text{Kap}}(L_S L_S^\intercal K)$. So we can instead use $L_S L_S^\intercal$ as a preconditioner on a conjugate gradient solver to guarantee better efficiency in solving $K\boldsymbol{s}=\boldsymbol{b}$. Moreover, since the preconditioner only needs to be applied to the residual $\boldsymbol{e}_k = K\boldsymbol{s}_k - \boldsymbol{b}$ at iteration k of the conjugate gradient, we only need to apply two low-cost sparse matrix-vector products to evaluate $L_S L_S^\intercal \boldsymbol{e}_k$, 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 $L_{\widehat{S}}$

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 $L_{\widehat{S}}$ such that $K^{-1} \approx L_{\widehat{S}} L_{\widehat{S}}^\intercal$

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

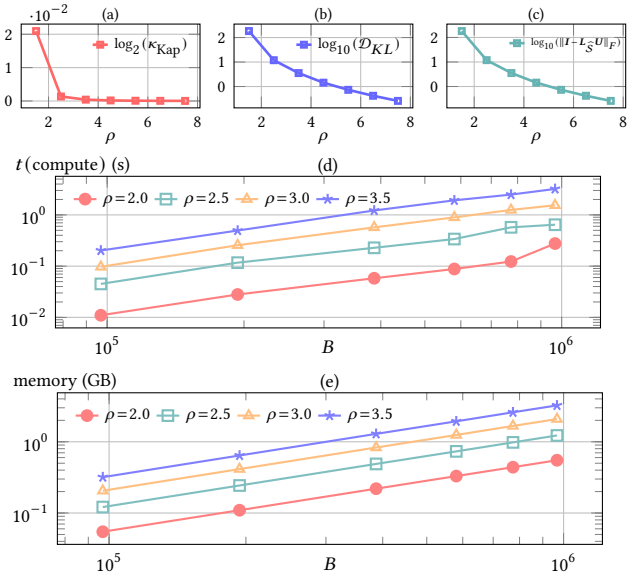


Fig. 7. **Asymptotic behaviors.** We test our algorithm in 3D for boundary points that are uniformly distributed over a square. In (a), (b) and (c), the three variational functionals presented in Appendix A can be seen decreasing as the sparsity parameter ρ increases (since the inverse-Cholesky factor 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 $K\,\boldsymbol{s}$ 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

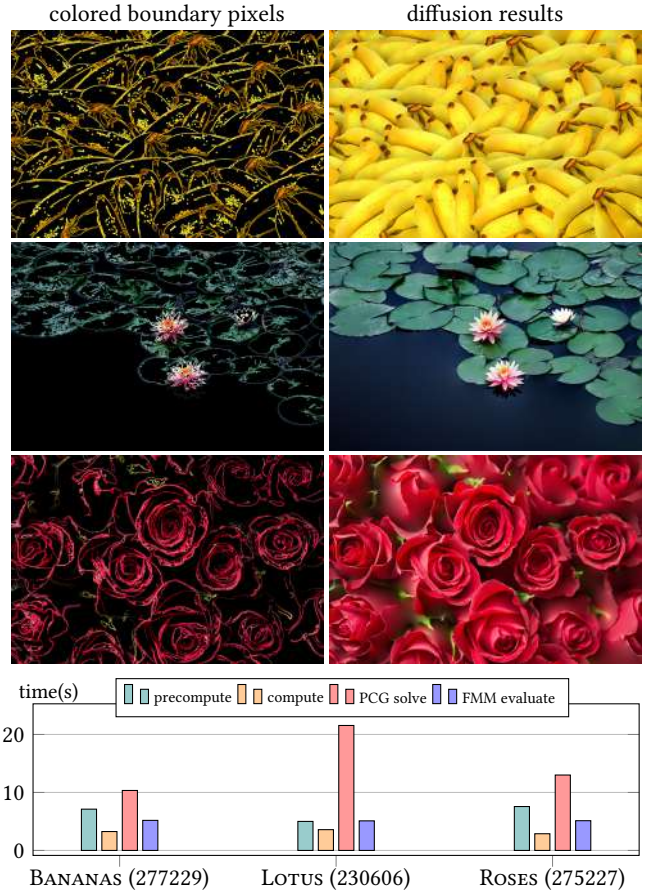


Fig. 8. **Diffusion pixels.** We provide stage-by-stage timings for our algorithm on three "pixel diffusion" examples. The number of input colored pixels (i.e., boundary points) for diffusion is shown between parentheses. The final image size is 1280×853, i.e., over 1 million target points to evaluate. 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

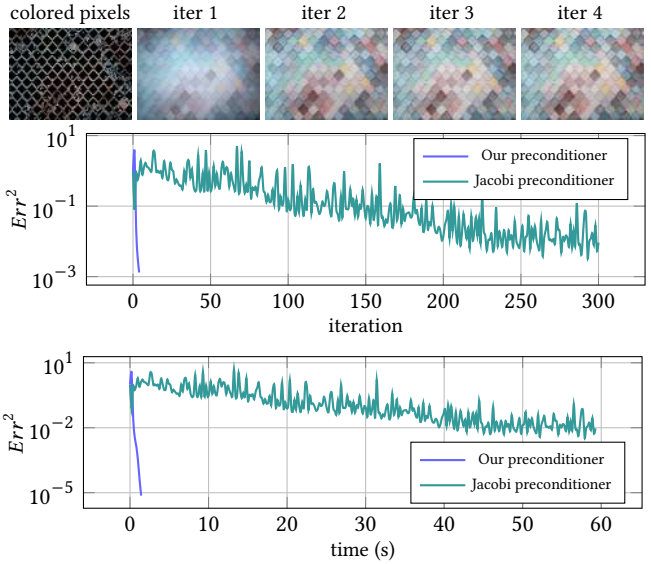


Fig. 9. **PCG convergence.** This example uses $109K$ colored pixels as Dirichlet boundary conditions. Applying our inverse-Cholesky based preconditioner (with $\rho = 4$) remarkably accelerates convergence compared to a Jacobi-based preconditioned solve, and the results are almost indistinguishable 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.

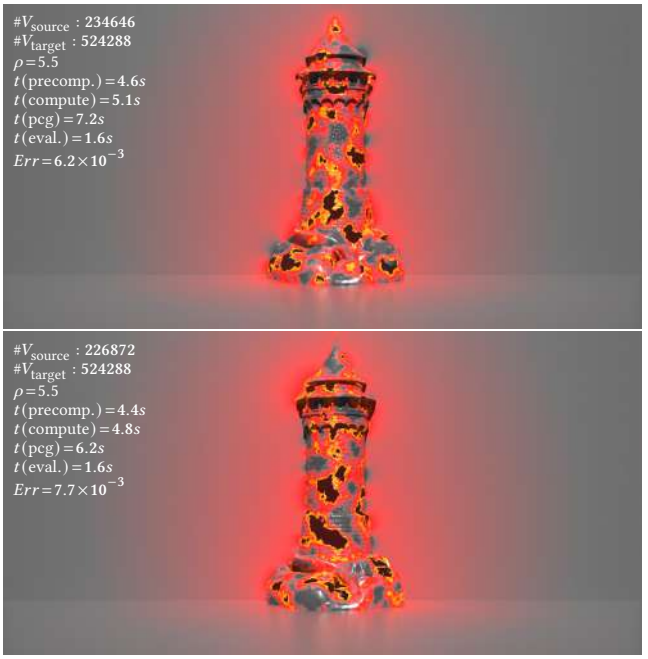


Fig. 10. **3D Laplacian.** We paint an intensity field on the Tower mesh ($0.2M$ points, with different irregular holes in the top and bottom examples) representing heat, which will radiate to the target planes ($0.5M$ points). The whole process takes less than 20 seconds for a relative error below 10^{-2}.

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

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

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

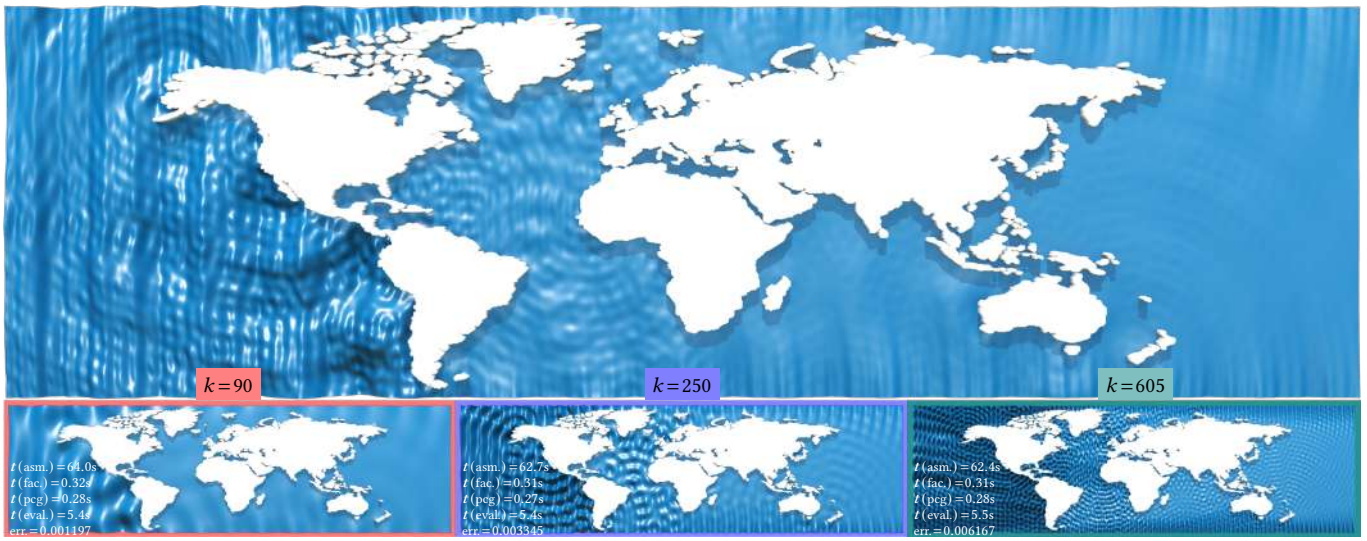


Fig. 11. **2D Helmholtz for various wave numbers.** We solve the scattering of input waves with $12.7K$ boundary points on the World Map model. For this least-squares case, assembling $\boldsymbol{K}_{s_j,s_j}$ becomes the most computationally expensive step, while time cost for computing the Cholesky 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 $k \neq 0$ 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

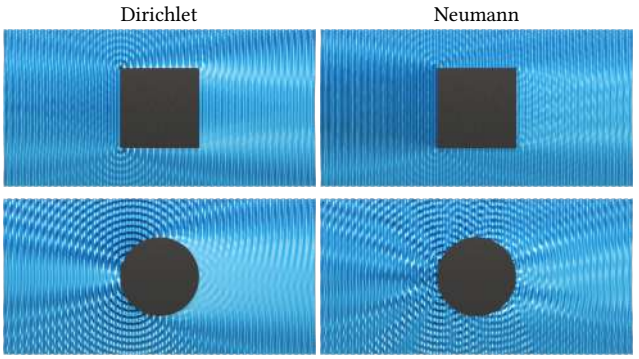


Fig. 12. **Dirichlet vs. Neumann conditions.** Both the Dirichlet (left) and Neumann problems (right) can be solved by MFS, using either the original Green's functions or its second-order normal derivatives. Both produce interesting reflections of input waves, here for a wave number set to $k=300$.

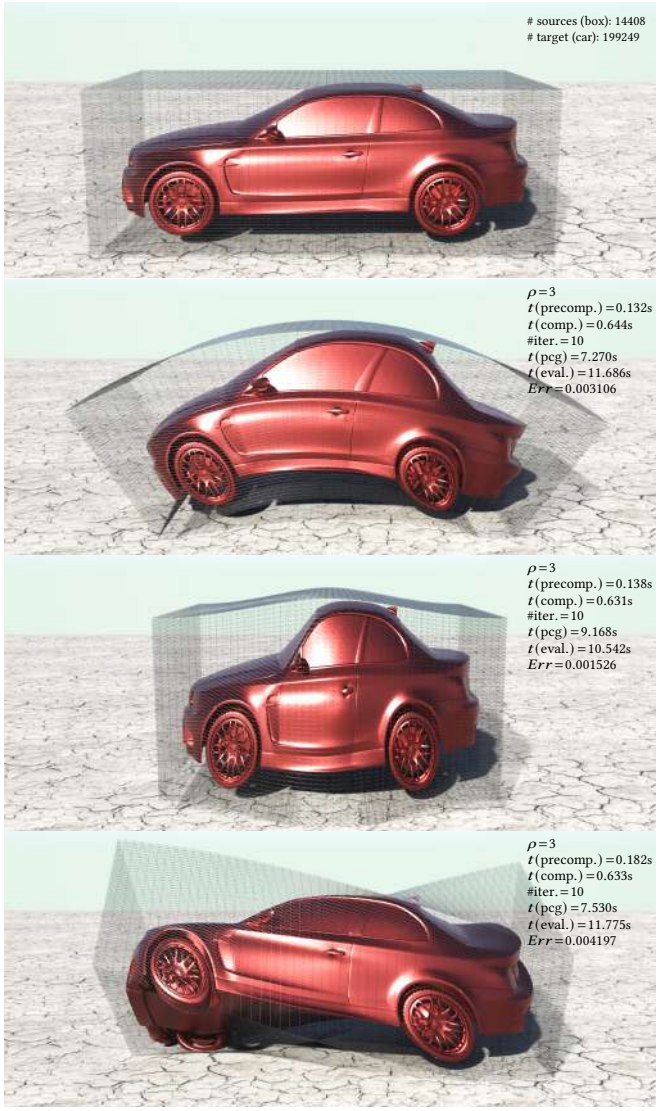
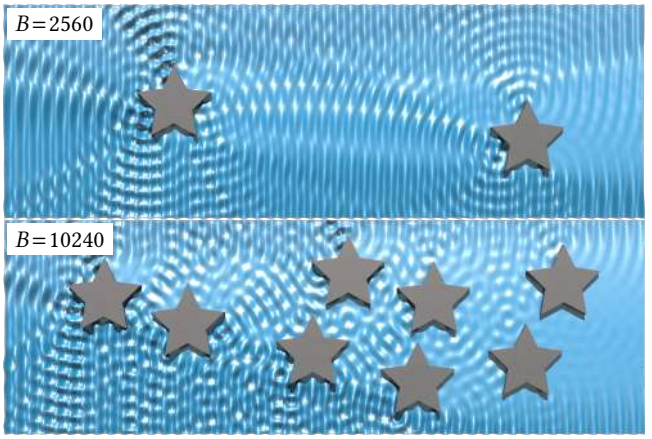


Fig. 13. **Densely constrained Kelvinlets.** While Kelvinlets [De Goes and James 2017] allow realtime sculpting of models, setting point constraints leads to dense linear solves, preventing fast results. With our preconditioner, we can constrain volumetric deformation using Kelvinlets far more efficiently: 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 $\boldsymbol{K}_{s_j,s_j}$, 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.

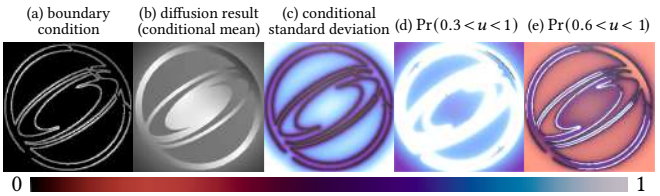


Fig. 14. **Uncertainty quantification.** The MFS solve and evaluation can be reinterpreted as conditioning of a Gaussian process where boundary values are the observed data (a), from which we can compute the solution as the conditional mean (b), as well as the conditional variance (c) to quantify the 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 $\boldsymbol{L}_S\boldsymbol{L}_S^\intercal$ 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.

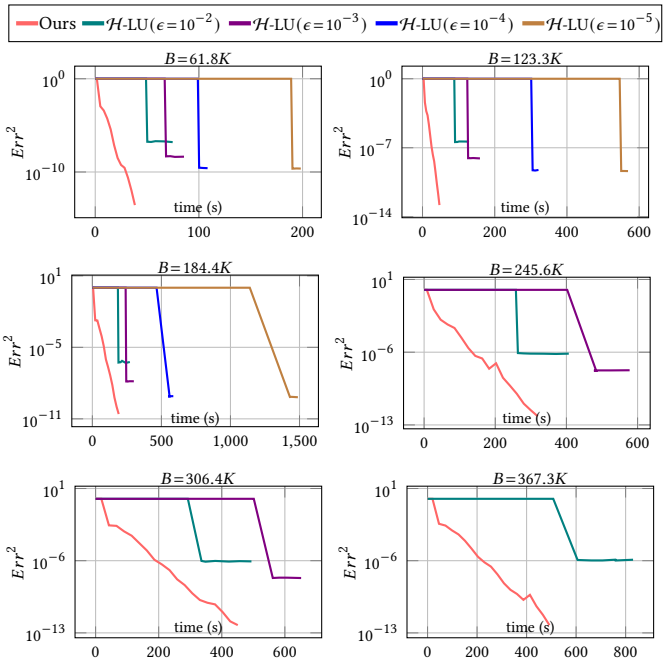


Fig. 15. **Comparison with $\mathcal{H}$-LU.** For this test, we use Poisson disk sampling to generate a number of points randomly within a unit bounding box. The right hand side vector is initialized with randomly 1 or −1 for each point. Using the HLIBpro library [Kriemann 2024], the quality and speed 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.

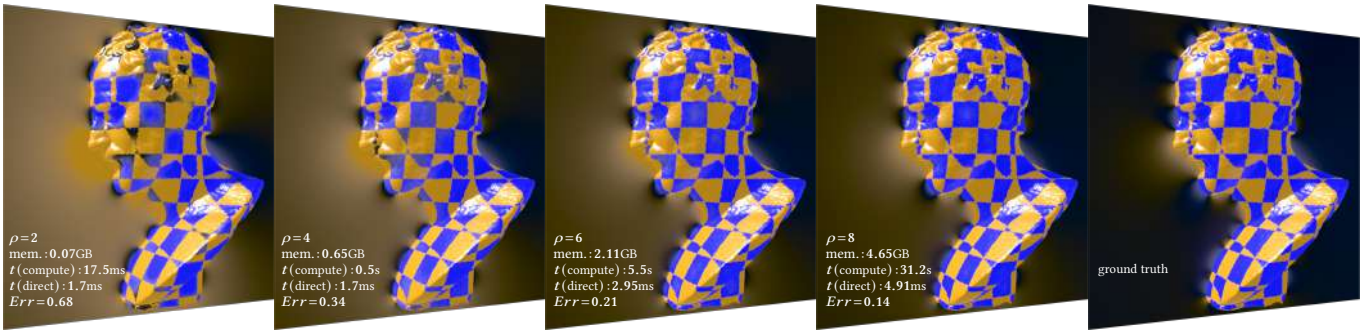


Fig. 16. **Inadequacy of the sparse inverse-Cholesky factor for direct solve.** Using $\boldsymbol{L}_{S_j,j}\boldsymbol{L}_{S_j,j}^{\mathsf{T}}$ to directly solve Eq. (5) is very fast, but its accuracy is rarely sufficient. Increasing the non-zeros of the sparsity pattern (by increasing ρ) does achieve better accuracy, but at the price of a rapid growth of memory consumption and computational cost. Thus we prefer using $\boldsymbol{L}_{S_j,j}\boldsymbol{L}_{S_j,j}^{\mathsf{T}}$ 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 $\boldsymbol{L}_S$-preconditioned system of size $B\times B$ is defined as

$$\kappa_{\text{Kap}} = \frac{1}{B}\frac{\text{tr}(\boldsymbol{K}\boldsymbol{L}_S\boldsymbol{L}_S^\intercal)}{\det(\boldsymbol{K}\boldsymbol{L}_S\boldsymbol{L}_S^\intercal)^{\frac{1}{B}}}.$$

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

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

The inequality holds because of both the inequality of arithmetic and geometric means and of Cauchy-Schwarz inequality. The minimum is reached when $\boldsymbol{L}_{S_j,j}$ and $\boldsymbol{K}_{S_j,S_j}^{-1}\mathbb{e}_j$ is collinear and the values of $\boldsymbol{L}_{S_j,j}^\intercal\boldsymbol{K}_{S_j,S_j}\boldsymbol{L}_{S_j,j}$ are the same for all j. Thus, $\boldsymbol{L}_{S_j,j}^\intercal\boldsymbol{K}_{S_j,S_j}\boldsymbol{L}_{S_j,j}$ 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

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

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

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

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

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

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

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

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

$$\boldsymbol{L}_{S_j,j}^\intercal\boldsymbol{K}_{S_j,S_j}\boldsymbol{L}_{S_j,j}-1=\frac{\boldsymbol{U}_{j,j}^4}{(1+\lambda_j)^2}-1=0.$$

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 $(\boldsymbol{L}_S\boldsymbol{L}_S^\intercal)^{-1}$ respectively. For these two covariance matrices, the KL divergence reduces to

$$\begin{aligned}
&\mathcal{D}_{KL}\left(\boldsymbol{K},\left(\boldsymbol{L}_S\boldsymbol{L}_S^\intercal\right)^{-1}\right)\\
&=-\log\det\left(\boldsymbol{K}\boldsymbol{L}_S\boldsymbol{L}_S^\intercal\right)+\text{tr}\left(\boldsymbol{K}\boldsymbol{L}_S\boldsymbol{L}_S^\intercal\right)-B\\
&=-\log\det\left(\boldsymbol{K}\right)-\log\det\left(\boldsymbol{L}_S\boldsymbol{L}_S^\intercal\right)+\text{tr}\left(\boldsymbol{L}_S^\intercal\boldsymbol{K}\boldsymbol{L}_S\right)-B\\
&=\sum\nolimits_j\boldsymbol{L}_{S_j,j}^\intercal\boldsymbol{K}_{S_j,S_j}\boldsymbol{L}_{S_j,j}-2\sum\nolimits_j\log\left(\boldsymbol{L}_{j,j}\right)+\text{const.}
\end{aligned}\tag{22}$$

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

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

Again, Eq. (7) is the solution of Eq. (23). In fact, when $(\boldsymbol{L}_S\boldsymbol{L}_S^\intercal)^{-1}$ 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.

快速基函数解法

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

Figure 1: 加速基函数解法。由于我们的工作允许通过高效的逆 Cholesky 预处理器，迅速解决来自 MFS 和 BEM 的密集线性方程，边界型方法现在可以用来处理大规模三维问题，正如这个说明性场景所展示的，包括拉普拉斯方程（从塔辐射的颜色）、线性弹性（例如，通过操纵盒子来变形老鹰），以及赫尔姆霍兹方程（在岛屿上散射输入波），这些示例涉及从 $10K$ 到 $200K$ 自由度，在笔记本电脑上计算得出。

基本解法（MFS）及其相关的边界元素法（BEM）在计算机图形学中越来越受欢迎，因为它们提供的维度更低：对于三维线性问题，它们只需要域边界上的变量即可求解和评估整个空间的解，使其成为各种应用中的宝贵工具。然而，MFS 和 BEM 的计算可扩展性较差，对大规模问题有巨大的内存需求，限制了它们在实际应用中的适用性和效率。通过利用与高斯过程的联系，并利用边界积分矩阵逆的稀疏结构，我们引入了一种变分预条件器，该预条件器可以通过稀疏逆-乔尔斯基分解以大规模并行的方式计算。我们证明，将我们的预条件器应用于预共轭梯度算法，可大大提高 MFS 或 BEM 求解的效率，在我们的系列测试中，效率提高了四个数量级。

CCS 概念:• 计算数学 → 求解器；• 计算方法 → 计算机图形学。
其他关键词和短语：基本解法、逆 Cholesky 因子分解、预处理、高斯过程

1 引言

当处理线性偏微分方程 (PDE) 时，如果域边界 $\Omega \subset \mathbb{R}^3$ 上存在给定的值，则可以使用基本解法 (MFS) 或边界元法 (BEM)。这两种方法都基于所涉及线性算子的格林函数，提供了一种仅使用基于表面的自由度 (DoFs) 求解 PDE 的方式，而不需要对 3D 域进行体积离散化，也无需执行相关的更大规模的线性求解。因此，这种“边界积分”方法显著降低了问题的维度，并消除了生成细体积网格的复杂性。正因如此，它被广泛应用于一系列变量数量较少的计算机图形学 (CG) 问题中，例如弹性 [James 和 Pai 1999]、数字雕刻 [De Goes 和 James 2017]、扩散曲线 [Bang 等人 2023] 和水波 [Schreck 等人 2019]。

然而，边界积分方程 (BIE) 中的矩阵，无论是通过 BEM 还是 MFS 方法得到的，在边界离散化后都变得完全填充且条件不佳，这对直接求解器提出了重大挑战：仅是组装 BIE 密集矩阵的所有条目就会

迅速耗尽大规模问题的内存；更糟糕的是，分解这个密集矩阵的立方复杂度常常使得即使对于中等规模的问题，直接求解也变得不切实际。迭代求解器的情况也并不轻松：BIE 矩阵的条件数往往会随着自由度 (DoFs) 数量的增加而恶化。即使是对于对称正定矩阵，共轭梯度 (CG) 求解器在处理大型问题时有时也难以快速达到足够低的误差范数，而对于非对称的 BIE 矩阵，则需要基于 GMRES 或 BiStabCG 的求解方法，这些方法甚至可能因为缺乏对该类矩阵的收敛性保证而无法收敛。由于 BIE 矩阵的密集性质在实践中对内存和计算工作提出了很高要求，随机方法因其无需网格和避免全局求解的特点而在最近受到广泛关注 [Sawhney 和 Crane 2020；Sawhney 等人 2022；Sugimoto 等人 2023]。这些蒙特卡罗方法能够快速提供逐点解的预览，并且可以说是目前唯一可用于处理数十万个自由度问题的方法。然而，其较慢的收敛速度和逐点评估的冗余性限制了它们在图形和仿真任务中的应用，这些任务通常需要在整个域内评估解且对精度要求极高，因此尽管 MFS 和 BEM 的扩展性有限，仍然被视为可行的替代方案。

从某种意义上说，这种长期存在的数值难题与线性方程组求解中经常遇到的问题截然相反：后者源于线性微分方程的离散化，通常产生由体积离散化引起的较大规模稀疏矩阵（由于导数的局部性），并且已经提出了多种有效的预处理器 [Zhu 等人，2010；Krishnan 和 Szeliski 2011；Chen 等人 2021b；Wu 等人 2022；Shao 等人 2022]。如上所述，格林函数（即在奇异载荷作用下微分方程的解）可以帮助我们写出仅涉及边界变量的更紧凑线性系统；但在这种情况下，得到的矩阵是密集的。因此，传统为稀疏微分算子设计的数值预处理方法（如多重网格、不完全 LU 或 Cholesky 分解）在此情境下无效。

贡献。在本文中，我们从高斯过程的最新研究成果中获得灵感（其协方差矩阵与 BIE 矩阵具有相似特性），提出了一种适用于密集、正定、对称矩阵的边界积分方程类型 $\mathbf{Ks} = \mathbf{b}$ 的高效预处理器。我们阐述了基于变分多尺度并行的预处理器原理，其中通过最小化预处理矩阵的 Kaporin 条件数（传统条件数的变体）来更准确地预测达到收敛所需的预处理共轭梯度迭代次数。在实际操作中，预处理通过计算 $\mathbf{K}$ 的稀疏逆 Cholesky 因子实现，采用大规模并行计算方式；在每次 CG 迭代中执行两次稀疏矩阵-向量乘积（而非传统不完全 Cholesky 预处理中的回代），相较于标准 CG 迭代成本仅有较小的边际开销，却能显著减少收敛所需的迭代次数。我们展示了在多种图形应用中，该预处理器能够显著加速 BIE 求解（对复杂和大规模问题，提速达四个数量级），从而有效应对基本解法的可扩展性挑战。最后，我们讨论了此项贡献与高斯过程之间的联系，展示了如何通过该方法直接量化结果的不确定性。

2 背景及相关工作

我们首先简要回顾了 MFS 和 BEM，然后提到了采用这些方法的一些实际 CG 应用，并讨论了为加速求解而提出的数值方法。

2.1 MFS/BEM 的连续根源

假设我们想在域 $\Omega \subset \mathbb{R}^3$ 中找到一个函数 u ，它满足形式为 $\mathcal{L}u(\mathbf{x}) = 0$ 的偏微分方程 (PDE)，其中 $\mathcal{L}$ 是线性算子，对于狄利克雷边界条件 $u(\mathbf{y})|_{\mathcal{M}} = b(\mathbf{y})$ ，其中 $\mathcal{M}$ 是 Ω 内的一个 (闭或开) 二维流形，而 $b(\cdot)$ 是一个给定的函数，它定义在 $\mathcal{M}$ 上，解 u 必须与之匹配。进一步假设我们知道 $\mathcal{L}$ 的格林函数 $G(\mathbf{x}, \mathbf{y})$ ，表示在点 $\mathbf{y}$ 处受奇异载荷作用的 PDE 的解，即满足 $\mathcal{L}G(\mathbf{x}, \mathbf{y}) = \delta(\mathbf{x}, \mathbf{y})$ ，其中 $\delta(\mathbf{x}, \mathbf{y})$ 是狄拉克 函数。有了我们线性 PDE 的格林函数，我们可以通过叠加原理，在 $\mathcal{M}$ 上的积分方程而不是在整个域 Ω 上求解 u ，因为多个 PDE 解的和仍然是一个解。因此，我们寻求 $\mathcal{M}$ 上奇异载荷的密度 σ ，使得所有格林函数在任何点上的积分正好满足边界条件，以得到正确的解，即必须找到密度 σ ，使得:

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

一旦找到满足方程 (1)(简称 “边界积分方程”，简称 BIE) 的密度，那么任意点 $\mathbf{x}$ 在 Ω 中的解 u 都可以通过简单地计算格林函数的无限和来表示:

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

换句话说，我们通过两步方法解决一个三维问题，其中我们需要先解决一个基本的二维问题——边界积分方程 (BIE)，然后才能高效地求解任意位置的解。请注意，我们上面使用了一个狄利克雷问题，但同样的方法也适用于诺伊曼边界条件，仍然得到边界积分方程和评估方程，但表达式有所变化，因为它们现在涉及格林函数的法向导数。因此，积分公式以上在许多情况下都很有用。例如，在弹性力学中，如果边界函数 b 表示均质体边界 $\mathcal{M}$ 上的位移场，那么满足方程 (1) 的密度 σ 将表示边界上的牵引力，这使我们能够通过方程 (2) 计算 $\mathcal{M}$ 内部的位移；类似地，在静电学中，沿着 $\mathcal{M}$ 的点电荷密度 σ 与输入边界静电势相匹配，足以重建整个域 Ω 上完整的电势场。

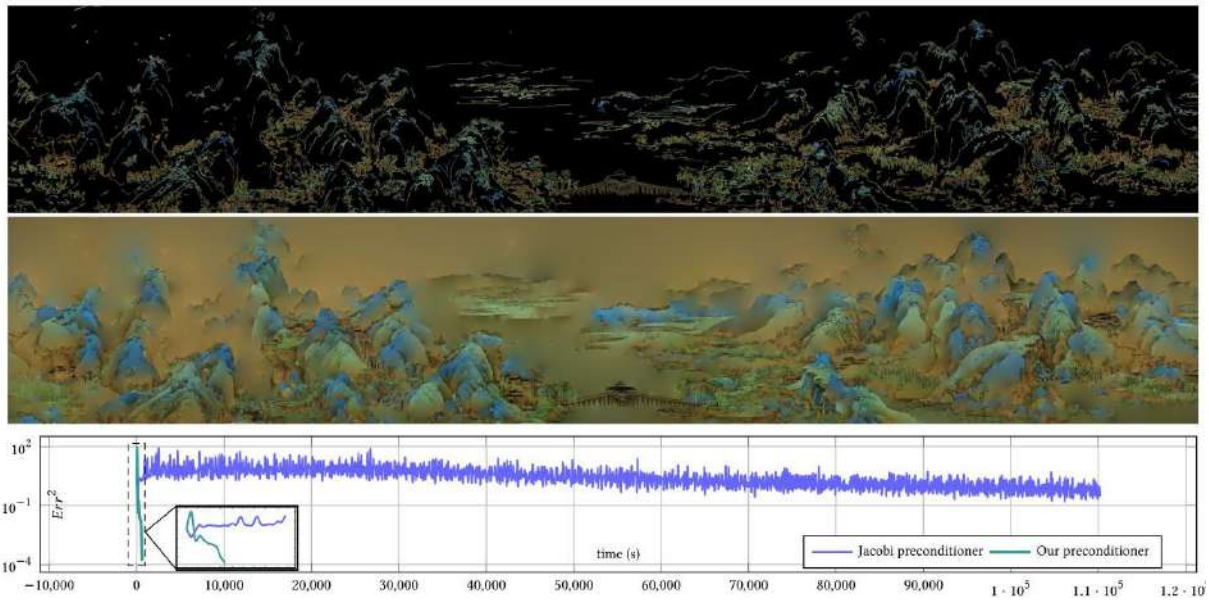


Figure 2: 大规模像素扩散。我们使用了 $1.35M$ 个边界点和 $8.4M$ 个目标点，以获得最终的图像分辨率 6957×1207。我们的算法仅需 9GB 内存来存储 K_{S_i,S_j} 和 $\mathbf{L}_S$，在稀疏参数 $\rho = 6$ 的情况下。预计算、组装及预处理的因式分解时间分别为 112.4 秒、11.3 秒和 1775.8 秒。只需 9 次 PCG 迭代便可使每个颜色通道的相对误差达到约 0.01。最后，FMM 在最终评估中扩散颜色所需的时间为 1049 秒。若没有我们的预处理器，解决这样的边界积分方程将非常耗时，甚至可能无法使用传统的预处理器达到如此低的误差。

2.2 边界积分方程的离散化

在实践中，我们只有边界点 $\{\mathbf{y}_i\}_{i=1..B}$ 位于 $\mathcal{M}$ 上，这些点编码了（逐点采样或局部积分的）边界条件，以及一组（通常更大）的目标点集 $\{\mathbf{x}_i\}_{i=1..T}$，这些点位于 Ω 内需要进行解的评估。出于计算目的，上述边界积分必须离散化，因此我们还添加了源点 $\{\mathbf{z}_i\}_{i=1..S}$，这些点包含一组数值 $\{s_i\}_i$，表示密度函数 σ 在 $\mathcal{M}$ 上的分布。

边界元素法。BEM 方法通过构建离散边界积分方程，假设边界点是表面 $\mathcal{M}$ 的三角网格离散化的一部分，源点也是如此。例如，使用与每个顶点相关联的线性基函数（比如，ϕ_i 对于 $\mathbf{y}_i$，ψ_i 对于 $\mathbf{z}_i$），Galerkin 有限元离散化将边界函数表示为 $b(\mathbf{y}) = \sum_i b_i \phi_i(\mathbf{y})$，其中 $b_i = b(\mathbf{y}_i)$，从而得到以下形式的 BIE：

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

一旦通过精确的求积方法 [Duffy 1982] 预先计算出等式 (3) 中的积分，我们确实得到了一个大小为 $B \times S$ 的线性系统（我们也称之为 BIE），基本上将未知的离散源强度 $\{s_j\}_j$ 与已知的边界值 $\{b_i\}_i$ 相关联。需要注意的是，这个关于 BEM 的简短总结远非完整，因为根据解或其导数在 $\mathcal{M}$ 上的连续性，存在 BEM 的多种变体：我们只解释了“对于 Dirichlet 问题的单层 BEM”的推导，但类似的方法可以处理 Neumann 问题，甚至是 Dirichlet 或 Neumann 问题的“双层 BEM” [Costabel 1987]。在我们的工作中，只需要知道这些离散化方法都会导致一个需要求解的密集线性系统，涉及格林函数及其导数。

基本解法。在此阶段，通过重新审视 BEM 推导，并将等式 (3) 中的基函数 ϕ_i 和 ψ_j 取为 Dirac delta 函数，可以很容易地解释 MFS；然后 BIE 简化为一个线性系统：

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

注意，在这种情况下，甚至不需要网格: 点采样就足够了。这种推导的简单性使得这种方法在计算机图形学中变得相当常见。

通用 BIE。正如我们所见，有多种方法可以推导出 BIE 的离散概念。无论最终要解决的 (离散)BIE 是通过 BEM 还是 MFS 推导出来的，我们都会将其表示为

$$Ks = b \tag{5}$$

其中 $\mathbf{K}$ 是一个大小为 $B \times S, \mathbf{s}$ 的稠密矩阵，s_j 是所有源强度的向量，我们需要求解它，$\mathbf{b}$ 是所有给定边界值的向量 b_i 。我们的论文专注于如何通过预调优共轭梯度算法的几次迭代高效地求解这个 BIE。

进一步假设。因为我们致力于在 CG 中的应用，我们将假设矩阵 K 是对称矩阵，因为在这种情况下数值求解器通常更高效。这可能被视为一种限制：实际上，在我们上面提供的解释中，只有当 Green 函数对称（这在各向同性情况下总是成立）且源样本和边界样本相同时，方程 (4) 中的基于 MFS 的 BIE 似乎是对称的。然而，可以证明，当不仅源样本和边界样本重合，而且它们的基函数也相同时，使用来自方程 (3) 的单层势的 BEM-Dirichlet 方法也会导致对称的 BIE 矩阵；同样，使用涉及 Green 函数的二阶导数 $\frac{\partial^2 G(\mathbf{x},\mathbf{y})}{\partial n(\mathbf{x}) \partial n(\mathbf{y})}$ 的双层势的 BEM-Neumann 方法在这种情况下也会导致对称矩阵。对于所有其他情况，K 可能不是对称的，但仍然可以通过求解 $\left(\mathbf{K}^\top \mathbf{K}\right) \mathbf{s} = \left(\mathbf{K}^\top \mathbf{b}\right)$ 使用最小二乘解；因此，我们的通用方程 (5) 仍然有效，现在 K 可能来自 Green 函数的内积。因此，在本文的其余部分，我们将假设边界点和源点是相同的 $(\mathbf{y}_i \equiv \mathbf{z}_i)$，并且我们的矩阵 $\mathbf{K}$ 的大小为 $B \times B$。

此外，格林函数 $G(\mathbf{x}, \mathbf{y})$ 的一个典型问题是其在 $\mathbf{x} = \mathbf{y}$ 时会出现奇异性点: 矩阵 $\mathbf{K}$ 的对角线可能因此有涉及 $G(\mathbf{y}_i, \mathbf{y}_i)$ 的不确定项。处理这个问题的典型方法是“正则化”格林函数。这可以通过找到无奇异性函数 $G^\varepsilon(\cdot,\cdot)$ 实现，使得 $\mathcal{L}G^\varepsilon$ 是平滑的狄拉克函数 [Cortez 2001]，或者通过定义 G^ε 以匹配 G ，几乎在除围绕奇点的半径为 $\varepsilon \ll 1$ 的小圆盘之外的每处，以去除奇点。另一种已知的方法是不通过正则化，而是稍微偏移源点，使它们不正好位于边界点上；但这种常见的方法使得矩阵不对称，因此需要最小二乘解法或特定解法。所以我们将假设本文中使用了正则化的格林函数 G^ε 。

2.3 评估解决方案

在我们的离散设置中，一旦从 BIE(方程 (5)) 中求解出源值 $\{s_i\}_i$ ，我们就可以通过简单的矩阵-向量乘法最终评估每个目标点 $\mathbf{x}_k$ 的解值: 对于 MFS，只需简单地评估

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

这仅仅是一个评估，并不是求解，但它的计算仍然可能非常昂贵，因为通常情况下，$G(\mathbf{x}_k, \mathbf{y}_j)$ 项都不为零。然而，由于格林函数通常快速衰减，这种评估可以通过快速多极方法 (FMM) 算法 [Greengard 和 Rokhlin 1987] 非常高效地实现，该算法通过自适应评估显著降低了这种大规模求和的复杂性。人们还可以利用 FMM 的代数版本，其中矩阵 K 被分割，然后通过分块的低秩子矩阵近似为层次矩阵 (或 $\mathcal{H}$ -矩阵 [Hackbusch 1999; Hackbusch 和 Khoromskij 2000])，以加速矩阵-向量乘法。

2.4 计算机图形学相关研究

许多图形论文利用了边界元素方法 (BEM)、多极子方法 (MFS) 或仅仅是格林函数带来的变量数量 (以及因此的矩阵求解) 的降维。在动画领域，模拟可变形体 [James 和 Pai 1999；Sugimoto 等人 2022]、流体 [Da 等人 2016] 甚至铁磁流体 [Huang 和 Michels 2020] 的快速方法已经利用了三角形网格上的 BEM 来加速计算。MFS 在波浪动画 [Schreck 等人 2019]、用于弹性的多边形有限元 [Martin 等人 2008]、声学 [James 等人 2006] 以及各种点集重建方法 [Carr 等人 2001；Zhong 等人 2019] 中得到了应用，而格林函数 (如果添加边界约束，MFS 也是如此) 是 De Goes 和 James [2017] 的交互式雕塑工作的关键。各种线性求解器被用来求解 BIE，通常是奇异值分解 (例如，[Schreck 等人 2019]) 或 GMRES(例如，[Bang 等人 2023])，而近期的工作都采用了某种形式的基于 FMM 的评估技术，因为其广泛的适用性和有效性 (例如，[Zhong 等人 2019；Bang 等人 2023])。然而，我们注意到，大多数得到的 BIE 都是过度约束的，因为作者们倾向于减少源的数量 (因此，仅近似边界条件而不是精确拟合它们)，以提供更快的求解: 实际上，如果使用了 FMM 评估，限制 MFS/BEM 方法更广泛使用的瓶颈是 BIE 求解，这对于非常大的矩阵 (即，对于直接求解器具有立方复杂度) 目前来说是过于昂贵的。我们的工作表明，如果正确利用矩阵 K (在某种意义上是表示 PDE 的线性算子的稀疏线性矩阵的逆) 的底层结构，使边界和源点重合实际上可能比仅仅节省源点更有效。尽管近期的工作从未使用超过 16,000 个源来保证可接受的时间，我们将展示如何将 BIE 求解扩展到数百万个自由度。

2.5 预处理相关研究

相对于大量关于预处理大型稀疏线性系统的工作，只有少数研究尝试对 BIE 线性系统进行预处理 (即涉及密集、对称和正半定矩阵的系统)。早期工作包括基于基变换的 [Steinbach 和 Wendland 1998] 以及 [Schippers 1985]，后者提出针对几个特定应用定制基于多网格的预处理方法，以有效解决基于对矩阵 $\mathbf{K}$ 进行逐案例分解为两个矩阵和使用求积评估推导出使多网格预处理剂有效的松弛方案。当使用 $\mathcal{H}$ -矩阵进行快速矩阵-向量乘法时，通过对 BIE 矩阵进行 LU 分解 [Kriemann 2013] 或通过嵌套 GMRES-based 构建 [Amlani 等人 2019]，可以更有效地实现 BIE 矩阵的预处理，性能改进报告称大约提高了两倍；然

而，在 $\mathcal{H}$ -矩阵中使用的低秩近似的精度会显著限制效率，而追求更高精度会削弱层次矩阵的好处 - 因此需要参数调整以获得良好结果。另一种 BIE 预处理器在 [Beatson 等人 1999] 中提出，特别设计用于 RBF 拟合。

然而，我们的方法具有普遍性，适用于大多数偏微分方程 (PDEs)(例如，椭圆或抛物方程，甚至对于一些非局部偏导数方程，只要格林函数衰减迅速即可)。我们的数值方法利用了高斯过程 (GP) 的最新发展，其中在 GP 统计中使用的协方差矩阵实际上是与我们上下文中的格林导出矩阵等价的。Vecchia [1988] 的早期工作通过一系列单变量条件密度的乘积来近似高斯分布的似然函数，每个密度依赖于先前排序变量中的一个子集。后来发现这等价于计算协方差矩阵的近似逆-cholesky 因子 [Katzfuss 和 Guinness 2021]；另外，Kaporin [1994] 推导了一个稀疏矩阵的近似逆-cholesky 预调条件子的闭式表达式，该表达式在稀疏性约束下最小化特定预调系统的条件数。Kaporin 的方法后来被证明等价于在对角缩放约束下最小化 Frobenius 范数目标 [Yeremin 等人，2000]。Schäfer 等人 [2021a] 随后证明了 Kaporin 的预调条件子和 Vecchia 近似是等价的，并且可以通过稀疏约束的 Kullback-Leibler(KL) 最小化获得；他们还提出了一种格林函数行列的重新排序和稀疏模式，以创建一个端到端的求解器，从边界值推导目标值，其复杂度可证明是对数线性的。在本文中，我们利用 Kaporin 的逆-cholesky 因子的变分定义，但应用于稠密矩阵，并引入了一种大规模并行的边界积分方程 (BIE) 预调条件子的实现。我们在各种图形应用中展示了时间可以轻易提高一到四个数量级，从而解锁了 MFS/BEM 高效处理非常大问题的潜力。

2.6 概述

本文的其余部分专注于高效地解决边界积分方程 (5)。在快速回顾了我们的方法所基于的 Kaporin 条件数的稀疏约束最小化闭式之后，我们将详细介绍 $\mathbf{K}$ 的预条件器，写成逆 Cholesky 因子的稀疏近似与其转置的乘积，这样在实际中只需几次预条件共轭梯度迭代即可解决 BIE(见图 3)。然后，我们将展示我们的快速 BIE 求解器的几个应用，证明其大规模并行特性在实际中带来了巨大的加速。

3 变分多尺度预处理器

我们现在详细说明如何计算，给定一个下三角稀疏模式 $\mathcal{S}$ ，一个稀疏逆 Cholesky 因子 $\mathbf{L}_{\mathcal{S}}$ 的 $B \times B$ 矩阵 $\mathbf{K}$ 来近似 $\mathbf{K}^{-1}$: 得到的近似因子 $\mathbf{L}_{\mathcal{S}}$ 可以用来形成一个低成本且有效的预处理器 $\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}$ ，在求解方程 (5) 时大大加速 PCG 的收敛。

3.1 Kaporin 的变分逆-Cholesky 因子

虽然实际的下半三角逆 Cholesky 因子 L 使得 $\mathbf{L}\mathbf{L}^{\top} = \mathbf{K}^{-1}$ 的计算速度较慢，但为了提高效率，人们总是可以寻找一个不完整的逆 Cholesky 因子 $\mathbf{L}_{\mathcal{S}}$ ，这是给定稀疏模式 $\mathcal{S} := \{(i,j) \mid i \geq j$ 和 $(\mathbf{L}_{\mathcal{S}})_{ij} \neq 0\}$ 下对 $\mathbf{L}$ 的稀疏近似。一种直接的方法是找到这样一个矩阵 $\mathbf{L}_{\mathcal{S}}$ ，以最佳近似 $\mathbf{K}^{-1}$ ，即最小化 $\left\|\mathbf{I} - \mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}\mathbf{K}\right\|_F$ 并受稀疏模式 $\mathcal{S}$ 的约束，因为这可以提供对矩阵 K 的优化预调条件。然而，这种 Frobenius 范数最小化问题需要求解非线性方程，其计算强度要高于我们打算求解大规模问题的初始线性系统。

Kaporin [1994] 发现了一个不同优化问题的简单、封闭形式解，这在我们的情境中非常有用，因为它可以以大规模并行的方式实现。首先，他证明了现在所谓的 Kaporin 条件数 κ_{Kap} ——即矩阵的特征值的算术平均值与几何平均值之比，而不是标准条件数 κ (其计算为最大特征值与最小特征值之比) ——对于估计 PCG 在给定的容差范围内达到收敛所需的迭代次数，给出了比 κ 更紧的上界 ϵ 。其次，他证明了在给定稀疏模式下，使 Kaporin 条件数 $\kappa_{\text{Kap}}\left(\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}\mathbf{K}\right)$ 最小的矩阵 $\mathbf{L}_{\mathcal{S}}$ 可以用矩阵 K 的小子块逆的封闭形式逐列表示: 如果我们用 $\mathcal{S}_j := \{i \mid (i,j) \in \mathcal{S}\}$ 表示 $\mathbf{L}_{\mathcal{S}}$ 的某一列 j 的稀疏模式 (即包含在该列非零元素的行索引)，那么 $\mathbf{L}_{\mathcal{S}}$ 的 $\mathbf{L}_{\mathcal{S}_j,j}$ 列可以独立于其他列表示为:

$$L_{\mathcal{S}_j,j} = \frac{K_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\widehat{\mathbf{e}}_j}{\sqrt{\widehat{\mathbf{e}}_j^{\top}K_{\mathcal{S}_j,\mathcal{S}_j}^{-1}\widehat{\mathbf{e}}_j}}, \ \forall j = 1..B, \tag{7}$$

其中 $K_{\mathcal{S}_i,\mathcal{S}_i}$ 是 $\mathbf{K}$ 的子矩阵，其行和列的索引在 $\mathcal{S}_j$ 中，而 $\mathbf{e}_j = (1,0,\ldots,0)^{\top} \in \mathbb{R}^{|\mathcal{S}_j|}$ 是长度为 $|\mathcal{S}_j|$ 的单位向量。这种 Kaporin 构造的 $L_{\mathcal{S}}$ ，恰好实现了 Vecchia 的近似 [Vecchia 1988]，此后又发现了两种其他变体解释 (详情见附录 A)，因此为给定稀疏性的 PCG 提供了一个最优预调条件器，如果在稀疏性与计算逆 Cholesky 因子 $L_{\mathcal{S}}$ 的时间之间找到良好的平衡，它有可能显著加速方程 (5) 的求解: 过于稀疏的约束模式可能评估非常快，但可能无法充分地调节矩阵 K^{-1} ，以保证在几次迭代中 PCG 的收敛；更好的调节可能会牺牲大量的稀疏性，这将需要更长的时间来评估。

注意，Kaporin 的工作主要应用于求稀疏矩阵的逆；由于这些矩阵通常具有密集的逆和逆-Cholesky 因子，这种预调条件的效力从根本上受到限制。最近，Schäfer 等人 [2021a] 表明，它可以用作近似密集 Green 矩阵的稀疏逆 Cholesky 因子，在对数线性成本下，这使其成为求解逆的颇有前景的方法。

本节的其余部分解决了 $\mathbf{L}_{\mathcal{S}}$ 的评估问题，其列在方程 (7) 中表示。我们将看到，我们可以使这种评估大规模并行，因为每个列 (或每组列) 可以独立评估，而且需要 $\mathbf{K}$ 的求逆子块本身可以并行执行。。还应注意，只需要 K 的少数元素 (那些在相关子块中使用的元素)，因此我们的构建甚至不需要完全组装矩阵 $\mathbf{K}$，从而通过跳过许多不必要的格林函数评估，进一步节省时间和内存。

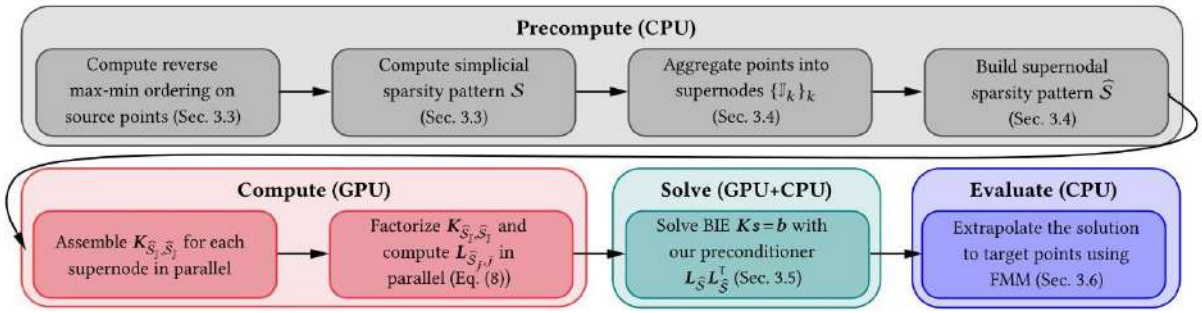


Figure 3: 概述。为了解决像等式 (5) 这样的通用边界积分方程，我们构建的近似 Cholesky 分解 L_S 对于 K^{-1} 包括一些基于 CPU 的预计算 (例如构建排序和稀疏模式)，然后是在 GPU 上组装 K 的子矩阵以大规模并行构建因子 $L_{\mathcal{S}}$；接着 $L_{\mathcal{S}}L_{\mathcal{S}}^{\top}$ 被用作预处理器，它允许预条件共轭梯度算法在几次迭代中收敛；最后在一系列目标三维点上执行基于 FMM 的评估，其中需要评估解决方案。

3.2 对 $\mathbf{L}_{S_j,j}$ 的局部 Cholesky 分解

$L_{\mathcal{S}}$ 的每一列可以通过等式 (7) 进行并行处理，通过一个小型的密集线性求解。由于 $K_{\mathcal{S}_i,\mathcal{S}_j}$ 是对称正定的，我们可以使用经典的 Cholesky 分解对 $K_{\mathcal{S}_i,\mathcal{S}_i}$ 进行因式分解，然后通过两次回代来获得 $K_{\mathcal{S}_i,\mathcal{S}_i}^{-1}\mathbf{e}_j$ 。然而，考虑到向量 e_j 的稀疏结构，如果使用 $K_{\mathcal{S}_i,\mathcal{S}_j}$ 的反向 Cholesky 分解，可以将两次回代替换为一次。更具体地说，在根据列稀疏性 $\mathcal{S}_j$ 组装 $K_{\mathcal{S}_i,\mathcal{S}_j}$ 之后，我们可以计算分解 $K_{\mathcal{S}_j,\mathcal{S}_j} = U_{\mathcal{S}_j,\mathcal{S}_j}U_{\mathcal{S}_j,\mathcal{S}_j}^{\top}$ ，其中 $U_{\mathcal{S}_j,\mathcal{S}_j}$ 是一个上三角矩阵。然后我们将等式 (7) 转化为

$$\mathbf{L}_{\mathcal{S}_j,j} = \mathbf{U}_{\mathcal{S}_j,\mathcal{S}_j}^{-\top}\mathbf{e}_j,\ \forall j = 1..B \tag{8}$$

来计算 $\mathbf{L}_{\mathcal{S}}$ 的 j^{th} 列。我们注意到计算这种逆分解不会产生额外成本: 如果 $\operatorname{chol}(\cdot)$ 表示标准的下三角 Cholesky 因子，那么 $U_{\mathcal{S}_j,\mathcal{S}_j}$ 可以通过

$$\mathbf{U}_{S_j,S_j} = \mathbf{R}_j \operatorname{chol}\left(\mathbf{R}_j\mathbf{K}_{S_j,S_j}\mathbf{R}_j\right)\mathbf{R}_j, \tag{9}$$

其中 $\mathbf{R}_j$ 是大小为 $|\mathcal{S}_j| \times |\mathcal{S}_j|$ 的置换矩阵，它将索引从 $1, 2, \ldots, |\mathcal{S}_j|$ 反转为 $|\mathcal{S}_j|, \ldots, 2, 1$ 。使用等式 (8) 代替等式 (7) 不仅简化了计算，而且我们将在第 3.4 节中看到，它还允许重用因子来计算其他列 $\mathbf{L}_{\mathcal{S}_i,j}$ ，这些列被聚集到同一个超节点中，因为 $\mathbf{U}_{\mathcal{S}_i,\mathcal{S}_j}$ (带有 $k < |\mathcal{S}_j|$) 的右下角 $k \times k$ 子矩阵也是 K_{S_j,S_j} 的右下角 $k \times k$ 子矩阵的反向 Cholesky 因子，适用于所有的 j 。

3.3 排序和稀疏模式

对于基于不完整矩阵分解的预处理器，自由度 (即行/列的顺序) 的排序和稀疏模式的选择对结果的质量起着至关重要的作用。在这项工作中，我们采用了反向最大-最小排序 P 对 $\mathbf{K}$ 进行排列，然后计算 $\left(\mathbf{P}^{\top}\mathbf{K}\mathbf{P}\right)^{-1} = \mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}$ ，正如在 [Guinness 2018; Chen et al. 2021b; Schäfer et al. 2021a, b] 中针对病态微分算子和高斯过程回归提出的那样。生成最大-最小排序是通过最远点采样实现的: 从任意边界点 $\mathbf{y}_{i_0}$ 开始，我们反复选择排序中的下一个边界点为到目前为止已选择的点中最远的一个，从而产生一系列有序索引

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

其中 $\operatorname{dist}(\cdot,\cdot)$ 是欧几里得距离。然后我们将这个最大-最小排序反向为 $\mathbf{P} = \{i_{B-1}, \ldots, i_1, i_0\}$ 以排列所有自由度。注意，尽管最大-最小排序的暴力计算将是 $O\left(B^2\right)$ ，但我们转而应用了 [Schäfer et al. 2021b, 算法 4.1] 中实现的 GPVecchia 库 [Zilber and Katzfuss 2021]，其复杂度为 $O\left(B\log^2(B)\right)$ 。

几何上，反向最大最小重排意味着边界样本在 $\mathcal{M}$ 上的多分辨率顺序 (见图 4)，其中 “粗尺度” 边界点 (在排序中出现较晚) 首先在域上展开，然后 “细尺度” 点 (在排序中出现较早) 进来填补空隙 (见

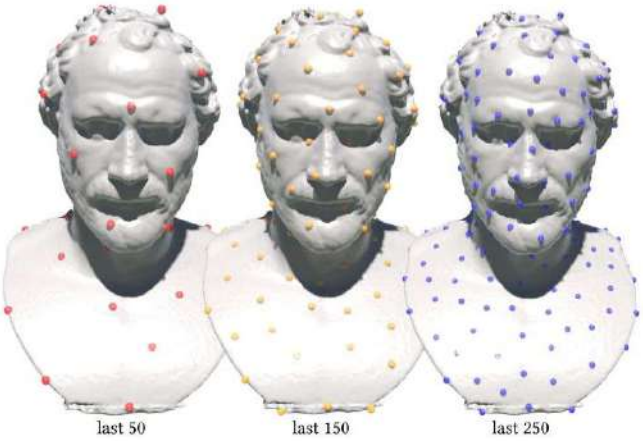


Figure 4: 反向最大-最小排序: 从几何上看，反向最大-最小排序将所有自由度 (此处为网格的顶点) 分解成不同的空间尺度，粗尺度点 (捕捉低频信息) 位于末尾，细尺度点 (处理高频细节) 位于开始。

[Chen et al. 2021b])，在通过 $\ell_{i_k} = \min_{p\in\{0,k-1\}} \operatorname{dist}\left(\mathbf{y}_{i_k}, \mathbf{y}_{i_p}\right)$ 定义的所有的点上建立长度尺度的概念，该长度尺度在反向最大最小排序中是单调递增的。由于协方差矩阵的逆-Cholesky 因子的第 k 列编码了高斯过程中第 k 个变量与排序中后续变量的条件相关性 [Katzfuss and Guinness 2021; Schäfer 2021]，反向最大最小排序确保了与第 k 个变量及其后续变量相关的空间位置在大致上均匀分布。在这种设置下，已知椭圆 PDE 的 Green 函数受到图 5 所示的屏蔽效应的影响，通过仅在一个子集上的点进行条件化，会导致与更远点的近似独立性，正如空间统计文献中长期以来所观察到的 [Stein 2002]。在 [Schäfer et al. 2021a, b] 中基于算子适应波 let 和数值均质化的先前工作，推导了关于屏蔽的严格界限，这意味着逆 Cholesky 因子的指数衰减 (见 [Owhadi and Scovel 2019; Altmann et al. 2021] 了解这些主题的概述)。这种屏蔽效应因此激发了我们选择稀疏模式: 我们根据每个点的长度尺度构建 $\mathcal{S}$ ，正如 [Chen et al. 2021b] 中为微分算子的预调优所提倡的，逆-Cholesky 因子中的非零条目被指定为

$$\mathcal{S} \;:=\; \{(i,j) \mid i \geq j \text{ and } \operatorname{dist}(\mathbf{y}_i, \mathbf{y}_j) \leq \rho \min(\ell_i, \ell_j)\}, \tag{11}$$

这意味着在稀疏性中的元素 (i, j) 如果 $\mathbf{y}_i$ 和 $\mathbf{y}_j$ 在彼此的支持半径内，该半径由 ρ 缩放，这个参数允许在预调优质量和计算成本之间进行权衡，那么该元素将被强制非零。在实际中，可以通过每列的快速范围搜索，并行地进行稀疏性计算，该搜索由 k -d 树 [Chen et al. 2021b] 支持。请注意，我们将方程 (11) 称为单纯形稀疏性，以区分我们接下来将介绍的超级节点稀疏性。

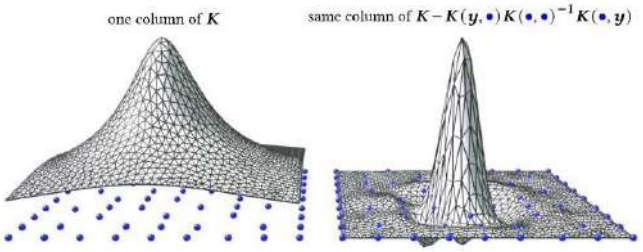


Figure 5: 屏蔽效应。协方差矩阵 K 具有长程相关性，但其由粗样本 (蓝色) 给出的 Schur 补的协方差条件是高度局部化的。K 的逆 Cholesky 因子编码了条件相关性，因此屏蔽导致了它们的近似稀疏性。

3.4 聚合分解

到目前为止，我们已经看到，矩阵 L_S 的每一列只需要根据我们选择的稀疏度，将小矩阵 $K_{\mathcal{S}_i,\mathcal{S}_j}$ 进行组装，这比组装整个矩阵 $\mathbf{K}$ 节省了时间和内存。然而，注意到逆-Cholesky 因子的多列可能需要几乎相同的稠密子矩阵 $K_{\mathcal{S}_i,\mathcal{S}_i}$，因此在其分解中出现冗余计算。通过将需要几乎相同分解的列聚合到超节点中 [Stein et al. 2004; Ferronato et al. 2015; Guinness 2018]，可以实现对 $\mathbf{K}_{S_j,S_j}$ 及其 Cholesky 分解 $\mathbf{U}_{S_j,S_j}\mathbf{U}_{S_j,S_j}^T$ 的重用，以消除显而易见的重复计算: $\mathbf{K}$ 的一个包含性子矩阵的分解将因此服务于超节点的所有列，因为每列可以从因子中提取所需的行和列。

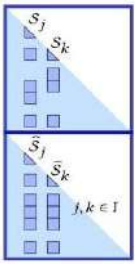
Algorithm 1 识别超级节点

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

for $j \leftarrow 1$ to B **do**
    processed$[j] \leftarrow$ false
end for
$k \leftarrow 1$

for $j \leftarrow 1$ to B **do**
    if processed$[j]$ **then**
        continue
    end if
    $\mathbb{J}_k \leftarrow \{\}$ ▷ initialize a new supernode
    for $i \in \mathcal{S}_{\rho,j}$ **do**
        if not processed$[i]$ and $\ell_i \leq \frac{3}{2}\ell_j$ **then**
            processed$[i] \leftarrow$ true
            $\mathbb{J}_k \leftarrow \mathbb{J}_k \cup \{i\}$
        end if
    end for
    $k \leftarrow k + 1$
end for

将列聚合成超节点是通过合并空间上接近的、尺度相似的索引 (见算法 1 中的伪代码) 来实现的，因为它们更有可能为其子矩阵的 K 分享相似的索引。当算法退出时，它返回一组超节点 $\{\mathbb{J}_k\}_k$ ，每个超节点由一定数量的索引组成



这些索引在反向最大最小排序中不一定是连续的——但是同一个超节点中的所有索引都将共享 (部分) 从 K 中提取的相同矩阵: 超节点将对由扩展超节点稀疏模式 $\widehat{\mathcal{S}}$ 派生出的选定索引的子矩阵进行分解。简单来说，一列的稀疏性 $j \in \mathbb{J}$ 是从属于 $\mathbb{J}$ 的所有列中收集的所有非零条目的并集；也就是说，它的稀疏性 $\widehat{\mathcal{S}}_j$ 是通过 (见插图) 定义的

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

我们可以通过以下方式定义超节点的稀疏模式:

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

图 6 说明了超节点的聚合和由此产生的超节点稀疏模式。我们表示超节点内的 K 子矩阵为 $K_{\widehat{\mathcal{S}}_\mathrm{I},\widehat{\mathcal{S}}_\mathrm{I}}$ ，并重用其逆-Cholesky 因子被用于计算所有列 $\mathbf{L}_{\widehat{\mathcal{S}}_j,j}, \forall j \in \mathbb{J}$ 。超节点方法的直接实现给出在算法 2 中，其中的三个步骤都可以大规模并行化。如图 7 所示，计算逆 Cholesky 因子随问题规模几乎线性增长，存储局部矩阵 $\mathbf{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}$ 的内存成本也是如此，因此全局逆 Cholesky 因子 $\mathbf{L}_{\widehat{\mathcal{S}}}$ 也是如此。

3.5 对 BIE 矩阵 K 进行预调

现在我们有一个快速算法来评估 L_S 给定的稀疏参数 ρ ，并且知道 $\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^\top$ 近似 $\mathbf{K}^{-1}$ ，我们可能会简单地通过 $\mathbf{b} = \mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^\top \mathbf{s}$ 计算方程 (5) 的解，这是通过两次回代轻松评估的。然而，这种直接解的准确性对于较

小的 ρ 可能不够好，而将 ρ 增加以获得更密集、更精确的分解会导致计算和内存成本迅速增长 (参见图 16 的示例)。然而，我们发现我们计算的 Kaporin 解在结果条件数 $\kappa_{\text{Kap}}\left(\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}\mathbf{K}\right)$ 上是最优的。因此，我们可以转而使用 $\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}$ 作为共轭梯度求解器的预调条件，以保证在求解 $K\mathbf{s}=\mathbf{b}$ 时的更好效率。此外，由于预调条件只需要在共轭梯度迭代的 k 次上应用于残差 $\mathbf{e}_k=\mathbf{K}\mathbf{s}_k-\mathbf{b}$ ，我们只需要应用两次低成本稀疏矩阵-向量乘法来评估 $\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}\mathbf{e}_k$ ，因此线性系统的调节开销是非常有限的。(请注意，这与 [Chen et al. 2021b] 中微分算子的不完整 Cholesky 预调条件形成鲜明对比，其中需要两次回代以执行预处理。) 在实际中，我们可以使用相对较小的 ρ 来大幅减少迭代次数: 在图 8 中，对于一个大约有 260 K 自由度的系统，$\rho=6$ 的值使得预调优共轭梯度法 (PCG) 在相对误差低于 0.03 的情况下恰好 7 次迭代收敛，而简单的 Jacobi 预调优器则需要几百次迭代。

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

Data: 超节点稀疏模式 $\{\widehat{\mathcal{S}}_{\mathbb{I}}\}_{\mathbb{I}}$, Green 函数 $G(\cdot,\cdot)$, 源点 $\{\mathbf{z}_i\}_i$
Result: 逆 Cholesky 因子 $\mathbf{L}_{\widehat{\mathcal{S}}}$, 使得 $\mathbf{K}^{-1}\approx\mathbf{L}_{\widehat{\mathcal{S}}}\mathbf{L}_{\widehat{\mathcal{S}}}^{\top}$

function AssembleLocalMFSMatrix
  for each supernode $\mathbb{J}$ **do**
    $n_{\mathbb{J}}\leftarrow\left|\widehat{\mathcal{S}}_{\mathbb{J}}\right|$
    $\mathbf{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}\leftarrow\mathbf{0}_{n_{\mathbb{J}}\times n_{\mathbb{J}}}$
    for $i\in\mathbb{J},j\in\mathbb{J}$ **do**
      $\mathbf{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}[i,j]\leftarrow G(\mathbf{z}_i,\mathbf{z}_j)$
    end for
  end for
end function

function LocalCholessyFactorize
  for each supernode $\mathbb{J}$ **do**
    compute the inverse Cholesky decomposition
    $\mathbf{K}_{\widehat{\mathcal{S}}_{\mathbb{J}},\widehat{\mathcal{S}}_{\mathbb{J}}}=\mathbf{U}_{\mathbb{J}}\mathbf{U}_{\mathbb{J}}^{\top}$
  end for
end function

function ComputeGlobalFactor
  for $j\leftarrow 1$ to B **do**
    find $\mathbb{J}$ such that $j\in\mathbb{J}$
    $n_j\leftarrow\left|\widehat{\mathcal{S}}_j\right|$
    $\mathbf{L}_{\widehat{\mathcal{S}}_j,j}\leftarrow\left(\mathbf{U}_{\mathbb{J}}\left[-n_j:,-n_j:\right]\right)^{-\top}\mathbf{e}_j$ `(Python notation)`
  end for
end function

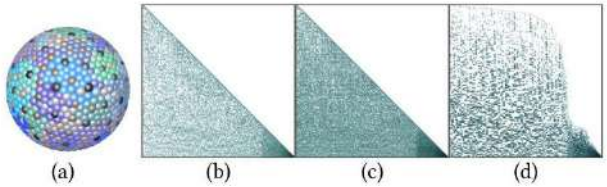


Figure 6: 聚合稀疏模式。超节点簇 (每个由相同颜色的点组成) 显示在具有 1275 个点的球体模型上 (a)，以及带有 $\rho=3$ 的单纯形稀疏模式 (b) 和超节点稀疏模式 (c)。超节点模式的重排 (d) 通过将超节点内的列连续放置 (仅用于可视化目的)。

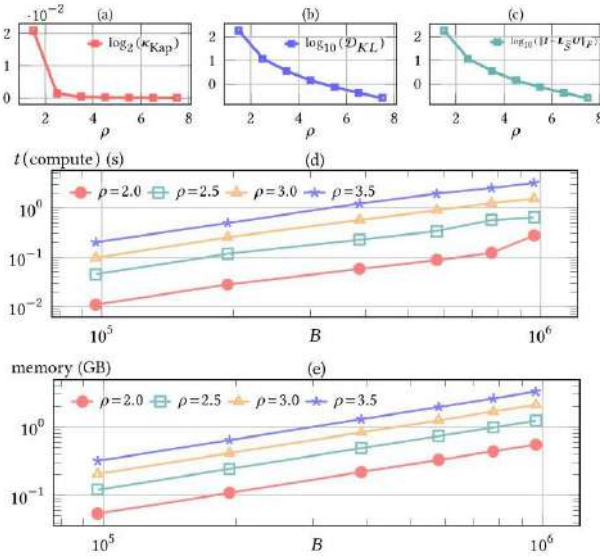


Figure 7: 渐近行为。我们在 3D 中测试了我们的算法，边界点在正方形上均匀分布。在 (a)、(b) 和 (c) 中，附录 A 中呈现的三个变分函数随着稀疏参数 ρ 的增加而减小 (因为逆 Cholesky 因子变得不那么稀疏)——特别是，Kaporin 条件数迅速下降。在 (d) 和 (e) 中，我们的基于 GPU 的逆 Cholesky 预处理计算在不同稀疏级别上显示出边界值 B 的准线性增长，无论是内存成本还是时间成本，都证实了我们的构造的可扩展性。

3.6 黑箱 FMM 用于矩阵-向量乘积

尽管我们的预处理器可以高效构建和应用，但如果需要对 Ks 进行全面评估，对于大型问题，PCG 的运行时性能将会显著降低。然而，存在大量工作致力于加速这种密集矩阵-向量乘法。在这里，我们利用在第 2 节中已经提到的快速多极方法来加速 PCG 中的密集矩阵-向量乘积。我们采用了 [Wang et al. 2021] 的黑盒快速多极方法实现，该方法仅需要提供一个在两个点 $\mathbf{x}$ 和 $\mathbf{y}$ 之间评估正则化格林函数 G^ε 的例程，而无需手动实现 FMM 操作符，如多极到局部、多极到多极等。他们的代码使用切比雪夫多项式为非振荡核构造格林函数的低秩近似，并通过奇异值分解 (SVD) 进一步压缩，将矩阵-向量乘法的复杂度从二次降低到线性。因此，它非常适合我们场景中 2D 和 3D 的各种格林函数。请注意，相同的 FMM 过程 (或者 alternatively，一个 $\mathcal{H}$ -矩阵) 可以用于通过方程 (6) 将解插值到目标点。

4 应用与结果

在这一节中，我们首先讨论一些实现细节，然后在我们方法的多种 CG 相关应用上进行测试: 我们展示了我们的预调处理器如何显著提高求解拉普拉斯方程、线性弹性方程和带边界条件的赫尔姆霍兹方程的性能。最后，探讨了 MFS 与高斯过程之间的联系，使得不确定性量化成为可能。

4.1 实现

给定边界点 $\{\mathbf{y}_i\}_i$ 和目标点 $\{\mathbf{x}_k\}_k$ ，我们将它们重新缩放，使得它们的边界框为单位大小。我们直接在 CPU 上执行一系列预计算，包括计算最大-最小顺序、识别超节点以及构建稀疏模式 ($\mathcal{S}$ 及其相关列稀疏性 $\mathcal{S}_j$)。我们开始利用 GPU 上的大规模并行性，进行稀疏逆 Cholesky 因子的列向构建以及局部矩阵 $K_{\mathcal{S}_i,\mathcal{S}_j}$ 的组装及其因式分解。在 PCG 迭代过程中，我们结合 NVIDIA ® cuSPARSE 和 cuBLAS 以优化线性代数运算，并使用来自 [Wang et al. 2021] 的基于 CPU 的黑盒 FMM 来加速矩阵-向量乘积。我们实现的源代码可在 https://gitlab.inria.fr/geomerix/public/mfs-chol 获取。

本文展示的所有示例都是在笔记本电脑上运行的 (配备 AMD Ryzen 7 5800H CPU，8 核心和 16G RAM)，并使用 NVIDIA GeForce RTX 3060 Laptop GPU，配备 6GB RAM 以证明可扩展性 - 除了图 2，其中使用了 NVIDIA RTX A6000 GPU 和 48GB 的 RAM 来处理我们最大的矩阵大小。因为我们使用的所有格林函数都是各向同性的，$G(\mathbf{x},\mathbf{y})$ 总是表示为 $r = \| \mathbf{x}-\mathbf{y} \|$ 的函数。因此，我们使用了一种简单的正则化方法，即在表达式中将 r 改为 $r_\varepsilon = \sqrt{r^2+\varepsilon^2}$ ，其中 $\varepsilon = 10^{-5}$ (在 $\left[10^{-6}, 10^{-4}\right]$ 范围内的任何值都会提供视觉上相似的结果) - 但也可以使用其他正则化方法。我们方法中唯一的参数是 ρ ，用于构建稀疏模式 $\mathcal{S}$ 。由于我们处理的是采样超曲面的边界点，我们发现对于 2D 问题，ρ 应该设置得相对较大 (例如，大约 8)，而对于 3D 问题，应该使用较小的 ρ (例如，大约 5)，以在计算成本和预调优质量之

间取得良好的平衡。请注意，我们的预调优器的最终质量不仅取决于 ρ ，还取决于边界点的空间分布: 如果存在非常密集的边界点区域，ρ 可以适度减小。最后，我们使用相对误差 $Err = \| Ks - b \| / \| b \|$ 来衡量我们求解器的精度。

4.2 拉普拉斯方程

拉普拉斯方程 $\Delta u = 0$ 在施加边界条件后，一直是 CG 应用的主要部分。满足此偏微分方程的函数被称为调和函数，拉普拉斯算子的格林函数是

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

其中 $r = \| \mathbf{x} - \mathbf{y} \|$ 。我们通过展示“扩散像素”的例子，证明了我们的预调质子在拉普拉斯方程上的有效性，这些扩散像素可以被视为一种近似的无网格版本用于图像合成的扩散曲线 [Orzan et al. 2008; Sun et al. 2012]。对于给定数量的颜色像素 $\{\mathbf{y}_i\}_i$ 以及指定的颜色 $\{\mathbf{b}_i\}_i$ (包括三个 RGB 通道作为狄利克雷边界条件)，我们可以通过 PCG 求解方程 (5) 来获得每个源点的“颜色电荷” $\mathbf{z}_i \equiv \mathbf{y}_i$ 。最后，这些颜色电荷被外推以将边界颜色扩散到整个图像，这可以用来创建矢量图形 (通过计算更多的目标点轻松放大)，或者作为压缩真实世界照片的一种方式。在本文的每个示例中，我们采用输入图像，通过基本的坎尼边缘检测器滤波器提取其“边缘像素”(及其颜色)，并使用这些像素及其四个紧邻的邻居 (分别及其颜色) 作为基于拉普拉斯的边界积分方程的边界点 (分别及其狄利克雷边界颜色)。我们的解决方案因此是一种边界颜色的谐和混合，能够很好地再现原始图像。

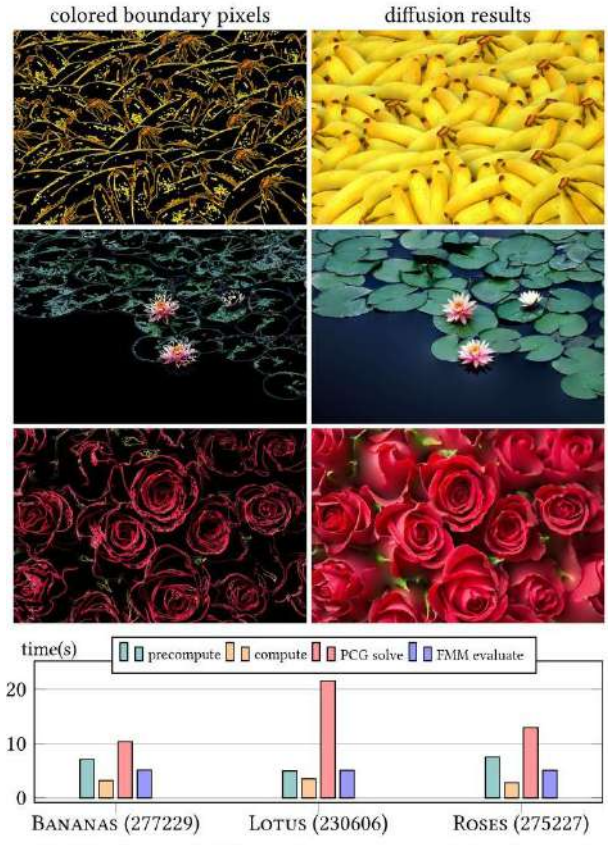


Figure 8: 扩散像素。我们为三种“像素扩散”示例的算法提供了分阶段的时间。用于扩散的输入彩色像素数 (即边界点) 在括号中显示。最终图像大小为 1280×853 ，即超过 100 万个目标点进行评估。在这里，稀疏模式使用 $\rho = 6$ 。PCG 时间包括解决三个 RGB 通道，每个通道使用 7 次 PCG 迭代。请注意，使用较少的迭代次数会更快，且产生的误差几乎不可察觉。

从这些边界颜色像素重建高分辨率图像很容易导致非常大的边界影响误差 ($\sim 270,000^2$ 对于图 8) 以及边界条件 (即指定的颜色) 通常涉及广泛的频率——这两个事实对传统的直接或迭代求解器提出了巨大的挑战。我们的预处理器能显著解决这些困难: 在图 8 中，我们分解了我们的方法在三个 1280×853 图像上的时间成本，这些图像的 BIE 矩阵大小超过 $230\text{ K} \times 230\text{ K}$ ，目标点数量超过 $1M$ ，总共大约 30 秒在我们的笔记本电脑上生成最终的扩散图像。对于每个 RGB 通道仅进行 7 次 PCG 迭代，平均相对误

差在 10^{-3} 到 10^{-2} 之间。在图 9 中，我们比较了在较小的 640×480 大小的示例上，PCG 使用我们的预处理器与简单的 Jacobi(对角) 预处理器性能。我们的方法迅速减少了误差，仅迭代两次后扩散结果几乎无法区分，而基于 Jacobi 的 PCG 需要多两个数量级的迭代才能达到可比的误差水平。我们进一步在图 2 中的更大图像上测试了我们的算法，BIE 求解有 $1.35M$ 个自由度，$8.4M$ 个目标点进行评估。仅需要 9 次迭代就能将误差减少到大约 1%，仅使用 9GB 的 GPU 内存——而简单的共轭梯度甚至在 1.5 K 次迭代和超过一天后都无法收敛。最后，我们在图 10 中测试了一个 3D 扩散案例。由于 MFS 是无网格的，我们不需要流形或单连通网格，所以我们的扩散是基于一个多孔网格上的绘制标量场，该标量场向两个灰色墙壁发射热量。对于大约 240 K 个自由度和 524 K 个目标点 (在这个灯照亮的两个墙壁上)，总共不到 20 秒就能求解和评估解决方案，需要 7 次 PCG 迭代以达到小于 10^{-2} 的相对误差。注意到与之相比，最近关于扩散图像的论文 (例如，[Bang 等人 2023]) 通常将他们的示例限制在 BIE 小于 $16K$ 个自由度——并且仅因光源点较少而近似边界条件。

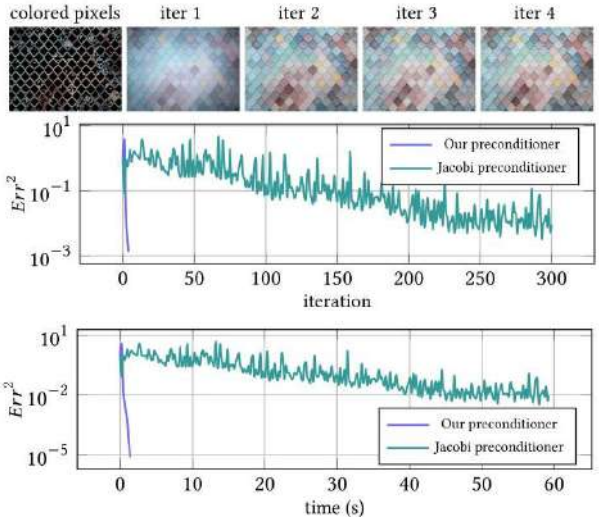


Figure 9: PCG 收敛性。此示例使用 109 K 彩色像素作为狄利克雷边界条件。应用我们基于逆 Cholesky 的预处理器 (带有 $\rho = 4$) 显著加快了收敛速度，与基于雅可比预处理的求解相比，结果在仅迭代 2 次后几乎无法区分。其他预处理器，如高斯-赛德尔或 SOR，在稠密矩阵上应用成本太高。

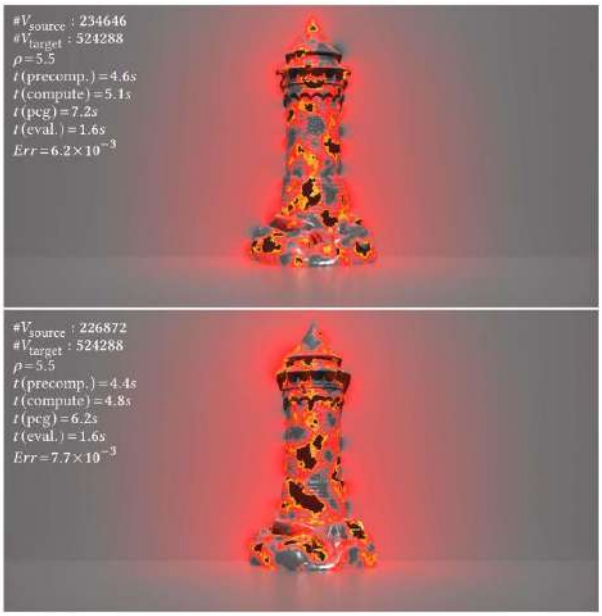


Figure 10: 3D 拉普拉斯算子。我们在 Tower 网格 (0.2M 个点，顶部和底部示例中具有不同不规则孔洞) 上绘制一个强度场，表示热量，该热量将辐射到目标平面 (0.5M 个点)。整个过程在相对误差低于 10^{-2} 的情况下不到 20 秒。

4.3 线性弹性

我们的第二个示例涉及线性弹性。通过位移 (向量) 场 u 编码的弹性变形方程是:

$$\Delta u + \frac{1}{1-2v}\nabla\left(\nabla\cdot u\right) = 0$$

其中 v 是泊松比，量化弹性材料的不可压缩性。线性弹性的基本解被称为凯尔文解，写作

$$G\left(\mathbf{x},\mathbf{y}\right) = \begin{cases} \frac{a-b}{r}\ln\left(1/r\right)\mathbf{I} + \frac{b}{r^2}\mathbf{r}\mathbf{r}^\top, & \text{in 2D} \\ \frac{a-b}{r}\mathbf{I} + \frac{b}{r^3}\mathbf{r}\mathbf{r}^\top, & \text{in 3D} \end{cases} \tag{14}$$

其中 $a = 1/2^{d-1}\pi$ 和 $b = a/4\left(1-v\right)$ 在 $d = 2,3$ 维度中，而 r 仍然表示距离 $\parallel \mathbf{x}-\mathbf{y} \parallel$ 。Kelvinlets 最近已被应用于数字雕刻中实现实时体积变形 [De Goes 和 James 2017]。然而，使用凯尔文解强制位置约束可能会显著降低性能: 它涉及求解一个密集的线性系统，非常类似于方程 (5) 中的 BIE，通过 Cholesky 或 LU 分解进行，并且可以通过秩一更新添加进一步的约束。我们的预调优器直接适用于凯尔文方法中的多个约束情况，极大地提高了求解该问题的效率。在我们的结果中，我们使用一个围绕初始对象的盒子形状边界，并在边界点上提供一系列位移向量，我们选择这些点位于盒子的每个面的网格上，见图 13(顶部)。然后求解由 MFS 推导的 BIE，以返回这些边界点上的力，由此我们通过基于 FMM 的方法推导出应用于汽车的弹性变形评估。我们在这个示例中使用了 14408 个边界点，导致 BIE 矩阵的大小为 43224×43224 。我们的 PCG 求解器在 8 秒内解决了 BIE 方程，误差低于 5×10^{-3} 。由于汽车模型大约有 199 K 个顶点，其变形在 10 秒内完成评估。图 1 中的鹰例子有 134 K 个顶点，基于与汽车相同的 BIE 解决方案，变形耗时 5 秒。

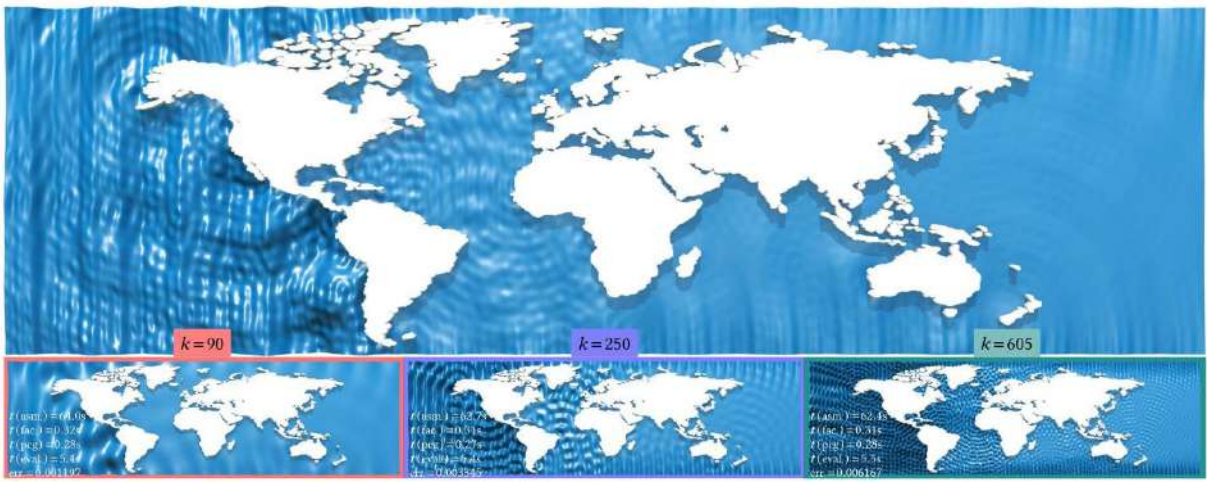


Figure 11: 2D Helmhotz 方程在不同波数下的情况。我们使用 $12.7K$ 边界点在世界地图模型上解决输入波的散射问题。在这个最小二乘法案例中，组装 K_{S_j,S_j} 成为计算成本最高的步骤，而计算 Cholesky 分解 (使用 $\rho = 8$) 和 PCG 求解的时间成本几乎可以忽略不计。我们还观察到随着 k 的增加，由于高频 Helmhotz 方程的条件数变差，误差变得更大。尽管如此，我们只对 PCG 应用了 15 次迭代，为这个特定的波传播目的给出了足够准确的结果。

4.4 Helmhotz 方程

我们讨论我们的预处理器另一个可能的应用，这次是为了展示我们的方法可以处理的操作符范围，而针对一个稍微更复杂的情况。Helmholtz 方程在计算机图形学中经常被使用，无论是用于声波传输 [James 等人 2006]，还是用于线性水波的动画模拟 [Schreck 等人 2019]，这些波在空间和时间上可分离。对于一个复值函数 u 在空间中，它施加

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

其中 $k \neq 0$ 是代表解的频率的波数。已知 Green 函数的形式为

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

其中 i 是虚数单位，$H_0^{(1)}$ 是第一类零阶汉克尔函数 (即 $H_0^{(1)}(x) = J_0(x) + \mathrm{i}Y_0(x)$ ，其中 J_0 和 Y_0 分别是第一类和第二类零阶贝塞尔函数)。与前面描述的拉普拉斯和弹性情况形成鲜明对比的是，我们的方法不能直接应用，因为赫尔姆霍茨的边界积分方程 (BIE) 矩阵 $\mathbf{K}$ 现在是复数且不是厄米正定矩阵。因此，我们解决了为赫尔姆霍茨方程所提到的最小二乘 BIE 问题，如第 2 节所述。

因此，在实施方面有两个主要区别需要我们解决。首先，在 BIE 求解阶段，我们必须显式地存储 $\mathbf{K}$ (以前我们从未这样做过)，因为计算每个项 $\left(\mathbf{K}^{\mathrm{H}}\mathbf{K}\right)_{ij}$ 需要计算内积 $\sum_{k=1}^{B} G(\mathbf{y}_k, \mathbf{z}_i)^{\mathrm{H}} G(\mathbf{y}_k, \mathbf{z}_j)$ ，这会带来很多冗余计算。其次，在插值阶段，我们直接在每个目标点上并行计算矩阵-向量乘积，因为黑盒型快速多极方法 (FMM) 通常由于使用低秩加速而在处理振荡型核函数时表现不佳。因此，这个赫尔姆霍茨情况需要更多的内存和稍多一点的时间来评估最终的目标值。在我们的测试中，我们求解了一个输入的驻波，该波沿着 $\mathbf{d}$ 方向传播并被边界散射。平面波分解为空间和时间部分，即 $u_{\text{in}}(\mathbf{x}, t) = \bar{u}_{\text{in}}(\mathbf{x}, k) \exp(-\mathrm{i}\omega_k t)$ ，其中 ω_k 是角频率，$\bar{u}_{\text{in}}(\mathbf{x}, k) = \exp\left(\mathrm{i}\mathbf{d}^{\top}\mathbf{x}\right)$ 。因此，基于狄利克雷条件的频率空间中的 BIE 是 $\sum_{j=1}^{B} G(\mathbf{y}_i, \mathbf{z}_j) s_j + \bar{u}_{\text{in}}(\mathbf{y}_i, k) = 0$ 。然后我们可以通过矩阵-向量评估得到的波解 $\bar{u}(\mathbf{x}, k)$ (并在重建波

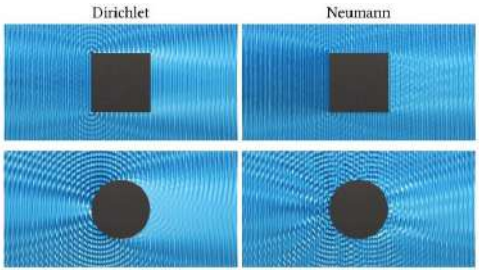


Figure 12: Dirichlet 与 Neumann 条件。Dirichlet(左) 和 Neumann 问题 (右) 都可以通过 MFS 解决，使用原始的 Green 函数或其二阶法向导数。两者都能产生输入波的有趣反射，此处为波数设置为 $k = 300$ 的情形。

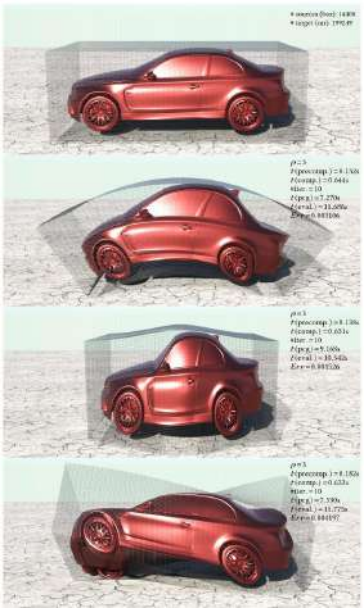


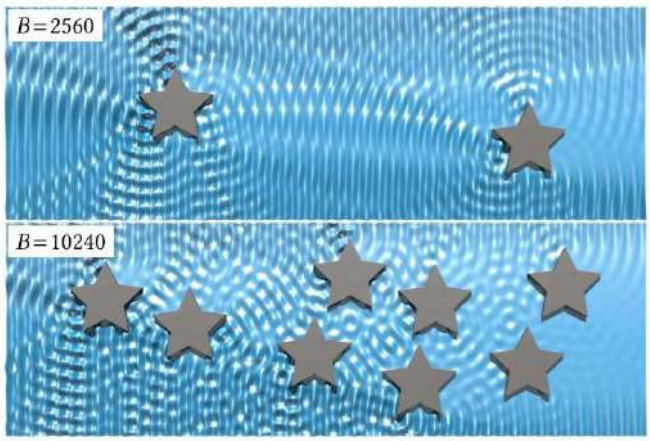
Figure 13: 稠密约束的 Kelvinlets。虽然 Kelvinlets [De Goes 和 James 2017] 允许实时雕塑模型，但设置点约束会导致稠密线性求解，从而阻碍快速结果。使用我们的预调谐器，我们可以更高效地使用 Kelvinlets 约束体积变形: 对于这个极端案例，在汽车的边界框上超过 $14K$ 个约束，我们解决 BIE，使得能够在不到 8 s 的时间内计算出汽车的诱导弹性变形。

$\bar{u}(\mathbf{x}, k) \exp(-\mathrm{i}\omega_k t)$ 之前将项 $\bar{u}_{\text{in}}(\mathbf{x}, k)$) 加回去)，仅取其实部作为时间上的波高。在表 1 中，我们提供了使用我们的预调谐器解决最小二乘 BIE 与基于递归分治策略的 Eigen 库 [Guen-nebaud 等人 2010] 中的 SVD 求解器的比较。即使对于小问题，我们也提供了数量级的速度提升；随着问题规模的增加，我们的方法速度可以提高三个数量级。在图 11 中，我们在具有 12.7 K 边界点的卷积边界上测试了我们的预

调谐器。

表 1. 与奇异值分解 (SVD) 的比较。我们将我们的求解器 (针对 $\rho = 8$) 与 Eigen 库中实现的基于分而治之的 SVD 进行比较。我们使用 $k = 300$ 解决亥姆霍兹方程，使用不同数量的边界点来散射输入波。我们的方法在速度上超过 SVD 一到三个数量级，这取决于边界大小。

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



通过性能分析，我们发现大部分执行时间花在组装项 $K_{\mathcal{S}_i,\mathcal{S}_i}$ 上，而计算预调条件器和随后迭代 PCG 至收敛总共花费的时间不到 1 s 。由于我们使用 GPU 并行计算评估，将解外推到 288 K 目标点仍然相当高效。值得注意的是，我们的 PCG 求解误差随着波数的增加而增加，因为矩阵 K 变得更加病态: 如何高效地解决高频亥姆霍兹方程仍然是应用数学中一个非常活跃的研究领域。

用我们的方法解决由诺伊曼问题产生的边界积分方程系统同样高效。类似于我们上面讨论的狄利克雷变体，我们通过计算格林函数的二阶法向导数来构建问题，并写出边界积分方程

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

这相当于对诺伊曼边界条件使用双层边界元法 (BEM)。这两种变体都导致输入波的有趣反射，如图 12 所示，并在 [Schreck 等人，2019] 中进行了演示。

4.5 讨论

为了总结这一部分，我们讨论几个更多的点，以更好地评估我们的贡献及其后果。

不确定性量化。从随机过程的视角研究传统图形问题最近引起了人们的兴趣 [Sellán 和 Jacobson 2022]。随机方法不是寻求确定性函数作为解决方案，而是关注解决方案的分布，从而可以量化求解过程中隐藏的不确定性。在许多方面，高斯过程的概念与边界值问题非常契合——基于观测数据训练计算条件均值进行预测可以直接映射到像我们的边界值问题 [Schäfer et al. 2021a] 中将边界源插值到未知的目标值。为了明确这种关系，我们可以用高斯过程重新解释 MFS 方法。假设一个 PDE 的解决方案不再是确定的而是随机的，并且在边界点和目标点上遵循零均值的高斯分布，即，

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

在这个意义上，格林函数可以被视为为 PDE 定制的 GP 核函数。接下来，从边界值 (观测数据)，我们可以“预测”目标上的解决方案 (未观测变量)，这是通过条件高斯过程计算得出的:

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

这正好重新组装了我们的 MFS 求解的两个阶段。通过 MFS 对 PDE 的解决方案因此可以被视为在边界值给定的情况下最可能的 (或预期的) 解决方案，可能还有其他概率较小的解决方案。为了量化目标 $\mathbf{y}_i$ 上解决方案的不确定性，我们可以进一步基于边界数据计算其条件方差:

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

我们指出这些也是 BIE 矩阵的 Schur 补数的对角条目。对于一个未观测到的点，知道了它的条件均值和方差，不仅可以得到最期望的解，还可以测量解在给定范围内出现的可能性 (见图 14 的示例)，这在求解 PDE 时统计确定关键区域可能很有用，有助于排除不显著的自由度以降低计算成本，或细化采样以提高结果的置信度。这种解释也可以看作是使用 GP 或径向基函数求解 PDEs 的特例 [Fornberg 和 Flyer 2015; Cockayne 等人 2017; Chen 等人 2021a, 2023]，因为径向基/协方差函数是待求解 PDE 的 Green 函数，从而简化了问题。虽然本文中我们简单地将 Green 函数作为输入到高斯过程中的先验核函数，但要更好地量化求解的可靠性，先验核应该适应问题，并考虑计算中的各种不确定性来源，例如离散化误差或边界数据中的隐藏噪声，这样我们才能开发出更可靠的解的稳健滤波器。

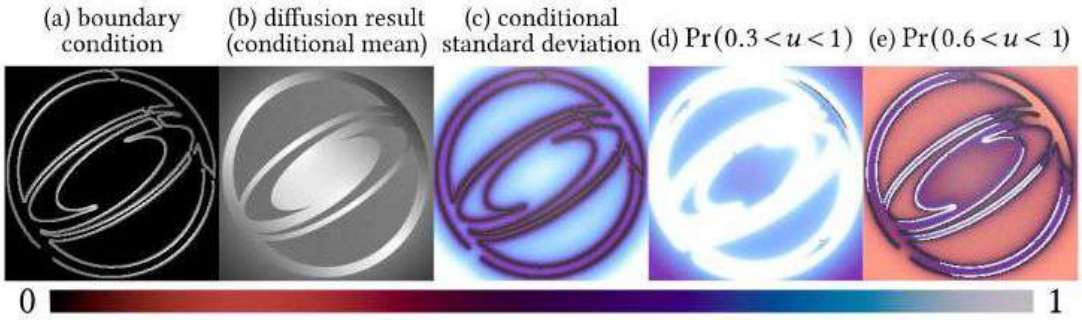


Figure 14: 不确定性量化。MFS 求解和评估可以重新解释为高斯过程的有条件情况，其中边界值是观测数据 (a)，由此我们可以计算解作为条件均值 (b)，以及条件方差 (c) 来量化解的不确定性。这使我们能够评估预期不同解范围的空间概率分布 (d, e)。

GP 与我们的方法对比。让我们也重新审视之前基于 GP 的作品与本文中提出的方法之间的差异，以明确我们的贡献。显然，高斯过程与 MFS/BEM 非常相似，在它们都从训练点 (我们的边界点) 开始，并评估预测点 (我们的目标点) 的意义上。我们依赖的最近的工作 [Guinness 2018; Schäfer 等人 2021a; Katzfuss 和 Guinness 2021] 计算了 GP 条件均值 (我们的 PDE 解) 而没有分离解决方案和评估阶段: 因此它涉及一个更大的协方差矩阵 $[K(\mathbf{x},\mathbf{x}), K(\mathbf{x},\mathbf{y}); K(\mathbf{y},\mathbf{x}), K(\mathbf{y},\mathbf{y})]$，大小为 $(B+T)\times(B+T)$，从这个矩阵的逆 Cholesky 因子的子矩阵可以直接通过矩阵-向量操作后的回代推导出条件均值。虽然他们展示了这种方法对于具有 Matérn 相关性的插值是有效的，但他们适应我们情况的直接" 从训练到预测" 方法以及" 首先预测点" 引入了显著的近似误差。这可能是因为我们的 Green 函数比 Matérn 的衰减得更慢，更加奇异，我们的回归问题在本质上比插值更具外推性。增加 ρ 以缓解这个问题在实际问题中不是一个可行的解决方案，因为如图 16 所示，这会增加计算成本。[Schäfer 等人 2021a] 的" 最后预测点" 方法承诺提供显著更准确的外推，但如果目标点过多——这在 MFS 中通常是常见的情况——则是禁止使用的。因此，我们使用了一种更接近 Kaporin 原始想法的方差预调方法，这次应用于密集矩阵，通过使用带有 FMM 的 PCG，实现了准确性与成本之间的更好权衡，提供了一种快速、可控的方式来纠正仅使用 $\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}$ 的近似解决方法。或者，我们可以使用 [Schäfer 等人 2021a] 的方法，对边界点和目标点进行联合反向最大最小排序，以使用 K 计算高效的矩阵-向量乘积，或者使用 FMM 矩阵-向量乘积来提高" 最后预测点" 方法的计算效率；我们将这些替代方法的探索推迟到未来的工作中。

$\mathcal{H}$ -矩阵基于的 LU 预调条件。由于层次矩阵是数据稀疏的，它们的 LU 分解可以某种程度上更高效地执行 [Kriemann 2013]。尽管常规 LU 在内存和执行时间上表现出超线性行为，我们还是测试了 "$\mathcal{H}$ -LU" 预调条件方法，使用现有的库 [Krie-mann 2024] 来计算预调条件器及其在 PCG 中的使用。如图 15 所示 (在 PCG 迭代之前的预调条件阶段计算时间被视为误差曲线的平坦部分，因为在此预计算期间没有观察到误差减少)，LU 分解极大地改善了条件数，但是时间成本不合理。对层次矩阵使用更激进的近似可以加快分解速度，但这种精度的大幅下降阻止了 PCG 迭代在执行矩阵-向量乘法时的收敛。

5 结论

在这篇论文中，我们提供了一种简单的方法，可以显著加速基本解法和边界元法，通过预条件化共轭梯度法提供高效的预条件器来求解边界积分方程。

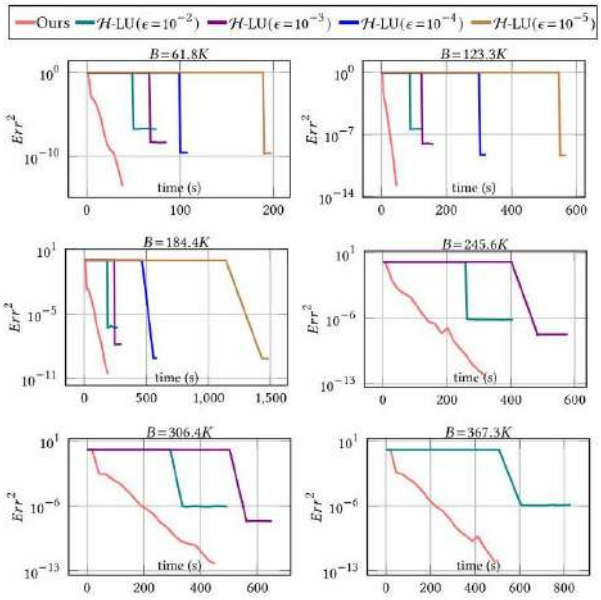


Figure 15: 与 $\mathcal{H}$ -LU 的比较。在此测试中，我们使用泊松盘采样在单位边界框内随机生成一系列点。右侧向量初始化为每个点的随机 1 或 -1。使用 HLIBpro 库 [Kriemann 2024]，H-LU 预处理器的质量和速度受到其低秩近似的 $\mathcal{H}$ -算术精度 ϵ 的影响。虽然 $\mathcal{H}$ -LU 预处理器在低精度下可能具有适度效率，但 PCG 的收敛性本质上受到限制。进一步提高精度会迅速增加计算和内存成本，特别是对于大型问题。当系统大小超过 200 K, $\mathcal{H}$ -LU 时，我们的笔记本电脑上的 $\epsilon = 10^{-4}$ 仅因内存不足而无法运行；更糟的是，与我们的方法相比，线性求解的误差仍然停留在高水平，并且没有显示出收敛问题 (这里，$\rho = 2.5$)。

尽管研究人员常常避免使用稠密矩阵，认为没有快速求解器能够处理它们，但我们通过借鉴高斯过程文献中描述通过约束最小化进行条件化的内容，证明了情况并非总是如此: 通过计算大型稠密正定矩阵 (来自 BIE) 的逆-cholesky 因子的稀疏近似，我们展示了在不要求表面或体积网格的情况下，可以显著提高边界求解器的效率。

局限性。虽然我们在所有尝试的示例中，基于 GPU 的分解步骤从未遇到过故障，但应注意，它执行的局部 Cholesky 分解可能会分解，如果正则化参数 ε (第 4.1 节) 选择得太大，例如，是源点之间最小间距的 10 倍大 - 因为 $\mathbf{K}$ 可能不再是正定对称矩阵。谢天谢地，我们希望这个 ε 足够小，以尽可能接近真实的 Green 函数，所以在实际情况下永远不会发生这种情况。然而，如果有点相互重叠，从而产生退化的 K_{S_j,S_j} 项，这种分解也可能发生。虽然可以一开始就简单地过滤掉这些情况，但探索 $\mathrm{LDL}^{\mathrm{T}}$ 分解来计算 $\mathbf{L}_{\mathcal{S}_j,j}$ 可能会很有趣，因为它适用于不定矩阵。虽然理论保证可能不再成立，但在实践中我们可能仍然会看到改进的收敛性。最后，使用最小二乘矩阵而不是非对称的 BIE 可能听起来是个糟糕的主意，因为它会增加条件数。然而，Kaporin 的变分预调谐器的有效性几乎补偿了这种做法: 例如，我们解决了拉普拉斯 BIE 的 least-square 问题，大约有 $16K$ 个源点；与原始 BIE 相比，最小二乘版本只需要额外的两次 PCG 迭代就能在误差低于 10^{-5} 的情况下收敛。

未来工作。在本研究中，我们对三种线性的椭圆型偏微分方程 (PDEs) 进行了预调条件的测试，但这种方法应该也适用于许多其他来自几何处理和基于物理的模拟的问题。例如，将我们的方法应用于基于径向基函数 (RBF) 的表面重建似乎非常直接 [Carr et al. 2001]。此外，我们使用了一个黑箱型快速多极方法 (FMM) 来加速矩阵-向量乘法。实现特定于偏微分方程的 FMM 可能会进一步提高求解和评估阶段的运行时性能。或者，如果对精度的需求不那么严格，$\mathcal{H}$ -矩阵表示可能潜在地适用于实现更快的矩阵-向量乘法，因此我们计划评估这种代数版本的 FMM 是否能进一步加速我们的方法。未来，研究使用高斯过程如 [Owhadi 2023] 中的高效求解非线性偏微分方程将非常有意思，以继续利用我们所依赖的联系: 的确，我们认为从随机角度分析和求解偏微分方程可能有助于降低计算成本并提高虚拟仿真我们现实世界的可靠性。

6 致谢

我们感谢匿名审稿人对我们论文表述的改进所提供的帮助。图 8 和图 9 中使用的图像由 pix-abay.com 提供。图 4 和图 16 中的 DEMOSTHENES 网格由 Ryan Baumann 提供，图 1 和图 10 中的 Tower 网格由 jansen-tee3d 通过 Thingiverse 提供，图 13 中的 CAR 网格由 Mike Pan 和 Morgan McGuire 提供。FS 感谢美国海军研究办公室的支持，奖项编号为 N00014-23-1-2545(解耦计算)。MD 感谢 Ansys、Adobe

Research 和 MediTwin 财团的支持，以及为 JC 提供资金的 Choose France Inria 主席。

参考文献

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-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

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

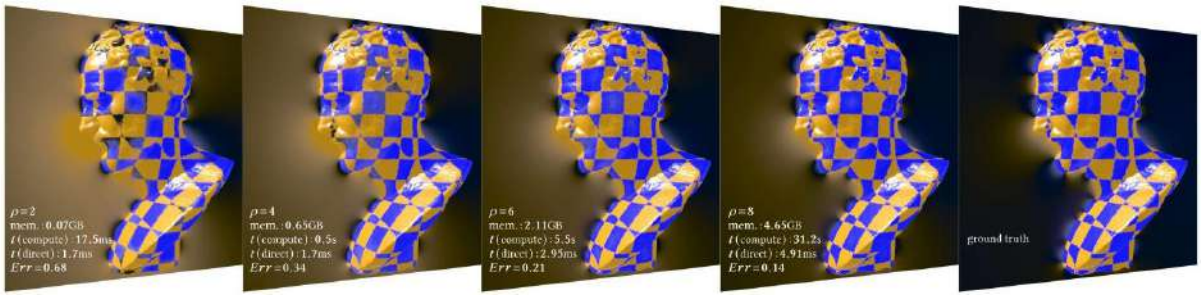


Figure 16: 稀疏逆 Cholesky 因子的不足，用于直接求解。使用 $\mathbf{L}_{S_j,j}\mathbf{L}_{S_j,j}^{\top}$ 直接求解方程 (5) 非常快，但其准确性通常不足。增加稀疏模式中的非零元素（通过增加 ρ）确实能够提高准确性，但代价是内存消耗和计算成本的迅速增长。因此，我们更愿意使用 $L_{S_j,l}L_{S_j,j}^{l}$ 作为共轭梯度的预处理器，因为它被证明效率要高得多。

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 nu-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 (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. 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 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. 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, 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.

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 version, 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),

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.

Aleksey Yuryevich Yeremin, Liliya Yurievna Kolotilina, and A. A. Nikishin. 2000. Factorized sparse approximate inverse preconditionings - III. Iterative construction of pre-conditioners. 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.

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.

变分公式

我们在这篇论文中提出的预处理器是 Kaporin 条件数在稀疏性约束下的最小化者，但其在方程 (7) 中的表达式也可以从另外两个变分公式推导出来。我们简要地解释这些不同的公式。

Kaporin 条件数。首先，我们证明方程 (7) 是 κ_{Kap} 的最小化者。根据 [Kaporin 1994]，对于大小为 $B \times B$ 的 $L_{\mathcal{S}}$ -预处理的系统，Kaporin 条件数定义为

$$\kappa_{\text{Kap}} = \frac{1}{B} \frac{\operatorname{tr}\left(\mathbf{K}\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}\right)}{\det\left(\mathbf{K}\mathbf{L}_{\mathcal{S}}\mathbf{L}_{\mathcal{S}}^{\top}\right)^{\frac{1}{B}}}.$$

通过利用矩阵迹和矩阵行列式的性质，并定义 $\mathbf{K}_{\mathcal{S}_j,\mathcal{S}_j} = \mathbf{U}_{\mathcal{S}_j,\mathcal{S}_j}\mathbf{U}_{\mathcal{S}_j,\mathcal{S}_j}^{\top}$ ，我们可以将 κ_{Kap} 展开为

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

不等式成立是因为算术平均数和几何平均数的不等式以及 Cauchy-Schwarz 不等式。当 $\mathbf{L}_{\mathcal{S}_i,j}$ 和 $K^{-1}_{\mathcal{S}_i,\mathcal{S}_j}\mathfrak{E}_j$ 共线且 $\mathbf{L}^\top_{S_j,j}\mathbf{K}_{S_j,S_j}\mathbf{L}_{S_j,j}$ 的值对于所有 j 相同时，达到最小值。因此，可以将 $\mathbf{L}^\top_{S_j,j}\mathbf{K}_{S_j,S_j}\mathbf{L}_{S_j,j}$ 归一化为 1，这导致了方程 (7)。

约束最小二乘问题。[Kolotilina 和 Yeremin 1993] 发现方程 (7)(以及等价的方程 (8)) 也是一个约束最小二乘问题的最小化者，该问题被表述为

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

其中 $\mathbf{U}$ 是上三角的 Cholesky 分解，使得 $\mathbf{K}=\mathbf{U}\mathbf{U}^\top$ 。为了解决这个约束问题，我们首先写出它的 Lagrange 函数为:

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

其中 λ_j 是 B Lagrange 乘子。对 $\mathcal{L}$ 关于 $\mathbf{L}_{\mathcal{S}_j,j}$ 求导，最优性条件为

$$\frac{\partial\mathcal{L}}{\partial\mathbf{L}_{S_j,j}}=-2\mathbf{U}_{j,j}\mathbf{e}_j+2\left(1+\lambda_j\right)\mathbf{K}_{S_j,S_j}\mathbf{L}_{S_j,j}=\mathbf{0},$$

从中解出 $L_{\mathcal{S}_j,j}$ 并表示为

$$\mathbf{L}_{S_j,j}=\frac{\mathbf{U}_{j,j}}{1+\lambda_j}\mathbf{K}^{-1}_{S_j,S_j}\mathbf{e}_j=\frac{\mathbf{U}^2_{j,j}}{1+\lambda_j}\mathbf{U}^{-\mathrm{T}}_{S_j,S_j}\mathbf{e}_j. \tag{21}$$

最后，通过将式 (21) 代入约束条件中，求得每个 Lagrange 乘子 λ_j

$$\mathbf{L}^\top_{\mathcal{S}_j,j}\mathbf{K}_{\mathcal{S}_j,\mathcal{S}_j}\mathbf{L}_{\mathcal{S}_j,j}-1=\frac{\mathbf{U}^4_{j,j}}{\left(1+\lambda_j\right)^2}-1=0.$$

因此，$U^2_{j,j}/\left(1+\lambda_j\right)=\pm1$ 和最小化器式 (21) 简化为式 (8)。

KL 散度。最近，[Schäfer et al. 2021a] 发现式 (7) 也最小化了两个零均值高斯分布的协方差 $\mathbf{K}$ 和 $\left(\mathbf{L}_\mathcal{S}\mathbf{L}_\mathcal{S}^\top\right)^{-1}$ 的 KL 散度。对于这两个协方差矩阵，KL 散度简化为

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

计算最小化 KL 散度关于每一列 $\mathbf{L}_{\mathcal{S}_j,j}$ 的最优性条件，得到

$$\frac{\partial\mathcal{D}_{KL}}{\partial L_{S_j}}=2K_{S_j,S_j}L_{S_j,j}-\frac{2}{L_{j,j}}{}_j=0. \tag{23}$$

再次，式 (7) 是式 (23) 的解。实际上，当 $\left(\mathbf{L}_\mathcal{S}\mathbf{L}_\mathcal{S}^\top\right)^{-1}$ 很好地逼近 K 时，KL 散度在数值上接近 Frobenius 范数，如图 7 所示: 原因是随着式 (20) 中的项 $\mathbf{L}_{j,j}\mathbf{U}_{j,j}$ 接近 1，它将很好地逼近 $\log\left(\mathbf{L}_{j,j}\mathbf{U}_{j,j}\right)+1$ 。