Implementing and Testing the Interpolated Factored Green Function Method For the Accelerated Evaluation of Potentials in Electromagnetic Simulators

by

Michael P. Acquaviva

Supervisor: Piero Triverio

April 7, 2025

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Michael P. Acquaviva
michael.acquaviva@mail.utoronto.ca
University of Toronto

Abstract

The boundary integral formulation of the scattering problem is central to computational electromagnetics, but it results in dense linear systems whose solution is computationally demanding. Fast algorithms such as the Fast Multipole Method (FMM) and the Adaptive Integral Method (AIM) mitigate this cost but can face challenges with parallelism, kernel generality, and high-frequency performance. The Interpolated Factored Green Function (IFGF) method offers a recent alternative, achieving $\mathcal{O}(N \log N)$ complexity through a decomposition of the Green's function into oscillatory and smooth factors, the latter of which is efficiently interpolated in space. This thesis investigates the IFGF method from both theoretical and practical perspectives. A serial implementation of IFGF was developed in Python for prototyping, and then reimplemented in C++ with an Eigen-based linear algebra engine for integration into REBEL, a GMRES-based electromagnetic solver. The algorithm was validated against direct kernel evaluations and benchmarked for runtime and memory efficiency. In parallel, the performance and scalability of IFGF are discussed in relation to FMM and AIM, with attention given to their structural trade-offs. Experimental results confirm that IFGF achieves near-log-linear scaling in both runtime and memory while preserving accuracy across a range of problem sizes and wavenumbers. This work demonstrates the viability of IFGF as a general-purpose fast solver and establishes a foundation for further comparative studies on complex geometries and large-scale electromagnetic simulations.

Acknowledgements

I would first like to thank Professor Piero Triverio for his invaluable supervision and mentorship throughout the course of this project. Professor Triverio not only guided this work, but also sparked my interest in electromagnetics through his teaching of ECE259 – one of my favourite courses during my undergraduate studies at the University of Toronto. His clarity, insight, and passion for the subject matter have been a source of continual inspiration.

I would also like to thank Jasper Hatton for introducing me to REBEL – the Robust Electromagnetics Boundary Element Library. His explainations provided me with a solid foundation for understanding the architecture of iterative solvers, like GMRES, and the intricacies of the REBEL library. I am grateful for his patience and willingness to help me navigate the complexities of the codebase.

Yongzhong Li was instrumental in helping me understand the process of integrating new acceleration methods, like the method I present in this thesis, into the REBEL library. His guidance helped make this project a reality and continues to pave the way for future work on this topic.

Lastly, I wish to express my gratitude to the professors and faculty of the University of Toronto Engineering Science program. Their dedication, high standards, and passion for teaching have shaped my approach to learning and research, and have played a significant role in my development as an engineer.

Contents

List of Figures

List of Tables

List of Symbols

Geometric Quantities

Symbol	Description
$\vec{r}$	Observation point in $\mathbb{R}^3$
$\vec{r}'$	Source point in $\mathbb{R}^3$
$\vec{r}_n$	Centroid of triangle n
H	Side length of an octree box
D	Depth of the octree used in IFGF
N	Number of sources or targets
$r_<, r_>$	Minimum and maximum of $\|\vec{r}\|$, $\|\vec{r}'\|$
γ	Angle between $\vec{r}$ and $\vec{r}'$

Electromagnetic Quantities

Symbol	Description
$\vec{J}(\vec{r})$	Surface current density vector
$\vec{A}(\vec{r})$	Magnetic vector potential
$\phi(\vec{r})$	Scalar electric potential
$\vec{E}(\vec{r})$	Electric field vector
$\vec{D}(\vec{r})$	Electric displacement field
$\vec{B}(\vec{r})$	Magnetic flux density
ϵ	Permittivity of the medium
μ	Permeability of the medium
ω	Angular frequency
λ	Wavelength, $\lambda = 2\pi/\kappa$
κ	Wavenumber, $\kappa = \omega\sqrt{\mu\epsilon}$

IFGF-Related Quantities

Symbol	Description
w_n	Weight of nth basis function
$\vec{f}_n$	Basis function vector on triangle n
$u(\vec{r})$	Scalar field at $\vec{r}$
$F_k^d(\vec{r})$	Field contribution from box k at level d
$G(\vec{r}, \vec{r}')$	Helmholtz Green's function
$g(\vec{r}, \vec{r}', \vec{r}_s)$	Analytic IFGF factor
$G(\vec{r}, \vec{r}_s)$	Oscillatory IFGF factor
$\delta(\vec{r} - \vec{r}_n)$	Dirac delta at $\vec{r}_n$

Mathematical Operators and Functions

Symbol	Description
$*$	Convolution operator
$\mathcal{F}$	Fourier transform
$\mathcal{F}^{-1}$	Inverse Fourier transform
P_l	Legendre polynomial of degree l
$j_l, h_l^{(1)}$	Spherical Bessel and Hankel functions of order l
j	Imaginary unit, $j = \sqrt{-1}$
$\mathcal{O}(\cdot)$	Big-O asymptotic complexity

Chapter 1

Introduction

As the complexity of modern electromagnetic systems increases, so too does the need for accurate, scalable, and computationally efficient design and simulation tools [2]. At the heart of many electromagnetic simulations lies the *scattering problem*: determining how electromagnetic fields and waves interact with three-dimensional objects [3]. This problem is ubiquitous across numerous applications, including antennas, radar, wireless communications, and integrated circuits [1]. Accurate modelling of these interactions is essential to design, verify, and optimize electromagnetic systems with increasingly stringent performance requirements.

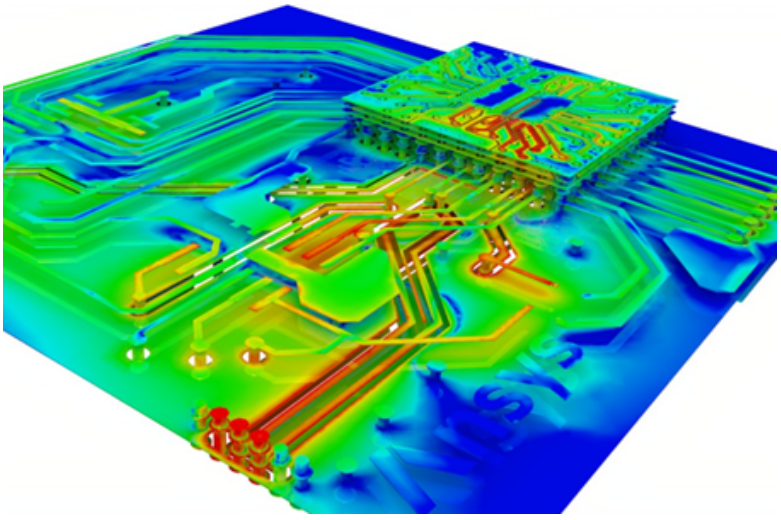


Figure 1.1: An example of a complex electromagnetic simulation: a current density model of an interposer connecting multiple chiplets. Credit: ANSYS [1]

These demands are compounded by trends in modern engineering. As systems operate at ever-higher frequencies – from gigaherts to millimeter-wave and beyond – the electrical size of the geometries under consideration increases, leading to more oscillatory behaviour and the need

for finer spatial discritization in simulations [4]. This increases the number of unknowns in the numerical solution, leading to inflated computational burdens. At the same time, the materials involved in these systems are becoming more complex, encompasing anisotropic, lossy, and dispersive media that must be modelled with precision. In integrated circuits, for example, interconnects and packaging materials now involve layered dielectrics and high-conductivity metals with sub-wavelength features. Together, these factors have made classical numerical solutions to the scattering problem increasingly impractical. New methods are needed that can efficiently handle large-scale problems while maintaining accuracy and fidelity.

When formulated as a boundary value problem (BVP), the scattering problem results in a dense linear system whose naïve direct solution is computationally expensive, requiring $\mathcal{O}(N^3)$ operations and $\mathcal{O}(N^2)$ memory, where N is the number of discritized points on the surface of the scatterer [5]. While iterative methods, like the Generalized Minimal Residual (GMRES), can help manage the computational cost of solving these systems, even the best-case runtimes face the dominant bottleneck of the matrix-vector product involving the Green's function interaction matrix. Consequently, an entire field of research has emerged, over several decades, to accelerate this operation to sub-quadratic time and memory complexity.

Two widely adopted acceleration strategies are the *Fast Multipole Method* (FMM) [6] and the *Adaptive Integral Method* (AIM) [7]. These methods reduce the runtime to $\mathcal{O}(N \log N)$ or even $\mathcal{O}(N)$ under favourable conditions [2]. However, they are not without their limitations. While after it's 1987 invention by Louis Greengard and Vladamir Rokhlin, FMM was counted among the top ten algorithms of the twentieth century, it has two main drawbacks, stemming from the way it approximates the Green's function. Firstly, the FMM is not suited for high-frequency problems, as it depends on a series expansion of the exponential factor of the Green's function. To maintain accuracy, the number of terms in this series must be increased as the frequency increases, leading to a loss of efficiency. Secondly, the application of the FMM is limited to homogenous media, since the series expansion of the Green's function is only valid at a particular wavenumber, κ [5]. Similarly, the classic AIM relies on an assumption of homogeneity in the medium, however it's non-reliance on a series expansion allows it to generalize well to higher frequencies [8]. AIM, instead, relies on a projection of the source points to a regular Cartesian grid where then the convolution with the Green's function is performed by means of a Fast Fourier Transform (FFT). This reliance on the FFT, however, presents a parallelization bottleneck. Furthermore, the reliance on a regular grid limits the applicability of AIM to problems with regular geometries. More advanced varients of AIM, such as the implementation presented by Shashwat Sharma, Utkarsh R. Patel, and Piero Triverio in [9], have attempted to extend AIM to layered media, but these methods are still limited by the need for a regular grid and the FFT bottleneck.

The topic of this thesis is the *Interpolated Factored Green Function Method* (IFGF) [5], a novel

technique for accelerating the solution of discrete integral operators used in the boundary value problem (BVP) formulation of the scattering problem. This method was developed by Dr. Cristoph Bauinger and Dr. Oscar Bruno at the California Institute of Technology (Caltech) and was published in their 2021 paper in the Journal of Computational Physics. IFGF introduces a fundamentally different approach to computing fast matrix-vector products of the Green's function interaction matrix by factoring the free space Green's function into a highly-oscillatory component and a slowly varying, smooth componment. The latter can be approximated using a low-degree polynomial interpolation over spatial subdomains, enabling a recursive, hierarchical evaluation scheme without the need to rely on frequency-dependant multipole expansions or FFTs.

This method achieves an asymptotic runtime and memory complexity of $\mathcal{O}(N \log N)$, while maintaining accuracy across both the low- and high-frequency regimes. The IFGF method is also inherently parallelizable [10] due to its localized nature. Moreover, unlike FMM, IFGF does not suffer a loss of efficiency at high frequencies, since its interpolation strategy adapts naturally to oscillatory Green's functions. While IFGF is still subject to the same limitations as other Green's function-based solvers with respect to homogeneity of the background medium, its simplicity, scalability, and potential for parallel computation make it a promising candidate for accelerating large-scale electromagnetic simulations [2].

The purpose of the work set out in this thesis is threefold. First, to implement the IFGF method in a manner which is suitable for implementation in an iterative solver. Second, to validate the computational costs and accuracy of the IFGF method while integrating it into a larger GMRES-based solver. Third, to compare the performance of the IFGF method to that of AIM and FMM on practical, large-scale problems. This work will be presented in the following chapters:

- Chapter 2 reviews the mathematical background of the scattering problem and the existing acceleration techniques (AIM and FMM).
- Chapter 3 describes the IFGF method in detail, including its mathematical formulation and the implementation of the algorithm.
- Chapter 4 presents the methodology and implementation of the IFGF algorithm, including its spatial partitioning structures, interpolation schemes, and evaluation logic.
- Chapter 5 evaluates the performance of the IFGF-based solver on test problems.
- Chapter 6 concludes with a discussion of the results and outlines future work to be done in this area.

Chapter 2

Mathematical Background

This chapter provides the mathematical foundation required to understand and implement the Interpolated Factored Green Function (IFGF) method. We begin by reviewing the formulation of the electromagnetic scattering problem, focusing on the boundary integral approach and its numerical discritization. This is followed by a discussion of classical acceleration techniques, which set the stage for the IFGF method.

2.1 Electromagnetic Scattering and the Boundary Integral Formulation

The electromagnetic scattering problem concerns the interaction between an incident field or wave and an object, called a *scatterer*, typically a conductor or a dielectric. This interaction causes part of the incident field or wave to be reflected, transmitted, or absorbed by the scatterer. This phenomenon is central to many applications involving electrical systems and devices such as antennas, radar, integrated circuits (ICs) and wireless communications. As these systems grow in electrical size and complexity, the underlying mathematical problem of determining the scattered field induced by a known excitation becomes increasingly challenging and expensive to solve [4].

Scattering problems involve solving the classical N-body problem, which is a present in many areas of physics. This problem involves solving for the interactions between a large number of discrete entities that interact pairwise through a kernel function. In electrostatics and gravitation, the N-body problem involves computing the potential or force at each particle due to all the others. In electromagnetic scattering, we instead seek to solve for the induced charges and currents on a surface, but the key computational bottleneck remains fundamentally the same:

the need to compute the pairwise interactions between all the entities [5].

This interaction, for a system of N entities, is encoded in a dense matrix referred to as the Green's function matrix. Each entry in this $N \times N$ matrix quantifies the contribution of a source point to a field point, and the matrix-vector product involving this matrix represents the total field due to all sources. In this thesis, we approach scattering using a boundary integral formulation, which allows us to express the scattering problem in terms of surface integrals over the scatterer's surface. This approach is particularly advantageous for complex geometries, as it reduces the dimensionality of the problem from a volumetric formulation to a surface formulation. The remainder of this section builds up the boundary integral formulation from Maxwell's equations for electromagnetism.

2.1.1 Maxwell's Equations and Time-Harmonic Fields

The foundation for any electromagnetic scattering problem is *Maxwell's Equations*, which describe the behaviour of electric and magnetic fields in space and time. In their fully time-dependant form, the are a set of coupled partial differential equations (PDEs) that relate the electric field $\vec{E}$, the magnetic field $\vec{H}$, the electric displacement field $\vec{D}$, the magnetic flux density $\vec{B}$, free charge density ρ_v, and free current density $\vec{J}$. The equations are given by [11]:

$$\nabla \cdot \vec{D} = \rho_v \quad \text{(Gauss's Law for Electricity)} \tag{2.1}$$

$$\nabla \cdot \vec{B} = 0 \quad \text{(Gauss's Law for Magnetism)} \tag{2.2}$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \quad \text{(Faraday's Law of Induction)} \tag{2.3}$$

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \quad \text{(Ampere-Maxwell Law)} \tag{2.4}$$

For scattering problems involving continuous-wave excitation at a single frequency, we can assume *time-harmonic fields* of the form $\vec{E}(\vec{r}, t) = \Re\left\{\vec{E}(\vec{r})e^{j\omega t}\right\}$, where ω is the angular frequency and $\vec{r}$ is the position vector of the observation point. Under this assumption, the time derivatives are replaced with multiplication by $j\omega$ and Maxwell's equations reduce to their frequency-domain form:

$$\nabla \cdot \vec{D} = \rho_v \tag{2.5}$$

$$\nabla \cdot \vec{B} = 0 \tag{2.6}$$

$$\nabla \times \vec{E} = -j\omega\vec{B} \tag{2.7}$$

$$\nabla \times \vec{H} = \vec{J} + j\omega\vec{D} \tag{2.8}$$

The fields are related to material properties through the constitutive relations: $\vec{D} = \varepsilon\vec{E}$ $\vec{B} =$

$\mu\vec{H}$, and $\vec{J} = \sigma\vec{E}$, where ε is the electric permittivity, μ is the magnetic permeability, and σ is the electrical conductivity of the medium.

In the context of electromagnetic scattering, incident fields and waves interact with a scatterer to induces surface currents and charge distributions. These induced currents and charges, in turn, generate a scattered field. The total field, $\vec{E_t}$ is the superposition of the incident field $\vec{E_i}$ and the scattered field $\vec{E_s}$ [3]:

$$\vec{E_t} = \vec{E_i} + \vec{E_s} \tag{2.9}$$

A similar relationship holds for the total magnetic field $\vec{H_t}$. The goal of the scattering problem is to compute the scattered field $\vec{E_s}$ given the incident field $\vec{E_i}$ and the properties of the scatterer. This is the starting point from which the boundary integral formulation is derived.

2.1.2 Potentials and the Helmholtz Equation

In scattering problems, it is advantageous to express the fields in terms of potentials – this allows us to decouple the equations and solve for the fields using Green's functions (which we will discuss in Section 2.1.3). The electric field can be expressed in terms of the electric scalar potential ϕ and the magnetic vector potential $\vec{A}$ as:

$$\vec{E} = -\nabla\phi - j\omega\vec{A} \tag{2.10}$$

The magnetic field can be expressed in terms of the vector potential as:

$$\vec{H} = \nabla \times \vec{A} \tag{2.11}$$

These equations are derived under the *Lorenz gauge* condition $\nabla \cdot \vec{A} + j\omega\varepsilon\phi = 0$. Under this gauge, Maxwell's equations decouple into two wave equations, one for the electric potential and one for the magnetic potential. The wave equation for the electric potential is given by:

$$\nabla^2\phi + \kappa^2\phi = -\frac{\rho_v}{\varepsilon} \tag{2.12}$$

where $\kappa^2 = -j\omega\mu(\sigma + j\omega\varepsilon)$. κ is the complex wavenumber and is strictly real for lossless dielectric media. The wave equation for the magnetic potential is given by:

$$\nabla^2\vec{A} + \kappa^2\vec{A} = -\mu\vec{J} \tag{2.13}$$

These equations introduce the *Helmholtz operator* $\nabla^2 + \kappa^2$, which is a second-order differential operator that describes the propagation of waves in a medium. Equations 2.12 and 2.13 are the *inhomogeneous Helmholtz equations* for the electric and magnetic potentials, respectively. The solutions to these equations describe the behaviour of the electric and magnetic fields in the

presence of sources [3].

2.1.3 Green's Functions and Integral Equations

To solve the inhomogeneous Helmholtz Equations, 2.12 and 2.13, we can use the method of Green's functions. A Green's function represents the response of the system – in this case, the electromagnetic field – to a unit point source. Once the appropriate Green's function, $G(\vec{r}, \vec{r'})$, is determined, the solution to the inhomogeneous Helmholtz equations can be expressed as an integral over the source distribution. Note, that the Green's function is a function of two position vectors: $\vec{r}$, the observation point, and $\vec{r'}$, the source point. For the scalar Helmholtz equation in free space, the Green's function satisfies:

$$\left(\nabla^2 + \kappa^2\right) G(\vec{r}, \vec{r'}) = -\delta(\vec{r} - \vec{r'}) \tag{2.14}$$

where $\delta(\vec{r} - \vec{r'})$ is the Dirac delta function, which represents a point source at $\vec{r'}$. The Green's function for the Helmholtz equation in free space is given by:

$$G(\vec{r}, \vec{r'}) = \frac{e^{-j\kappa|\vec{r} - \vec{r'}|}}{4\pi|\vec{r} - \vec{r'}|} \tag{2.15}$$

Using Equation 2.15, the scalar and vector potentials due to a distribution of sources can be written in continuous integral form as:

$$\phi(\vec{r}) = \frac{1}{\varepsilon} \int_V G(\vec{r}, \vec{r'}) \rho_v(\vec{r'}) dV' \tag{2.16}$$

and

$$\vec{A}(\vec{r}) = \mu \int_V G(\vec{r}, \vec{r'}) \vec{J}(\vec{r'}) dV' \tag{2.17}$$

where V is the volume of the scatterer and dV' is the differential volume element.

These continuous volumetric integrals cannot always be solved analytically [3] [12], especially for most practical geometries. Instead, the bodies are formulated using equivalent surface charges and currents, which allows us to express the potentials in terms of surface integrals over the scatterer's surface. This is the basis of the boundary integral formulation which works by applying boundary conditions at the surface of the scatterer. For a *perfect electrical conductor* (PEC), this is a valid assumption since the surface of the conductor is an equipotential surface, but due to the finite skin-depth of conductors [12], this assumption can be extended to the case of practical lossy conductors as well.

Section 2.1.4 discusses the numerical solution to the boundary integral formulation, which allows computers to solve the scattering problem.

2.1.4 Discritization and Linear Systems

To solve the boundary integral equations numerically, the continuous surface of the scatterer must be discritized into a finite number of elements – typically small patches which we refer to as a *mesh*. The mesh is typically comprised of a series of triangular elements which follow the geometry of the scatterer. For a simple sphere, such a triangular meshing is depicted in Figure 2.1. The surface of the scatterer is represented as a collection of triangular elements, each with a set of vertices. The surface current density $\vec{J}(\vec{r})$ is approximated as a continuous function over each triangular element, allowing us to express the surface integral in terms of a sum over the individual elements.

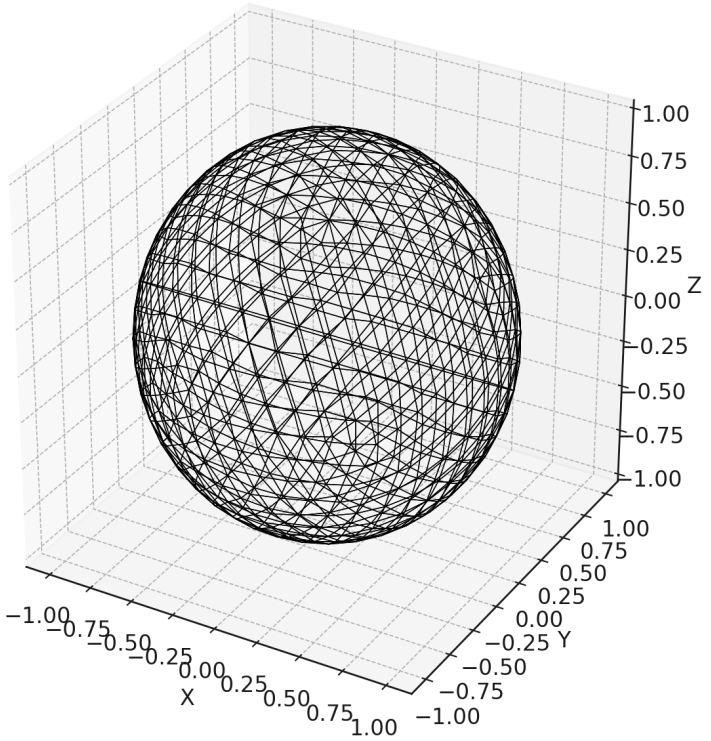


Figure 2.1: Triangular mesh of a sphere. The surface current density is approximated as a function over each triangular element.

This approach relies on the *method of moments* (MoM) [3] [11], which involves approximating the continuous surface current density $\vec{J}(\vec{r})$ as a linear combination of N known basis functions, $\{\vec{f_n}(\vec{r})\}$, with unknown coefficients $\{w_n\}$. The surface current density can be expressed approximately as:

$$\vec{J}(\vec{r}) \approx \sum_{n=1}^{N} w_n \vec{f_n}(\vec{r}) \tag{2.18}$$

Substituting this into the integral equations for the potentials yeilds a linear system of equations

of the form:

$$\mathbf{A}\vec{w} = \vec{b} \tag{2.19}$$

where $\mathbf{A} \in \mathbb{C}^{N \times N}$ is the matrix representing the Green's function interactions between the basis functions, $\vec{w}$ is the vector of unknown source coefficients, and $\vec{b}$ is the vector of known values representing the incident field.

Now, it is evident that the solution to the scattering problem reduces to solving the linear system given by Equation 2.19. The solution to this linear system yields the source coefficients, $\vec{w}$, which then can be used to reconstruct the surface current density $\vec{J}(\vec{r})$ using Equation 2.18. The choice of basis functions is critical to the accuracy of the solution, but is beyond the scope of this thesis. For the purposes of this work, we will assume a constant current distribution over each mesh element.

2.1.5 Solving the Linear System

The system outlined in Equation 2.19 is a dense linear system, which means that the matrix $\mathbf{A}$ is typically very large and requires a significant amount of memory to store. Furthermore, direct methods rely on the direct computation of the entries to fill the matrix $\mathbf{A}$, which is runtime and memory expensive.

Gaussian Elimination

Gaussian Elimination [12] is the fundamental algorithm for solving linear systems. It proceeds by transforming the matrix $\mathbf{A}$ into an upper triangular form using a series of row operations. Once in this form, the system can be solved using back substitution. The computational complexity of Gaussian elimination is $\mathcal{O}(N^3)$, where N is the number of unknowns in the system. This method is not suitable for large-scale problems due to its high computational cost and memory requirements.

LU Decomposition

LU decomposition improves on Gaussian Elimination [3] by factoring the matrix $\mathbf{A}$ into the product of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$. This allows for more efficient solution of the system, as the forward and backward substitution steps can be performed separately. The computational complexity of LU decomposition is also $\mathcal{O}(N^3)$, however, LU decomposition is widely used in smaller problems or as part of hybrid methods, such as block LU or domain decomposition preconditioners [12] in iterative frameworks.

Generalized Minimal Residual Method

For large-scale scattering problems, iterative solvers are preferred. The Generalized Minimal Residual (GMRES) [13] method is a popular iterative method for solving linear systems. It is particularly well-suited for large matrices, and it converges to the solution by minimizing the residual. The solution for the vector $\vec{w}$ is approximated by guessing a solution $\vec{w}_k$ and iteratively refining it until the residual $||\vec{r}_k|| = \mathbf{A}\vec{w}_k - \vec{b}$ is minimized.

The computational cost of GMRES is dominated by the matrix-vector product, which requires $\mathcal{O}(N^2)$ operations if unaccelerated. However, the method can be vastly accelerated if the matrix-vector product can be computed in sub-quadratic time. In Section 2.2, we will review classical acceleration techniques, including the Fast Multipole Method (FMM) and the Adaptive Integral Method (AIM), which are widely used to accelerate the matrix-vector product. We will then introduce the Interpolated Factored Green Function (IFGF) method in Chapter 3, which is the main focus of this thesis.

2.2 Classical Acceleration Techniques

As discussed in Section 2.1.5, the matrix-vector product involving the Green's function interaction matrix is the dominant computational bottleneck in solving the linear system [6]. This arises due to two main factors. First, the interaction matrix $\mathbf{A}$ itself must be populated with pairwise evaluations of the Green's function. At best, if the system is recriprocal, this requires $(\frac{N}{2})^2$ computations to fill the matrix, which is $\mathcal{O}(N^2)$ operations. Second, the matrix-vector product, $\mathbf{A}\vec{w}$, requires $\mathcal{O}(N^2)$ operations to compute, even if the matrix is sparse. This is the main bottleneck in the GMRES method, and it is the focus of many acceleration techniques.

Over the past several decades, two widely adopted acceleration techniques have emerged: the *Fast Multipole Method* (FMM) and the *Adaptive Integral Method* (AIM). These methods reduce the computational cost of the matrix-vector product to $\mathcal{O}(N \log N)$ or even $\mathcal{O}(N)$ under favourable conditions [6]. However, both have limitations in terms of accuracy, parallelizability, or applicability to different problem domains. In this section, we will review these classical acceleration techniques and their limitations.

2.2.1 The Fast Multipole Method

The Fast Multipole Method (FMM), originally developed by Greengard and Rokhlin [6], is a hierarchical algorithm designed to accelerate the computation of long-range interactions in N-body problems. Its core idea is that interactions between well-separated clusters of points can be efficiently approximated using a truncated series expansion of the Green's function. This hierarchical decomposition enables the method to reduce computational complexity from

$\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$ time and memory in many practical cases.

In the FMM framework, different types of interactions are treated based on spatial proximity [6]:

1. Near-field interactions are computed directly using the full Green's function;
2. Far-field interactions are approximated using multipole expansions, where groups of distant sources are aggregated and their collective effect is evaluated;
3. Singularity-resolving interactions are handled with special care when source and observer points are extremely close, to avoid numerical instability.

When the observation point $\vec{r}$ is far from the source point $\vec{r'}$, the Helmholtz Green's function can be approximated by the following multipole expansion [6]:

$$\frac{e^{j\kappa|\vec{r}-\vec{r'}|}}{4\pi|\vec{r}-\vec{r'}|} \approx j\kappa \sum_{l=0}^{\infty} (2l+1) j_l(\kappa r_<) h_l^{(1)}(\kappa r_>) P_l(\cos\gamma) \tag{2.20}$$

where

- $r_< = \min(|\vec{r}|, |\vec{r'}|)$ and $r_> = \max(|\vec{r}|, |\vec{r'}|)$,
- $j_l(\cdot)$ is the spherical Bessel function of the first kind of order l,
- $h_l^{(1)}(\cdot)$ is the spherical Hankel function of the first kind of order l,
- $P_l(\cos\gamma)$ is the Legendre polynomial of degree l,
- γ is the angle between $\vec{r}$ and $\vec{r'}$, i.e., $\cos\gamma = \hat{r} \cdot \hat{r'}$.

This expansion effectively approximates the oscillatory exponential term in the Green's function. For a Helmholtz equation with wavenumber κ over a domain of electric size κD, the computational complexity of FMM is approximately $\mathcal{O}((\kappa D)^2 N)$, where N is the number of discretization points.

Despite its strengths, the FMM has important limitations. First, the multipole expansion becomes increasingly expensive at high frequencies: to maintain a fixed level of accuracy, the number of terms in the series must grow with the wavenumber κ. Second, the method is restricted to homogeneous media, since the Green's function expansion assumes a constant wavenumber throughout the domain. As a result, FMM is not directly applicable to inhomogeneous or stratified media [9].

To address this, several variants have been proposed. For instance, the *Generalized FMM* (GFMM) and *Adaptive FMM* (AFMM) [14] attempt to compute *equivalent* Green's functions between points in non-homogeneous media. These methods typically require a different expansion for each source-observer pair, which leads to significantly increased implementation complexity.

Nevertheless, the Fast Multipole Method remains a powerful and widely adopted tool for accelerating the solution of N-body problems in homogeneous media. It is particularly effective in low-frequency and electrostatic regimes and is highly amenable to parallelization, making it well-suited for large-scale simulations.

2.2.2 The Adaptive Integral Method

The Adaptive Integral Method (AIM) [7] is another widely used technique for accelerating the computation of the Green's function matrix-vector product. Unlike the FMM, which relies on a series expansion of the exponential factor, AIM computes a convolution of the Green's function kernel on a regular grid of source points using a Fast Fourier Transform (FFT). The narritive of the algorithm is as follows:

Grid Projection

Let $\vec{r'}_i$ denote the positions of the source points with corresponding weights w_i. The first step in AIM is to project the source points onto a regular Cartesian grid using basis functions $\vec{\psi}_l(\vec{r'})$. Now, the sources can be written as:

$$\vec{J}(\vec{r'}) \approx \sum_{i=1}^{N} w_i \vec{\psi}_l(\vec{r'}) \tag{2.21}$$

where $\vec{\psi}_l(\vec{r'})$ is non-zero at nearby grid points.

Convolution with the Green's Function

Once projected onto the grid, the field at each observation point is computed by convolving the gridded source distribution with the free-space Green's function:

$$\vec{A}(\vec{r}) = \int_V G(\vec{r} - \vec{r'}) \vec{J}(\vec{r'}) dV' \tag{2.22}$$

This operation is a spatial convolution of the form:

$$\vec{A} = \vec{G} * \vec{J} \tag{2.23}$$

When in the Fourier domain, this convolution becomes a pointwise multiplication:

$$\mathcal{F}(\vec{A}) = \mathcal{F}(\vec{G}) \cdot \mathcal{F}(\vec{J}) \tag{2.24}$$

Using the FFT, this product can be computed in $\mathcal{O}(N \log N)$ time. The result is then transformed back to the spatial domain using the inverse FFT, which maintains the same memory and runtime complexity [8].

AIM is particularly effective for high-frequency problems, which is where the traditional FMM degrades in performance. While the classic AIM does work under the assumption of homogeneity, it can be extended to layered media by using a series of projections and convolutions. However, AIM's main limitation is its reliance on a regular Cartesian grid, which renders it less effective for problems with irregular, aperiodic geometries. Furthermore, since the FFT is a global operation, AIM is not inherently parallelizable.

In the next chapter, we will introduce the Interpolated Factored Green Function (IFGF) method, which addresses these limitations while maintaining the computational efficiency of AIM and FMM.

Chapter 3

The Interpolated Factored Green Function Method

Despite the significant advances offered by classical acceleration techniques like the FMM and AIM, limitations remain in their ability to handle general electromagnetic scattering problems, particularly in the high-frequency regime and in the presence of complex geometries. The Interpolated Factored Green Function Method (IFGF) [5], introduced by Dr. Christoph Bauinger and Dr. Oscar Bruno, offers a new approach to the evaluation of dense Green's function interactions. Like the FMM, it is hierarchical and achieves $\mathcal{O}(N \log N)$ time and memory complexity. Like AIM, it performs well at high-frequencies and scales efficiently with the wavenumber κ. Unlike either, IFGF relies on a factoring of the Green's function into a highly-oscillatory component and a smooth, slowly-varying component, hence avoiding the need for truncated series expansions or FFTs [15]. This makes IFGF relatively simple to implement and more amendable to parallelization.

In this chapter, we will describe the mathematical formulation and algorithmic structure of the IFGF method. We begin by presenting the theoretical motivation behind the Green's function factorization and the interpolation strategy. We then describe the spatial partitioning scheme and recursive evaluation algorithm, and conclude with a discussion of computational complexity, parallelism, and limitations.

3.1 Theoretical Motivation

At the core of the IFGF method is an observation about the structure of the Helmholtz Green's Function, given by Equation 2.15. Now, consider a square box centered at some point $\vec{r_s}$, with side length H, and let $\vec{r'}$ be a source point located inside the box. This setup is depicted in

Figure 3.1. Say we now want to evaluate the Green's function at a point $\vec{r}$ located outside the box. The Green's function can be re-written as:

$$G(\vec{r},\vec{r}\,') = \left(\frac{e^{jk|\vec{r}-\vec{r}_s|}}{4\pi|\vec{r}-\vec{r}_s|}\right)\left(\frac{|\vec{r}-\vec{r}_s|}{|\vec{r}-\vec{r}\,'|}\,e^{jk(|\vec{r}-\vec{r}\,'|-|\vec{r}-\vec{r}_s|)}\right) \tag{3.1}$$

Notice that the Green's function can be factored into two parts: one which depends only on the observation point and the location of the source box's center, and another which depends on the source point, the observation point, and the center of the source box. We call the first factor the *centered factor*, $G_s(\vec{r},\vec{r_s})$ and the latter the *analytic factor*, $g_s(\vec{r},\vec{r'},\vec{r_s})$.

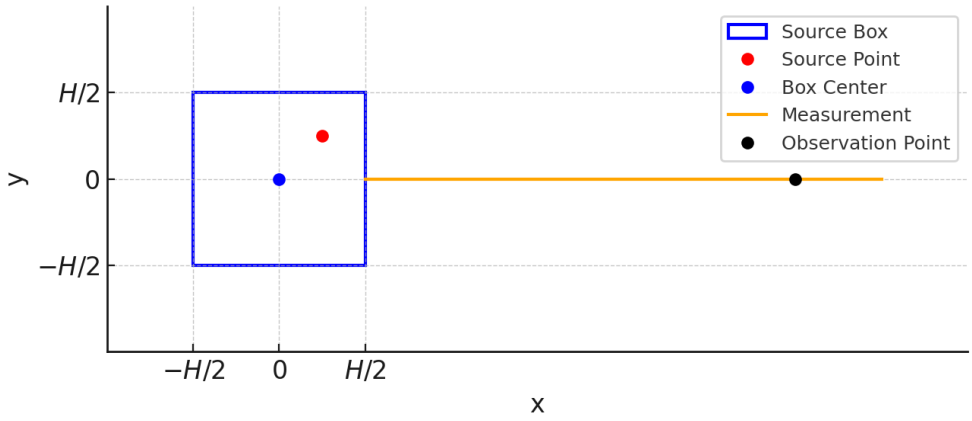
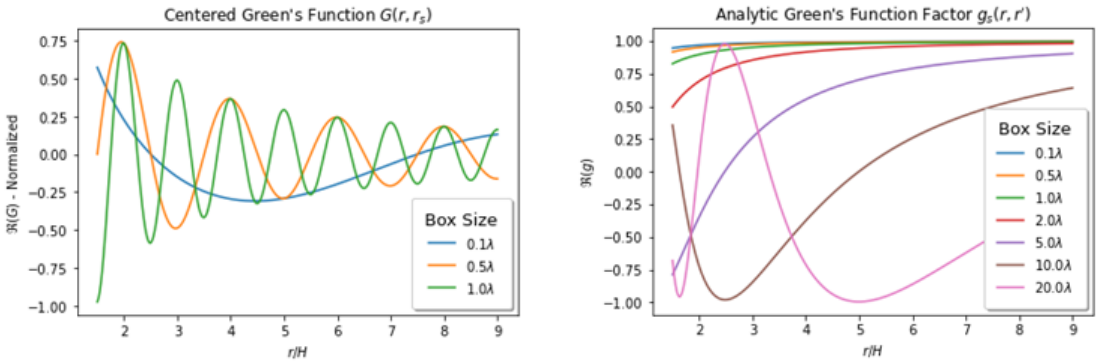


Figure 3.1: The source box and target setup for factorization in the IFGF method.

For simplicity of analysis moving forward, we will assume that the box is centered at the origin (i.e., $\vec{r_s}=0$). At first glance, this factorization may not seem particularly useful, however after analysing the properties of g_s and G_s, it is evident that the former is slowly oscillatory and the latter maintains the highly oscillatory nature of the Green's function. Figure 3.2 shows this behaviour for a variety of different electrical sizes of the source box.

Examining the analytic factor in Figure 3.2b, we can see that, even for electrically large boxes, the behaviour of the analytic factor is relatively smooth and slowly varying. This suggests that it can be approximated using a low-degree polynomial interpolation scheme with a small number of interpolation points. This observation – that the analytic factor g_s is slowly varying – is the central insight that makes the IFGF method efficient. Instead of evaluating the full Green's function directly for every source-target pair we can do the following:

1. Evaluate the centered factor $G_s(\vec{r},\vec{r_s})$ once per source box, and
2. Approximate the analytic factor $g_s(\vec{r},\vec{r'},\vec{r_s})$ using a low-degree polynomial interpolation scheme from a small sample of interpolation points.



(a) Visualization of the centered factor.

(b) Visualization for the analytic factor.

Figure 3.2: A visualization of the oscillatory behaviour for both factors in the IFGF method for multiple source box sizes.

This strategy significantly reduces the number of evaluations of the Green's function. In order to apply this factorization across the full domain, we must first partition the domain into manageable regions where the assumptions of locality and smoothness hold. This naturally leads to a hierarchical partitioning scheme, which involves building an octree structure over the source points, as is done in the FMM [5] [6]. Each of the regions corresponds to a cubic partition in space, and the leaf nodes are small enough, such that the Green's function does not vary too rapidly across them. Section 3.2 will describe this partitioning scheme in detail.

3.2 Spatial Partitioning

To efficiently apply the Green's function factorization discussed in Section 3.1, the domain containing the source points must be spatially subdivided in a way that preserves locality and facilitates interpolation. The IFGF method achieves this using a hierarchical spatial structure known as an octree. An octree is a recursive partitioning of three-dimensional space into cubic regions called *boxes*. The root node of the octree is a single box that contains all source points. This box is then subdivided into eight child boxes of equal side length, and this process is repeated recursively until the side length of the boxes reaches a target resolution. The number of levels in the tree is denoted by D, with level $d = 0$ corresponding to the root node and $d = D$ corresponding to the leaf nodes. The octree structure is illustrated in Figure 3.3. Note, in the figure, that the octree is shown in 2D for simplicity, but the actual implementation is 3D – as such each parent spawns 4 children instead of 8.

The depth of the octree is set by a criterion, which is based on the electrical size of each box. The boxes on level $d = D$ of the octree have size $H_d = \frac{\lambda}{4}$ [5]. This mimimum box-size is a hyperparameter of the model and a discussion on why this particular value was chosen will follow in Section 3.3.1.

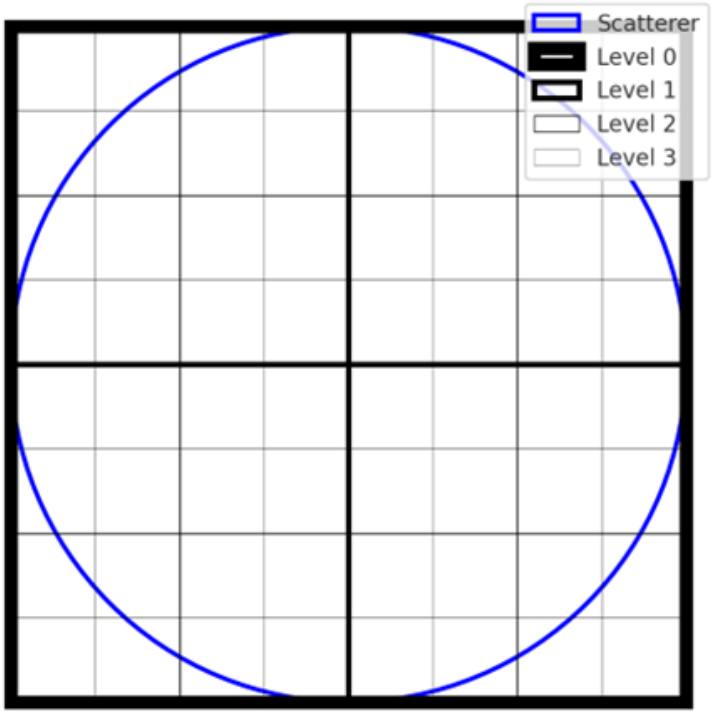


Figure 3.3: A 2D illustration of the octree structure used in the IFGF method.

There are three main steps in the IFGF algorithm, which will be discussed in more detail in Section 3.4, however the creation of the octree structure is the first step – the downward pass. Once the tree is built, it is important to note that only a small fraction of the leaf boxes will contain source points. On the lowest level of the octree, the full, unfactored Green's function is evaluated between the source points and the target points for neighbouring boxes. This step is essential for the IFGF method, as it allows for the accurate capturing of the close-proximity interactions which dominate the Green's function integral. For points located more than one box away (non-neighbouring points), the interactions between points is comparatively weaker and, therefore, the full Green's function with distant points can be approximated as a product of the analytic factor, found using interpolation, and the centered factor between the centers of the boxes.

During the last step of the IFGF algorithm, the tree is traversed in a bottom-up manner. Interactions between distant boxes are evaluated by accumulating the contributions from *cousin boxes* – which are the children of the neighbouring parent boxes. This process is repeated until the root node is reached, at which point the full Green's function interactions have been evaluated (without explicitly filling the interaction matrix). Figure 3.4 shows the neigbouring and cousin boxes for the 2D IFGF.

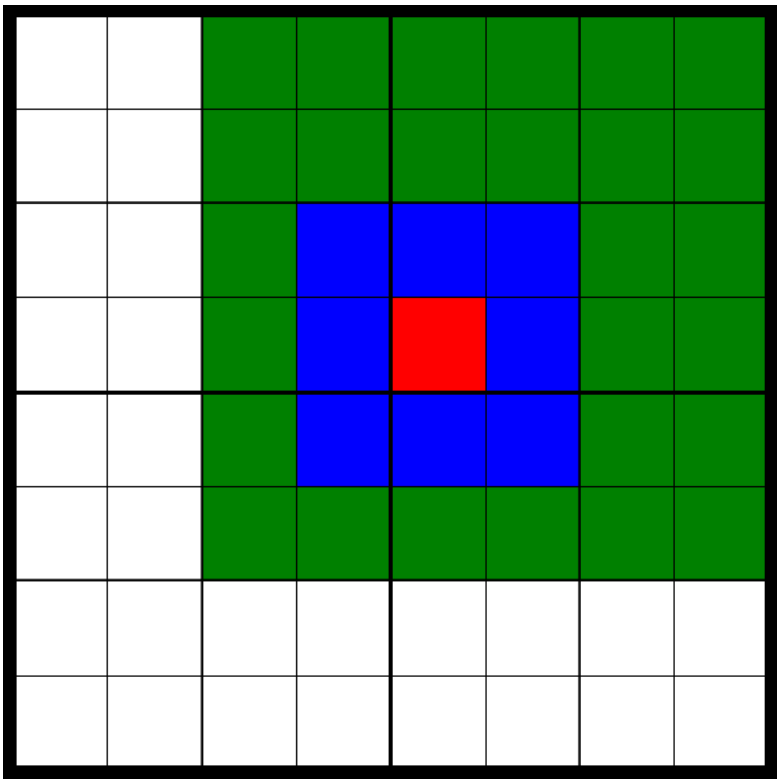


Figure 3.4: A 2D illustration of the cousin and neighbouring boxes in the IFGF method. The red box is the target box, the blue boxes are the neighbouring boxes, and the green boxes are the cousin boxes.

3.3 Interpolation

The efficiency of the IFGF method lies in its ability to approximate the analytic factor g_s at distances more than one box away from the source box (since we apply a direct evaluation of the Green's function for points within the same or neighbouring boxes). This section describes the interpolation scheme, the construction of *cone domains*, over which we select interpolation points, and how the Green's function is approximated using this scheme.

3.3.1 Cone Domains

In the IFGF method, interpolation is applied at all levels $d \leq D$ of the octree. At each level, for a given source box with side length H_d, a cone domain is defined that emanates from the center of the box. These cones capture the directions in which target points (or cousin boxes) may lie.

At the leaf level ($d = D$), the domain surrounding the box is partitioned into a set of cones, each aligned with a specific angular direction. In three dimensions, this results in a collection of conical sectors that tile the far field. Each cone originates at the center of the box and spans a fixed angular width in both azimuthal (θ) and polar (ϕ) directions. The leaf nodes in the octree are initialized with four cone domains [5].

the first interpolation points are placed within each cone at a radial distance of $\eta = \frac{\sqrt{3}}{2} H_d$ which corresponds approximately to the distance from the box center to the corners of the nearest cousin boxes. This guarantees that the interpolation is performed only in regions sufficiently far from the source box.

The interpolation grid within each cone consists of N_θ points in the azimuthal direction θ, N_ϕ points in the polar direction ϕ, and N_r points along the radial direction r. The total number of interpolation points per cone is given by $N_{\text{interp}} = N_\theta \cdot N_\phi \cdot N_r$.

Angular interpolation points are distributed within the angular extent of each cone. Radial interpolation points are distributed in the interval $[\eta, 2\eta]$, forming a band just beyond the source region, within the confines of the cousin boxes. The exact locations of the points are selected using a Chebyshev interpolation scheme described in Section 3.3.2.

Cone Refinement

Unlike the octree data structure, which is partitioned during the downward pass of the IFGF algorithm, the cone domains are assigned during an upward traversal of the octree. At the $d = D$ level, four cone domains are initialized (although we do not use cones at the leaf level). The spans of the cone domains are given by $\Delta_D^\theta = \frac{\pi}{2}$, $\Delta_D^\phi = \frac{\pi}{2}$, and $\Delta_D^r = \eta$.

Now, moving up the octree, if the parent box is a *electrically-large box*, i.e., $\kappa H \geq 1$, the cone

domains are split into eight smaller cones, each with $\Delta^{\theta}_{d-1} = \frac{\Delta^{\theta}_{d}}{2}$, $\Delta^{\phi}_{d-1} = \frac{\Delta^{\phi}_{d}}{2}$, and $\Delta^{r}_{d-1} = \frac{\Delta^{r}_{d}}{2}$. The rationale for this is as follows: if the parent box is electrically comparable in size to the wavelength, the Green's function will vary rapidly outside the box, in the interpolation domains, and therefore the cone domains must be refined to capture this variation. Since we started with a box size of $H_d = \frac{\lambda}{4}$, the cone domains are refined two-levels up from the leaf level. This is also the rationale for selecting $H_d = \frac{\lambda}{4}$ as the minimum box size – it allows us to use a minimum and known number of interpolation points in the first interpolation step.

Note, if the parent box is *electrically small*, i.e., $\kappa H < 1$, the cone domains are not refined. Furthermore, since not every box contains a source point, the boxes without any source points have their cones pruned from the octree. This is done to reduce the number of interpolation points and to bound the memory usage to $\mathcal{O}(N \log N)$. The pruning is done in-place during the cone construction upward pass.

3.3.2 Interpolation Scheme

To approximate the analytic factor $g_s(\vec{r}, \vec{r}\,', \vec{r}_s)$ within each cone domain, the IFGF method uses tensor-product Chebyshev interpolation. In one dimension, the interpolation nodes are chosen as the Chebyshev nodes of the first kind, given by $x_j = \cos\left(\frac{2j-1}{2N}\pi\right)$ for $j = 1, \ldots, N$, and mapped to the physical coordinate intervals for r, θ, and ϕ.

The interpolation is then expressed as a weighted sum over values of the analytic factor at precomputed points $\vec{r}_j$, using Lagrange basis functions $\ell_j(\vec{r}')$ [2]:

$$g_s(\vec{r}, \vec{r}\,', \vec{r}_s) \approx \sum_{j=1}^{N_{\text{interp}}} \ell_j(\vec{r}') \cdot g_s(\vec{r}, \vec{r}_j, \vec{r}_s). \tag{3.2}$$

This approximation is used in combination with the centered factor to reconstruct the full Green's function, as will be outlined in Section 3.4. Because the analytic factor is smooth in the far field, high accuracy can be achieved with a small number of interpolation points.

3.4 Algorithm Overview

As mentioned briefly before, the IFGF method consists of three main steps: the downward pass to initialize the octree structure, the direct evaluations on the leaf nodes, and the upward pass to evaluate and aggregate the Green's function interactions from distant boxes. This section will provide an overview of each of these steps, analyze the space and time complexity of the algorithm, and discuss the potential expansion to parallel implementations.

3.4.1 Downward Pass

The downward pass of the IFGF algorithm is responsible for constructing the octree data structure and initializing the cone domains. The process begins with a single box that contains all source points. This box is recursively subdivided into eight child boxes until the desired resolution is reached, as described in Section 3.2. The octree is built in a top-down manner, where each parent box is divided into eight child boxes, and this process continues until the leaf nodes are reached.

The leaf nodes are then assigned cone domains, which are initialized with four cones. The cone domains are refined as described in Section 3.3.1 based on the electrical size of the parent box – these are computed by using an in-order walkthrough of the octree structure. The irrelevant cone domains (i.e., those without source points) are pruned from the octree, in place, to reduce memory usage.

During the in-order walk of the octree, lists are assembled for the source points. Stored on each node of the octree are the source points contained within the box, the indicied of the neighbouring boxes, as well as the indicies of the cousin boxes. The cousin box list is referred to as the *interaction list*, since it contains the information required to aggregate the Green's function interactions from the cousin boxes. This list is used in the upward pass of the algorithm to evaluate the Green's function interactions between distant boxes, which is outlined in Section 3.4.3.

3.4.2 Direct Evaluations on the Leaf Nodes

The second step of the IFGF algorithm is to evaluate the full Green's function on the leaf nodes of the octree. This is done by computing the Green's function between all source points contained within the same box or neighbouring boxes. The full Green's function is evaluated using the standard formula given in Equation 2.15. This step is crucial for capturing the close-proximity interactions that dominate the Green's function integral. It is important to note that, unlike in the direct method for solving scattering problems, the full Green's function is not evaluated for every source-target pair. Instead, there are very few source points in each box, and not every box has neighbours that contain source points. Looking back at Figure 3.3, the scatterer's surface only intersects a small fraction of the boxes in the octree. This means that the number of direct evaluations of the full Green's function is significantly reduced compared to a direct method.

3.4.3 Upward Pass

Once the analytic factor $g_s(\vec{r}, \vec{r'}, \vec{r}_s)$ has been interpolated within each cone domain, the IFGF algorithm proceeds with an upward traversal of the octree. At each level d, the contribution of

each source box k to the field at a target point $\vec{r}$ is approximated as:

$$F_d^k(\vec{r}) = \frac{I_d^k(\vec{r})}{G(\vec{r}, \vec{r_k^d})}, \tag{3.3}$$

where $I_d^k(\vec{r})$ is the interpolated analytic factor and $\vec{r_k^d}$ is the center of box k at level d. The centered Green's function factor $G(\vec{r}, \vec{r_k^d})$ accounts for the oscillatory scaling with respect to the box center.

To propagate the field up the tree, each box at level $d-1$ accumulates contributions from the children k of its neighboring boxes. This re-centering step is computed using:

$$F_{d-1}^j(\vec{r}) = \sum_{\text{children } k \text{ of } j} \frac{G(\vec{r}, \vec{r_k^d})}{G(\vec{r}, \vec{r_j^{d-1}})} \cdot F_d^k(\vec{r}), \tag{3.4}$$

where $\vec{r_j^{d-1}}$ is the center of the parent box j. This process is recursively repeated until the root node is reached, at which point the full far-field contribution has been assembled. Of course, the direct evaluations we computed on the lowest levels of the octree are also included in the final accumulation of contributions.

3.4.4 Runtime and Memory Complexity

The first step in the IFGF algorithm is the downward pass, which involves building the three-dimensional octree structure. The octree is constructed top-down from a single box containing all source points. Each box spawns eight children boxes. In the worst case, if there are N points and each leaf box in the tree contains exactly one point, the tree has a total number of boxes given by:

$$\sum_{d=0}^{D} \frac{N}{8^{D-d}} = N \sum_{d=0}^{D} \frac{1}{8^d} = N \frac{1-(1/8)^{D+1}}{1-1/8} = \mathcal{O}(N). \tag{3.5}$$

Since, on each level of the octree, the number of points must still be N, and there are $D = \log_8(N)$ levels. The total cost of visiting each box is $\mathcal{O}(1)$. Therefore, to build the octree and to assign points, the total cost is $\mathcal{O}(N \log N)$.

Likewise, to construct cones, each box in the octree is visited exactly once. There are $\mathcal{O}(N)$ boxes in the octree, and the cost of assigning the cone domains is $\mathcal{O}(1)$. The cones, however, are refined by a factor as we move up the tree. Since there are $\mathcal{O}(\log N)$ levels in the tree, the total cost of assigning the cone domains is $\mathcal{O}(N \log N)$.

Next, moving onto the direct evaluations on the lowest level. Each leaf box has a finite number of neighbouring boxes, as depicted in Figure 3.4. Since the number of interpolation points per box is bounded, the cost of evaluations for each node in the octree is $\mathcal{O}(1)$. Therefore, the total cost of evaluating the Green's function on the lowest level is $\mathcal{O}(N)$.

Finally, the upward pass of the algorithm involves traversing the octree and aggregating contributions from cousin boxes. This step follows the same runtime logic as the downward pass. Each box is visited once, and the cost of aggregating contributions is $\mathcal{O}(1)$. Therefore, the total cost of the upward pass is also $\mathcal{O}(N \log N)$.

The memory complexity of the IFGF algorithm is determined by how many nodes are in the octree and how many points each node contains. The octree has $\mathcal{O}(N)$ nodes, and each level of the octree must contain N points in total – there are $log_8(N)$ levels. This makes the total memory complexity of the IFGF algorithm $\mathcal{O}(N \log N)$.

3.4.5 Parallelization

One of the notable strengths of the IFGF algorithm is its natural compatibility with parallel computation. Unlike methods such as the Adaptive Integral Method (AIM), which rely on global Fast Fourier Transforms (FFTs), IFGF avoids any form of global communication. Instead, all data exchanges occur locally between neighbor and cousin boxes, both in the finest-level evaluations and during the multilevel interpolation phase.

At level D, interactions are either directly computed or interpolated from cousin boxes, and these computations are entirely independent across target points. Similarly, as the algorithm proceeds through coarser levels, interpolations are performed from child cone segments to parents, with each step requiring only localized data from neighboring structures. This eliminates the need for synchronization barriers or collective operations across the domain.

Additionally, the spatial decomposition and preprocessing stages can be parallelized through recursive partitioning of space, further contributing to scalability. Recent hybrid implementations using MPI and OpenMP [10] have demonstrated strong scaling to thousands of cores with minimal communication overhead. Because the IFGF algorithm maintains a fixed number of interpolation operations per box and avoids global reductions, its runtime scales nearly linearly with the number of processing units, making it highly suitable for modern parallel computing architectures.

3.5 Limitations

While the Interpolated Factored Green Function (IFGF) method offers substantial advantages in terms of efficiency, scalability, and parallelizability, it is not without limitations. A fundamental constraint of IFGF is its reliance on the availability of an explicit, translation-invariant Green's function. This restricts its applicability to problems set in homogeneous media; in cases involving inhomogeneous or anisotropic materials, where the Green's function varies spatially or lacks a closed-form expression, the interpolation strategy central to IFGF cannot be directly applied. Furthermore, although the method performs well across a wide range of frequencies, the choice of interpolation parameters such as the number of levels, radial and angular discretization, and cone segmentation still requires manual tuning to achieve optimal performance. While these parameters are fixed for the purpose of ensuring $\mathcal{O}(N \log N)$ complexity, they can affect both accuracy and runtime depending on the problem geometry and wavenumber. Finally, in the static case $\kappa = 0$, while IFGF remains efficient, the method does not yet exploit the potential for $\mathcal{O}(N)$ complexity that is theoretically achievable with further modifications in the Laplace regime, such as those offered by the FMM [5] [15].

Chapter 4

Implementation of the IFGF Method

Chapter 3 introduced the Interpolated Factored Green Function (IFGF) Method from a mathematical and algorithmic perspective, with a focus on its hierarchical structure, interpolation strategy, and theoretical asymptotic performance. This chapter will shift gears from theory to practice. The goal is to describe the implementation of a serial IFGF library, starting from an initial Python prototype used for concept validation, and culmunating in a C++ engine designed for integration into the Robust Electromagnetic Boundary Element Library (REBEL) – which is a GMRES-based solver built by students and faculty at the University of Toronto. Throughout the chapter, attention is given to software architecture, computational strategies, and the translation of abstract algorithmic steps into concrete code.

4.1 Overview of the Implementation Strategy

The implementation of the IFGF method followed a structured, iterative strategy that prioritized correctness and the potential for integration into a larger simulation framework. As such, IFGF was written as a C++ library, whose functions were accessible by external users, while its internal intricies were, as best as possible, hidden from the user. The process began with a lightweight Python prototype in two-dimensions. Once the data structures were working as planned, this was expanded to three-dimensions. The Python module served as a reference model for validating core algorithmic ideas, such as Green's function factorization, cone-based interpolation, and hierarchical evaluation. Once the mathematical behaviour was well-understood and tested in Python, development shifted to a compiled C++ library. A python implementation is not practical for large-scale simulations, as there is a significant amount of overhead associated with its interpreted nature. For small simulations, up to 1000 points, however, Python worked well.

Figure 4.1: Overview of the steps taken to implement the IFGF method. The process began with a Python prototype, which was then translated into a C++ library. The final step involved integrating the C++ library into the GMRES-based solver and compairing the method to classical acceleration techniques.

Figure 4.1 provides an overview of the steps taken to implement the IFGF method. The process began with a Python prototype, which was then translated into a C++ library. The final step involves integrating the C++ library into the GMRES-based solver and comparing the method to classical acceleration techniques. Between each step were a series of unit tests to ensure correctness and performance. Each model was tested against the computationally expensive, direct solution, to the interaction matrix-vector product.

4.2 Python Prototype Development

The first working implementation of the IFGF method was developed in Python. The goal of this prototype was to validate the mathematical formulation and explore the structure of the algorithm in a flexible, readable environment. Although Python is not suitable for high-performance applications, it was invaluable for testing the ideas behind IFGF and for quickly debugging the different components. With careful design and small problem sizes (typically up to 1000 points), the prototype performed well enough to confirm the method's correctness.

The prototype made extensive use of NumPy arrays for point storage, matrix operations, and interpolation grids. All source and target points were stored as `np.ndarray` objects, which allowed for fast, vectorized evaluations of pairwise distances, kernel functions, and barycentric interpolation weights. List structures were used throughout the octree to hold child boxes and interpolation cones. When constructing the interaction list for each box—i.e., the set of neighboring and cousin boxes—the code used Python `sets`. This made it easy to eliminate duplicates and quickly check whether a given box was already in the list, without having to write additional logic.

Each major stage of the IFGF algorithm was handled by a specific Python class. The top-level interface was the `IFGF` class, which wrapped the entire algorithm. It was initialized with source points, target points, and a `Kernel` object. The user then called `evaluate(weights)` to compute the potential at each target. This call triggered the three core steps of the method: building the octree and cones, computing direct interactions at the finest level, and then recur-

sively interpolating values upward through the tree. A minimal example of this interaction is shown in Algorithm 1.

Algorithm 1 External interface for implementing and calling IFGF as user

```
kernel ← Kernel(wavenumber)
ifgf ← IFGF(sources, targets, kernel)
u ← ifgf.evaluate(weights)
```

The `Octree` class implemented the hierarchical spatial decomposition. Each box in the tree was represented by a `BoundingBox` object, which computed its center, side length, and stored the indices of the points it contained. The splitting process was handled recursively by `Octree.split()`, which created up to eight children per box and assigned points accordingly. Each box also computed its interaction list using `compute_interaction_list()`, which returned a set of geometrically nearby boxes.

The interpolation system was managed by the `Cone` class. Each cone held Chebyshev nodes in both radial and angular directions, stored as precomputed NumPy arrays. The cone also contained logic to refine itself based on the electrical size of its associated box. Interpolation was handled by the `Interpolation` module, which constructed low-order approximations of the analytic factor using tensor-product Chebyshev grids and barycentric weights.

The `Kernel` class wrapped the physics of the scattering problem. It exposed a common `evaluate(source_points, target_points, weights)` interface for both all kernel interactions. This method supported vectorized and pointwise evaluations. It was used at the finest level to compute neighbor interactions, and served as the reference when validating IFGF results against the direct $\mathcal{O}(N^2)$ matrix-vector product.

Intermediate data structures were passed between classes as NumPy arrays, Python dictionaries, and sets. For example, the analytic field samples computed at each cone were stored in a dictionary, with the box ID and cone index serving as the key. This allowed constant-time lookup during the upward interpolation phase, and made it easy to modularize the different steps in the algorithm.

A simplified pseudocode version of the evaluation routine is shown on the following page in Algorithm 2. It closely mirrors the actual organization of the Python implementation, and captures the overall flow of the algorithm [5].

Algorithm 2 IFGF Evaluation Routine)

function EVALUATE($\vec{w}$)

▷ Step 1: Downward pass (build cones, interaction lists)

  for $d = 1$ to D **do**

    for $B \in$ `Octree.level`(d) **do**

      B`.generate_cones()`

      B`.compute_interaction_list()`

    end for

  end for

▷ Step 2: Direct evaluations on the lowest level

  for $B \in$ `Octree.level`(D) **do**

    for $x \in B$`.neighbor_targets` **do**

      $u[x] \leftarrow u[x] +$ `Kernel.evaluate`$(x, \vec{r}_s, \vec{w})$

    end for

    for $C \in B$`.cones` **do**

      for $x \in C$`.interpolation_points` **do**

        $F_C[x] \leftarrow$ `Kernel.evaluate`$(x, \vec{r}_s, \vec{w})$

      end for

    end for

  end for

▷ Step 3: Upward interpolation to parents and targets

  for $d = D$ downto 1 **do**

    for $B \in$ `Octree.level`(d) **do**

      for $x \in B$`.cousin_targets` **do**

        $u[x] \leftarrow u[x] +$ `Interpolate`(F_C, x)

      end for

      if $d > 1$ **then**

        $P \leftarrow B$`.parent`

        for $C \in P$`.cones` **do**

          for $x \in C$`.interpolation_points` **do**

            $F_P[x] \leftarrow F_P[x] +$ `Interpolate`(F_C, x)

          end for

        end for

      end if

    end for

  end for

end function

4.3 C++ Library Design

Following the successful prototyping of the IFGF method in Python, a C++ implementation was developed to support larger-scale simulations and improved performance. The algorithm's structure—including hierarchical traversal, interpolation, and kernel evaluation—translated naturally into C++, and care was taken to preserve the modular layout established in the Python version.

The code was compiled using the MinGW toolchain on Windows, which provided support for standard optimization flags and good compatibility with local development tools. Each major component from the Python implementation was translated into a corresponding `.cpp` and `.hpp` file, maintaining a one-to-one correspondence with the original modules. This included classes for the IFGF interface, octree construction, bounding boxes, interpolation cones, kernel evaluation, and general utilities.

Eigen was selected as the linear algebra engine due to its expressive syntax and strong alignment with NumPy. The use of Eigen made it straightforward to port vectorized operations from Python to C++, while maintaining clarity and performance. Source and target points were stored in `MatrixX3d` format, with each row representing a 3D coordinate. Field values and weights were stored as `VectorXd` or `MatrixXd`, depending on the use case. Chebyshev grids and interpolation tables were also constructed using Eigen matrices. To simplify syntax across the codebase, a lightweight wrapper was written around Eigen to expose frequently used operations such as dot products, slicing, and interpolation. This modular interface mirrored REBEL's handling of LAPACK and allowed the linear algebra backend to remain interchangeable if needed.

Interaction data between boxes – such as target relationships and field contributions – was tracked using hash maps via `std::unordered_map`. This provided average-case constant-time access during interpolation and recursion. An earlier version of the code used `std::map`, but since it is backed by a red-black tree with $\mathcal{O}(\log N)$ lookup time, it was replaced to improve runtime. These maps were used throughout the tree traversal to associate box identifiers with cone data, interpolated coefficients, and parent-child relationships.

The final result was a standalone C++ IFGF library that retained the algorithmic structure of the Python version. It was written with integration in mind and is designed to operate as a drop-in module for boundary integral solvers such as REBEL, as discussed later in Section 4.4.

4.4 Integration with GMRES

The Interpolated Factored Green Function (IFGF) method operates on discrete point cloud data. It accepts a set of source and target coordinates in $\mathbb{R}^3$, along with scalar complex weights, and

returns the resulting scalar potential field. This input format differs from traditional boundary element method (BEM) solvers used in computational electromagnetics, which operate on surface meshes and represent unknown surface currents using vector-valued basis functions. These currents are expanded as a weighted sum of known functions over triangle elements, as shown in Equation 2.18.

To integrate IFGF into this setting, the mesh-based formulation must be approximated as a discrete source model. This was achieved by collapsing each basis function into a single point located at the centroid of its supporting triangle. The surface current supported on the triangle was assumed to be uniformly distributed and aligned with the orientation of the basis vector $\vec{f}_n$ [3] [12]. Under this approximation, the continuous current is represented as a sum of vector-valued delta functions:

$$\vec{J}(\vec{r}) \approx \sum_{n=1}^{N} w_n \vec{f}_n \, \delta(\vec{r} - \vec{r}_n), \tag{4.1}$$

where $\vec{r}_n$ is the centroid of the nth triangle and $\delta(\cdot)$ is the Dirac delta function. This effectively converts a mesh-based formulation into one compatible with IFGF's pointwise interface.

Since IFGF only evaluates scalar fields, each component of the magnetic vector potential $\vec{A}(\vec{r})$ must be computed independently. To do this, IFGF is called three times—once for each Cartesian direction—using scalar weights defined as $w_n^{(i)} = w_n f_n^{(i)}$ for $i \in \{x, y, z\}$. Each call returns the scalar potential due to the current projected in that direction. The full vector field is reconstructed by combining the components:

$$\vec{A}(\vec{r}) = A_x(\vec{r})\,\hat{x} + A_y(\vec{r})\,\hat{y} + A_z(\vec{r})\,\hat{z}. \tag{4.2}$$

This enables IFGF to serve as a drop-in replacement for the matrix-vector product in the GMRES iteration. The overall integration strategy is summarized in Algorithm 3, which outlines how IFGF is called within each iteration.

Figure 4.2 illustrates the conversion from mesh to point cloud. Each red cross represents the centroid of a triangle in the mesh and serves as a discrete source point for the IFGF evaluation.

4.5 Testing and Validation

The IFGF implementation was validated against a brute-force reference solution computed via direct kernel evaluation. For a given source and target configuration, both methods were used

Algorithm 3 GMRES with IFGF-based Matrix-Vector Product

Initialize residual $r_0 = b - Ax_0$, $v_1 = r_0/\|r_0\|$
for $j = 1$ to m **do**
    // Matrix-vector product via IFGF
    for $i \in \{x, y, z\}$ **do**
        Compute directional weights $w_n^{(i)} = w_n f_n^{(i)}$
        $A_i = \texttt{IFGF}(r_n, t_m, w_n^{(i)})$
    end for
    Combine result: $\vec{A} = A_x\hat{x} + A_y\hat{y} + A_z\hat{z}$
    Orthogonalize $\vec{A}$ against $v_1, \ldots, v_j$
    Normalize and store v_{j+1}
end for
Solve least-squares problem to get Δx
Update solution: $x = x_0 + V\Delta x$

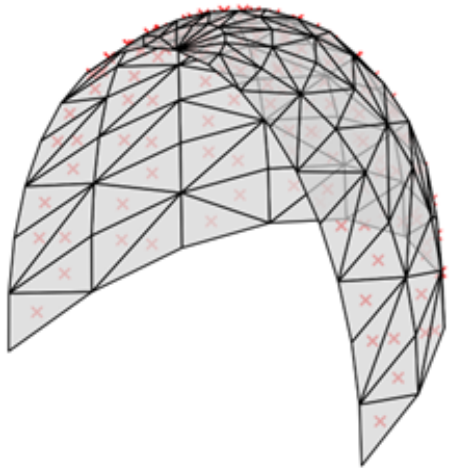


Figure 4.2: Surface mesh used in REBEL, with red crosses indicating triangle centroids. These points serve as the discrete source locations in the IFGF method.

to compute the scalar potential at all observation points. The accuracy of IFGF was quantified using the relative root-mean-square error (RMSE), defined as:

$$\epsilon = \sqrt{\frac{\sum_{i=1}^{N} \left| I_{IFGF}^{(i)} - I_{Direct}^{(i)} \right|^2}{\sum_{i=1}^{N} \left| I_{Direct}^{(i)} \right|^2}} \tag{4.3}$$

where $I_{IFGF}^{(i)}$ is the field value computed using IFGF at the ith observation point, and $I_{Direct}^{(i)}$ is the corresponding value obtained by evaluating the kernel directly.

To correctness of the algorithm, the implementation was benchmarked for runtime and memory usage across various problem sizes, validating its expected $\mathcal{O}(N \log N)$ asymptotic scaling. The results of these tests are presented in Chapter 5.

Chapter 5

Results and Performance Evaluation

This chapter presents the numerical results used to evaluate the accuracy, efficiency, and scalability of the Interpolated Factored Green Function (IFGF) implementation. The analysis covers both the initial Python prototype and the final C++ library, with comparisons made against the direct evaluation method to validate correctness. Benchmarks were conducted across problem sizes to assess asymptotic performance.

Section 5.1 begins with early validation and scaling behavior of the Python prototype. Section 5.2 outlines the hardware, compiler configuration, and test cases used for evaluation. The remaining sections present results related to accuracy (Section 5.3), runtime scaling (Section 5.4), and memory usage (Section 5.5), followed by a discussion of findings in Section 5.6.

5.1 Preliminary Python Results

Initial runtime tests were performed using the Python prototype implementation of the IFGF algorithm. The geometry consisted of a unit-radius sphere populated with randomly distributed source and target points. The wavenumber was fixed at 8π rad/m, corresponding to a wavelength of 0.25 m. The number of points N was increased in powers of two to study runtime scaling. Each source was assigned a randomly generated complex weight, uniformly sampled in magnitude and phase.

Both IFGF and a naïve direct evaluation method were applied to the same source-target configuration, and the total runtime was measured in seconds using Python's built-in timing functions. The results are plotted in Figure 5.1, showing runtime as a function of N on a log-log scale. As expected, the IFGF method demonstrates a more favorable asymptotic scaling than the direct evaluation, particularly as the number of points increases.

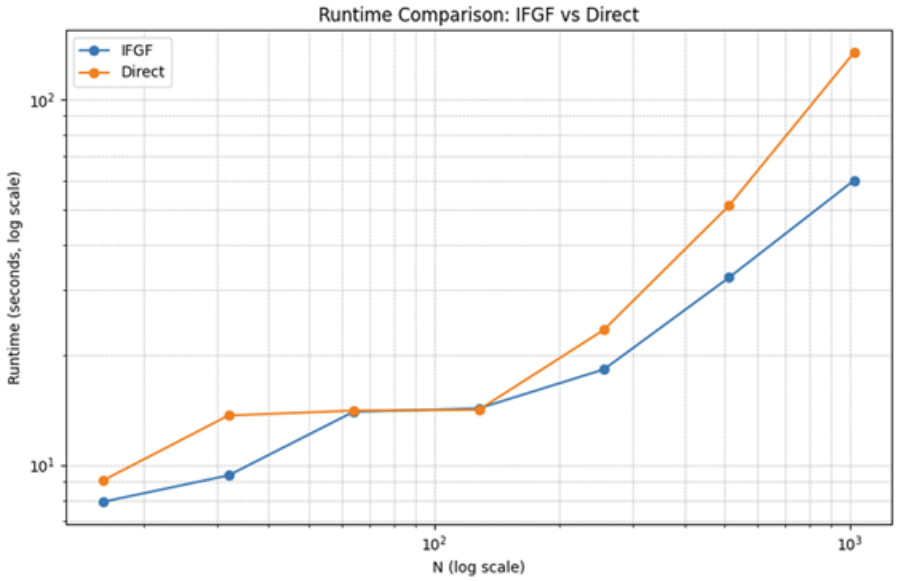


Figure 5.1: Runtime comparison between IFGF and direct evaluation for a spherical point cloud with wavenumber 8π rad/m. Source weights were randomly distributed. Both methods were tested across increasing point counts N.

5.2 Experimental Setup

This section describes the experimental conditions under which the C++ IFGF implementation was evaluated. Benchmarks were designed to measure runtime, memory usage, and accuracy across varying problem sizes. All tests were conducted on synthetic geometries with controlled source distributions and wavenumber values. The goal was to assess the algorithm's scalability and numerical behavior under simple operating conditions.

5.2.1 Hardware and Compiler

All C++ benchmarks were performed on a Dell XPS 15 running Windows 11, equipped with an Intel Core i7 processor. The code was compiled using the MinGW-w64 toolchain with optimization flags enabled. No parallelism was used; all tests reflect serial performance.

5.2.2 Test Geometry

All C++ tests were conducted on point clouds sampled from the surface of a sphere. Source and target points were randomly distributed over the unit sphere using a uniform sampling strategy in spherical coordinates. For each test case, the radius of the sphere was set to $R \in \{1, 2, 4, 8, 16, 32\}$ meters. A constant wavenumber of $k = 2\pi$ [rad/m] was used for all configurations.

The number of source points N was scaled proportionally to the surface area, i.e., $N \propto R^2$. This keeps the density of the points, and therefore the resolution, constant across tests. Specifically, values of $N \in \{1, 4, 16, 64, 256, 1024\} \times 10^3$ were used. Each point was assigned a complex-valued coefficient sampled uniformly from the interval $[0, 1]$, representing a random source distribution over the surface.

Figure 5.2 illustrates a representative geometry used in the evaluation, corresponding to a sphere of radius $R = 8$ m and $N = 64{,}000$ source points.

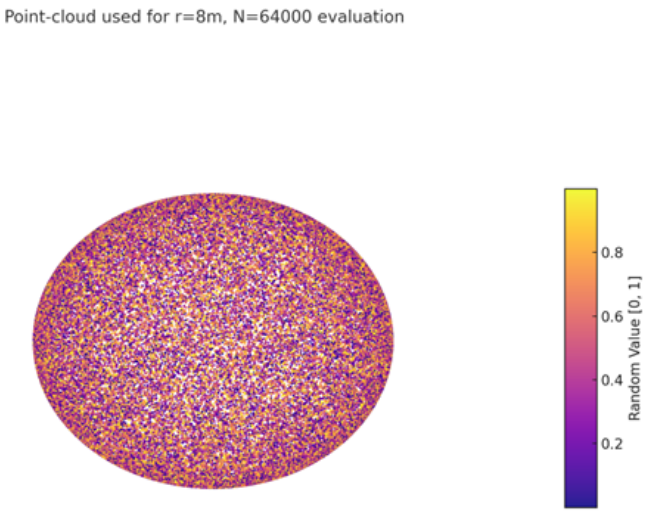


Figure 5.2: Randomly distributed point cloud on a sphere of radius $R = 8$ m, with $N = 64{,}000$ source points. Source coefficients are color-mapped by amplitude in $[0, 1]$.

5.3 Accuracy Validation

To quantify the accuracy of the IFGF implementation, its output was compared against the direct evaluation of the kernel function, which computes the interaction between all pairs of source and target points. Due to the $\mathcal{O}(N^2)$ cost of the direct method, this validation was restricted to the first three data points in the C++ test suite.

The relative root-mean-square error (RMSE), as defined in Section 4.5, was used to assess accuracy. The results are summarized in Table 5.1, where the error remains within 2% across all tested configurations.

Table 5.1: RMSE error between IFGF and direct evaluation for small test cases.

Wavenumber [rad/m]	Radius [m]	Points	RMSE Error ($\times 10^{-3}$)
2π	1	1,000	4.10
2π	2	4,000	6.80
2π	4	16,000	16.20

5.4 Runtime Performance

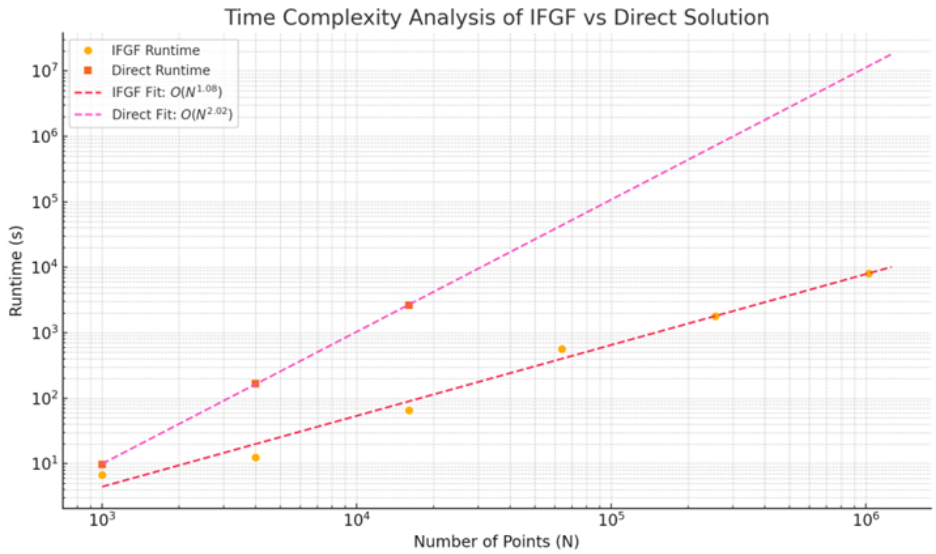


Figure 5.3: Empirical runtime scaling of IFGF and direct evaluation. Dashed lines show least-squares linear fits. IFGF scales approximately as $\mathcal{O}(N^{1.08})$, while the direct method scales as $\mathcal{O}(N^{2.02})$.

The runtime performance of the IFGF method was evaluated against a direct $\mathcal{O}(N^2)$ kernel summation baseline. For each test case, the total wall time required to compute the interaction at all target points was recorded, including both setup and evaluation phases.

Figure 5.3 presents the runtime data on a log-log scale, with a linear fit applied to each data set. The fitted curve for the IFGF method closely follows a trend of $\mathcal{O}(N^{1.08})$, while the direct evaluation exhibits the expected $\mathcal{O}(N^{2.02})$ complexity. These results align well with theoretical predictions and demonstrate that IFGF achieves log-linear scaling in practice.

5.5 Memory Performance

The peak memory usage of the IFGF implementation was measured during execution using the `psapi.h` system interface, linked via the MinGW toolchain on Windows. Memory consumption was recorded at runtime and plotted against problem size N, representing the number of

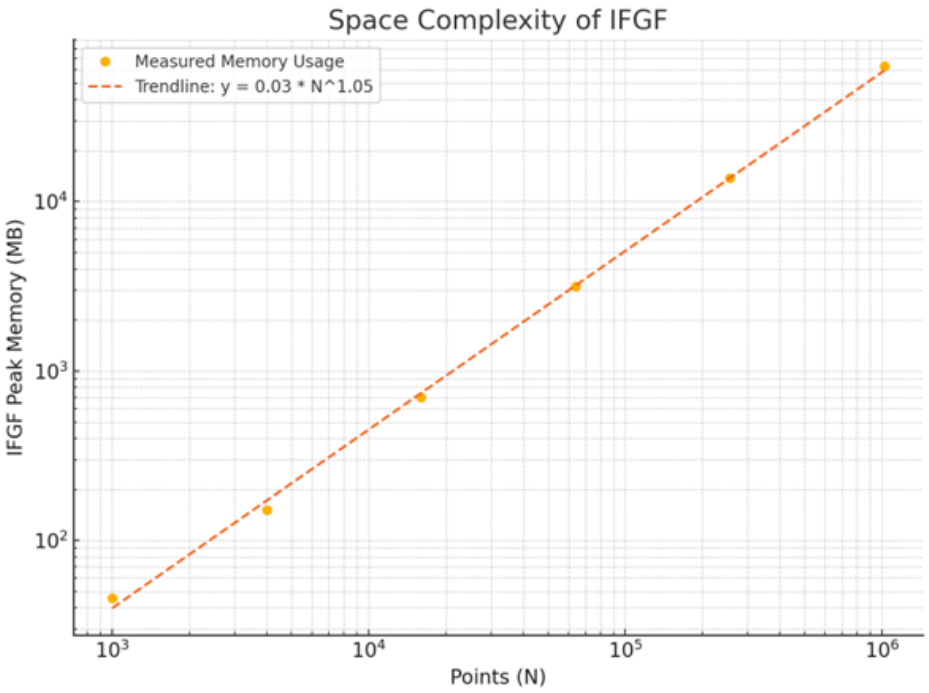


Figure 5.4: Peak memory usage of IFGF for varying N. Trendline indicates log-linear space complexity with slope 1.05. Measurements obtained using `psapi.h` with MinGW.

source and target points.

Figure 5.4 shows the peak memory usage in megabytes (MB) on a log-log scale. A least-squares linear fit was applied to the data, yielding a scaling trend of approximately $\mathcal{O}(N^{1.05})$. This result confirms that the IFGF method maintains near-linear memory complexity, consistent with theoretical predictions. The memory footprint remained well-bounded even for problem sizes approaching one million points.

5.6 Discussion of Results

The numerical experiments presented in this chapter provide empirical validation of the IFGF algorithm's theoretical performance claims. Across both Python and C++ implementations, the runtime scaling consistently approached $\mathcal{O}(N \log N)$ or better, with least-squares fits indicating effective complexity of $\mathcal{O}(N^{1.08})$ in the C++ evaluation. This confirms the algorithm's ability to reduce the asymptotic cost of matrix-vector multiplication compared to the direct $\mathcal{O}(N^2)$ baseline.

In terms of accuracy, the relative root-mean-square error (RMSE) remained on the order of 10^{-2} or lower for test problems, even under moderate electrical sizes. This demonstrates that

the combination of Chebyshev interpolation, surface cone segmentation, and hierarchical evaluation maintains acceptable precision while significantly reducing computational overhead. The increase in error with problem size is expected, as the interpolation error grows slightly with higher wavenumber and increased cone depth. However, the tradeoff between precision and cost remains favorable.

Memory scaling results exhibited log-linear behavior, with peak usage growing as approximately $\mathcal{O}(N^{1.05})$. This trend reflects the expected cost of storing the octree structure, cone domains, and intermediate interpolation values, all of which grow proportionally with the number of points and the number of levels in the hierarchy. The memory footprint remained well within tractable limits even for large-scale problems.

Overall, the results confirm that IFGF can be effectively applied in electromagnetic solvers such as REBEL, enabling scalable evaluations with manageable resource requirements and controllable accuracy. Further improvements can be obtained through parallelization and adaptive cone refinement, which were not explored in this implementation but remain future directions.

Chapter 6

Conclusion and Future Work

This thesis presented the development of the Interpolated Factored Green Function (IFGF) method as a fast algorithm for evaluating boundary integral operators in electromagnetic simulations. Beginning from its mathematical formulation, the method was validated through a Python prototype and implemented in a high-performance C++ library. The final implementation is being integrated into REBEL, a GMRES-based electromagnetic solver developed at the University of Toronto, where it will replace the classical matrix-vector product acceleration techniques.

This thesis work resulted in the creation of a C++ compiled library that can easily be integrated into existing solvers, with a minimal interface for users.

Results confirmed the algorithm's asymptotic efficiency, with empirical runtimes scaling approximately as $\mathcal{O}(N \log N)$ and memory usage following similar growth. Accuracy remained within acceptable limits for electrically large problems, and the modular design enabled direct integration into iterative solvers such as GMRES.

Future work will focus on completing the integration of IFGF into the full REBEL simulation pipeline, including automatic support for complex CAD-generated geometries. This will allow direct benchmarking of IFGF against other established fast methods, such as the Fast Multipole Method (FMM) and Adaptive Integral Method (AIM), across realistic, large-scale scattering scenarios. In parallel, future extensions may explore multi-threaded and distributed implementations to further improve performance, as well as adaptivity in the interpolation scheme to better handle challenging geometric features.

Once validated for perfect electrical conductors (PEC), the method will be extended to handle lossy and dispersive materials, broadening its applicability to a wider range of electromagnetic problems. The ultimate goal is to establish IFGF as a robust and efficient tool for large-scale

electromagnetic simulations, paving the way for advancements in antenna design, radar systems, and integrated circuit analysis.

References

[1] M. Commens, M. Swinnen, and K. Damalou, "The challenge of electromagnetic modeling and simulation for silicon interposers in 2.5d/3d-ic chip designs," Ansys Blog, November 2022.

[2] J. Paul and C. Sideris, "Algorithmic acceleration of the chebyshev-based boundary integral equation method for 3d maxwell problems using the interpolated factored green function," in *Proc. Int. Appl. Comput. Electromagn. Soc. Symp. (ACES)*, Orlando, FL, USA, 2024, pp. 1–1.

[3] P. Triverio, "Integral equation methods for computational electromagnetism: Advanced techniques and applications," Master's thesis, University of Toronto, Nov. 2023.

[4] W. C. Chew and L. J. Jiang, "The evolution of large-scale computing: Past achievements, present challenges, and future directions," *Proc. IEEE*, vol. 101, no. 2, pp. 227–241, Feb. 2013.

[5] C. Bauinger and O. P. Bruno, "Interpolated factored green function method for accelerated solution of scattering problems," *J. Comput. Phys.*, vol. 430, p. 110095, Apr. 2021.

[6] V. R. N. Engheta, W. D. Murphy, and M. Vassiliou, "The fast multipole method for electromagnetic scattering computation," *IEEE Trans. Antennas Propag.*, vol. 40, no. 6, pp. 634–641, 1992.

[7] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "Aim: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Sci.*, vol. 31, no. 5, pp. 1225–1251, 1996.

[8] S. Sharma and P. Triverio, "Aim: An extended adaptive integral method for the fast electromagnetic modeling of complex structures," *IEEE Trans. Antennas Propag.*, vol. 69, no. 12, pp. 8603–8617, 2021.

[9] S. Sharma, U. R. Patel, and P. Triverio, "Accelerated electromagnetic analysis of interconnects in layered media using a near-field series expansion of the green's function," in

Proceedings of the 27th IEEE Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS), San Jose, CA, USA, 2018, pp. 197–200.

[10] C. Bauinger and O. P. Bruno, "Massively parallelized interpolated factored green function method for high-performance scattering simulations," *J. Comput. Phys.*, vol. 475, p. 111837, 2023.

[11] D. K. Cheng, *Field and Wave Electromagnetics: Foundational Principles and Applications*, 2nd ed. Addison-Wesley, 1989.

[12] W. C. Gibson, *The Method of Moments in Electromagnetics: Advanced Computational Techniques*, 3rd ed. Boca Raton, FL: CRC Press, 2022.

[13] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Stat. Comput.*, vol. 7, no. 3, pp. 856–869, Jul. 1986.

[14] R. Beatson and L. Greengard, "A short course on fast multipole methods," 2000, lecture notes.

[15] C. B. E. Jimenez and O. P. Bruno, "Ifgf-accelerated integral equation solvers for acoustic scattering: Theory and applications," *arXiv preprint*, vol. arXiv:2112.06316, 2022.