Hierarchical Matrix Methods for Boundary Element Problems

by

Yong Da Li

Supervisor: Piero Triverio
April 2023

B.A.Sc. Thesis

Division of Engineering Science
UNIVERSITY OF TORONTO

Abstract

Hierarchical matrices are a nested matrix structure that allows the far-field field blocks of the dense Method of Moments (MoM) matrix to be low-rank approximated, while keeping the near-field blocks as unmodified dense blocks. The low-rank compression is performed by Adaptive Cross Approximation (ACA), seeing up to 90% memory compression and speedup in direct solve via LU factorization from $O(N^3)$ to $O(N^{2.332})$. These formulations are purely algebraic and do not make any assumptions about the kernel or structure regularity. Thus they are applicable to a wide range of problems with minimal modification to the formulation, as opposed to the Adaptive Integral Method (AIM) or Fast Multipole Method (FMM) which are kernel dependent. In this thesis, the electrostatic formulation of the MoM is accelerated by ACA compression of hierarchical matrices and its performance characteristics are investigated.

Acknowledgements

I would like thank Prof. Piero Triverio for being a kind and nurturing thesis supervisor, even when my progress is slower than expected. I would like to thank my friend Jasper Hatton (who also did his thesis with Prof. Triverio) for providing some template Gmsh code and discussing debugging ideas. I would like to thank Yongzhong Li and Damian Marek (graduate students under Prof. Triverio) for general software help. This thesis would be not possible with your support.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Computational electromagnetics is the study of numerical methods to solve electromagnetic problem. They are widely used to simulate electronic devices. They can be used to quickly prototype and ask "what if" scenarios before the device is fabricated [1]. This provides advantages of lower cost and faster design cycles. In semiconductor manufacturing, there is a high upfront cost to create the mask set for the design. After the masks are created, the cost per chip is roughly constant. In other words the cost to produce 10 chips is comparable 10,000 chips. In addition, the silicon design process is subject to schedule constraints especially for leading edge nodes. Fab capacity is booked up to years in advance. And the fab may only offer tape-outs once every few months. One cannot send a design to fabrication any time they want.

All these factors mean making a single test chip, characterizing it, and putting the post-silicon learnings into the next design is not suitable. Leading chip designers such as Intel may only schedule 1 or 2 test chips for the entire design lifetime of a new device. Instead, the semiconductor design industry has always used simulators to verify the correctness of their circuit before sending the designs to be fabricated.

One example of this design flow is extracting parasitic capacitances. After the circuit schematic is completed and verified to be functionally correct, the designer moves to layout. The transistors, contacts, and metal layers are laid out in physical space. From the physical layout, parasitic capacitances, resistances, and inductances can be computed. Then these unwanted values can be added to the initial the circuit simulation to provide a more accurate representation of circuit operation.

The extraction of the parasitic capacitance can be done with a computational electro-

magnetic simulator. The designer will go through many iterations of post-layout simulation, parameter extraction, tweaking the layout, and repeating. These steps should provide a good balance between speed and accuracy to enable rapid design iteration. As such, the computational tools themselves should have fast run-time, low memory usage, and produce accurate results.

1.2 Background

Maxwell's equations describe the interaction between electric and magnetic fields. They provide the mathematical basis for a wide range of electromagnetic applications such as semiconductor devices, wireless communication, and electric motors. The solution to Maxwell's equations is the goal of computational electromagnetic simulators and solvers. For simple geometries such as a perfect electric conductor (PEC) sphere, the analytic solution is known. But for geometries that are used in practice, closed form analytic solutions to Maxwell's equations rarely exist. One must resort to using numerical methods on a computer to solve Maxwell's equations. See Fig 1.1 for an overview of the numerical methods discussed in this section.

The finite element method (FEM) and the finite different time domain method (FDTD) are two popular methods for numerically solving Maxwell's equations. They both use the differential form of Maxwell's equations. The FEM subdivides a 3-D space to be solved into smaller parts called finite elements. Practically, this is achieved by constructing a mesh of the object. For each individual finite element, a boundary value problem is formulated from the differential form of Maxwell's equations that results in a system of algebraic equations. All the equations coming from the finite elements are combined into a larger system of equations that model the entire object. Then these equations are solved using variational methods that seek to minimize an associated error function.

The FDTD method solves Maxwell's equations in the time domain. The space to be solved is partitioned into a grid. Then the differential time-dependent Maxwell's equations are discretized. The differential form of the equations are used. The derivatives are approximated as differences between adjacent grid cells. Then the finite difference equations are solved in a leap-frog manner. The electric field is solved at one time step, then the magnetic fields are solved at the next time step. This process is repeated to perform a transient analysis. Or it may be continued until a steady-state behavior is formed.

Note that both of these methods require a volume discretization of the entire solution domain. This leads to a very large system of equations. Although the system is usually sparse, these volume methods usually require large amounts of computer memory (RAM)

to solve the system. They are very good at solving the fields in inhomogeneous medium or closed cavities such as waveguides. But they do not scale to scattering or radiation problems.

For scattering and radiation problems, a surface integral equation (SIE) formulation is used. And the Method of Moments (MoM) is used to numerically solve these SIEs. The MoM or sometimes referred to as the Boundary Element Method (BEM) differs from FEM and FDTD methods in that it does not require a volume discretization. The quantities of interest are the surface equivalent charges. They are only distributed on the surface. Thus, only a surface discretization is required. This is the method that this thesis will focus on.

The major drawback of the MoM formulation is that it results in a dense system. The theoretical formulation is examined in 2.1. But to give an intuition why the system is dense, it can be said that interaction of each small surface mesh element must be computed against every other surface mesh element. This is a quadratic operation, so the resulting matrix has space complexity $O(N^2)$ where N is the number of mesh elements. This is disadvantageous for a number of reasons:

1. A large amount of computer memory (RAM) is required to hold the problem in system memory as it's being worked on by the computer.

2. All matrix entries must be computed. This is called matrix "fill-in" or "assembly".

3. Matrix-vector products are very expensive, as all rows and columns must be considered resulting in $O(N^2)$ time complexity.

There has been many successful techniques developed in the last 30 years that aim to accelerate the MoM formulation. They achieve this by compressing the dense system through clever symmetries and approximations. In particular we briefly examine the adaptive integral method (AIM) and the fast multipole method (FMM). The AIM exploits the translation invariance of the Green's function kernel. The FMM approximates parts of the dense matrix using spherical wave expansion. Both of these methods are kernel dependent. When changing the environment from free space to a layered media, special development of analytical Green's functions are required. These methods typically are paired with an iterative solver such as the conjugate gradient method (CG) or generalized minimal residual method (GMRES). They're not suited for solving multiple right-hand sides (RHS) vectors which occurs when multiple target frequencies to be solved at the same time. The iterative solution scheme has to restart for every RHS vector. They may have slow convergence and typically require matrix preconditioners to speed up the convergence rate.

The method studied in this thesis is adaptive cross approximation method to produce hierarchical matrices. This method is a purely algebraic method that does not depend on a

priori knowledge of the kernel. This is a key advantage as it is more robust, able to scale from DC to high frequencies using the same formulation. The low-rank structure of the hierarchical matrix can be used to compute fast direct LU decomposition, allowing for direct solutions[2]. Iterative methods such as conjugate gradient (CG) or generalized minimal residual method (GMRES) typical use a preconditioning step to speed up convergence. But if the system is poorly conditioned, the rate of convergence may be slow and not converge to a low residual.

To date, there have been a handful of studies into the performance of hierarchical matrix methods at scale where the number of unknowns is in the 10,000 to 100,000 range [3]–[7]. Much of the development is theoretical and numerical examples are only demonstrated on smaller test cases with simple geometries. This thesis aims to investiage the

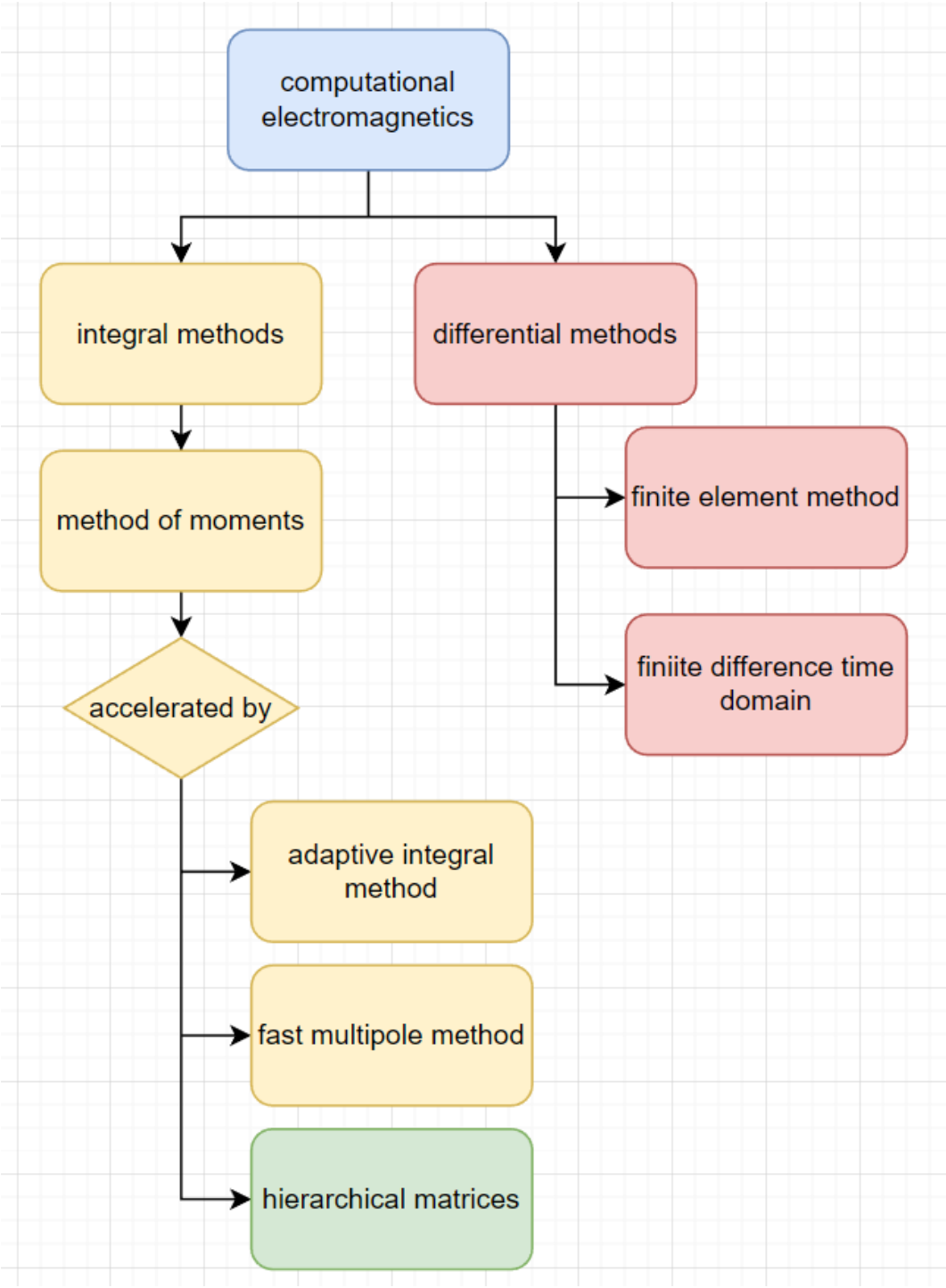

Figure 1.1: An overview of the types of numerical methods in computational electromagnetics. This thesis focuses on the yellow-colored method of moments variation, that is accelerated by the green hierarchical matrices. These methods only require a surface discretization. The red colored differential methods require a volume discretization leading to larger systems to be solved.

1.3 Adaptive Integral Method

A popular method for accelerating MoM problem is the Adaptive Integral Method (AIM) [8], first introduced by Bleszynski et al. in 1996. The $\mathbf{A}$ matrix formed by Method of Moments is split into a near-field region $\mathbf{A}_{NR}$ and far-field region $\mathbf{A}_{FR}$

$$\mathbf{A}\vec{\mathbf{x}} = \mathbf{A}^{\text{near}}\vec{\mathbf{x}} + \mathbf{A}^{\text{far}}\vec{\mathbf{x}}, \tag{1.1}$$

such that they sum to the total system $\mathbf{A}$ matrix, where $\vec{\mathbf{x}}$ is the solution vector.

The near-field matrix $\mathbf{A}^{\text{near}}$ is treated normally using the Galerkin discretization. But since it only contains the near-field interactions, most of the matrix entries in a single row (which represents a single mesh element) will be zero. Only the mesh elements within the near region of that mesh element will be populated. Thus the near-field matrix $\mathbf{A}^{\text{near}}$ is sparse matrix. And sparse matrix operations can be used to achieve nearly linear time complexity.

The far-field matrix $\mathbf{A}^{\text{far}}$ is never computed explicitly. It is always kept in its compressed form. And only matrix-vector products $\mathbf{A}^{\text{far}}\vec{\mathbf{x}}$ are used in the computation which keeps the storage complexity to $O(N)$. The compressed form of $\mathbf{A}^{\text{far}}$ doesn't store interactions of the original basis functions, but instead stores a uniform Cartesian grid of auxiliary basis functions. These auxiliary basis functions are point-like current sources chosen such that their strength approximates the far field interactions of the original basis. The original basis functions are transformed to this auxiliary grid by

$$\mathbf{A}^{\text{far}} = \mathbf{\Lambda}\mathbf{g}\mathbf{\Lambda}^T, \tag{1.2}$$

where $\mathbf{\Lambda}$ is the basis projection matrix and $\mathbf{g}$ is the Green's function matrix. The projection matrix $\mathbf{\Lambda}$ is sparse and the Green's function matrix $\mathbf{g}$ has a three-level Toeplitz structure. The Toeplitz structure is important since the matrix multiplication between a Toeplitz matrix and another vector is a discrete convolution between the two. After putting (1.2) into (1.1), we obtain

$$\mathbf{A}\vec{\mathbf{x}} = \mathbf{A}^{\text{near}}\vec{\mathbf{x}} + \mathbf{\Lambda}\mathbf{g}\mathbf{\Lambda}^T\vec{\mathbf{x}}. \tag{1.3}$$

We notice that the product $\mathbf{\Lambda}^T\vec{\mathbf{x}}$ is a vector. The multiplication of this vector with the Toeplitz matrix $\mathbf{g}$ forms the discrete convolution. Due to the translation variance of the Green's function kernel, this convolution is computing the equivalent far-field potential at the uniform grid points due the uniform grid sources which is exactly the goal.

As is known from the convolution theorem, a Fourier transform of a convolution is the point-wise product of their Fourier transforms. So this matrix-vector product $(\mathbf{g})\left(\mathbf{\Lambda}^T\vec{\mathbf{x}}\right)$ can

be accelerated by an FFT, computing the point-wise product, then transform back via the inverse FFT to the desired convolution. This leads to the final expression

$$\mathbf{A}\vec{\mathbf{x}} = \mathbf{A}^{\text{near}}\vec{\mathbf{x}} + \mathbf{\Lambda}\mathcal{F}^{-1}\left\{\mathcal{F}\left\{\mathbf{g}\right\} \cdot \mathcal{F}\left\{\mathbf{\Lambda}^{T}\vec{\mathbf{x}}\right\}\right\}, \tag{1.4}$$

where $\mathcal{F}$ is the FFT and $\mathcal{F}^{-1}$ is the inverse FFT. The solution scales in time complexity as $O(N^{1.5}\log N)$ and in memory complexity as $O(N^{1.5})$

The problem with the AIM is that it is kernel dependent. It only works for the Green's function kernel. It exploits the translation invariance of the kernel, so it is considered a physics based method. The AIM does not scale well to high frequencies requiring an increasing large number of matrix operators. It also will not work at DC zero-frequency conditions. These problems are alleviated with a hierarchical matrix method as it is a purely algebraic method with no dependence on the underlying kernel.

1.4 Fast Multipole Method

The Fast Multipole Method (FMM) was first proposed by Greengard and Rokhlin [9] in 1986 for acoustic problems. In 1992, it was first applied to electromagnetic problems by Engheta et al. with great success [10]. Since the FMM and hierarchical matrix methods are very similar, it's useful to understand the FMM to draw parallels to hierarchical matrices.

The FMM is used to rapidly compute pairwise interactions between source and observation points. Consider two clusters of points located far away from each other. One cluster is a set of N source points s_i and the other cluster is a set of N observation points o_i. To compute the electrostatic potential at one observation point, the electrostatic potential needs to be computed for all N s_i source points. The process needs to be repeated N times to compute the potential for all N observation points. This standard way of computing has time complexity $O(N^2)$.

The innovation of the FMM is that clusters of source charges sufficiently far away from observation points will appear as one lump-sum charge to the far observation point, instead of many individual charges. Instead of computing the interactions between s_i and o_i individually, the effect of all charges are lumped into one s. And the observation point is lumped into one o. The electrostatic potential is computed once. Then the results of that calculation are applied to all observation points. So by this method, all observation points in the same cluster experience exactly the same electrostatic effect. See Fig 1.2

This process is repeated at finer and finer levels of refinement until some stop condition. For example, the cluster sizes are progressively decreased until a minimum cluster size is reached. The FMM has time complexity $O(N \log N)$.

The condition that determines if clusters of points are sufficiently far enough apart to apply the FMM is called the admissibility condition. The admissibility condition is defined based on two things. The diameters of each individual cluster. And the distance between clusters. Let the cluster of source points be referred to as A and the cluster of observation points to be referred to as B.

The diameter of a cluster $diam(A)$ or $diam(B)$ is defined as the maximum distance between two points within the cluster. The distance between clusters $dist(A, B)$ is defined as the closest distance between one point from cluster A and another point from cluster B. Then the admissibility condition is

$$\max\{diam(A), diam(B)\} \leq \eta \cdot dist(A, B), \tag{1.5}$$

where η is a control parameter. A larger η parameters allows clusters of points closer together to be admissible, but lowers the accuracy of the method. Typically η is set to one.

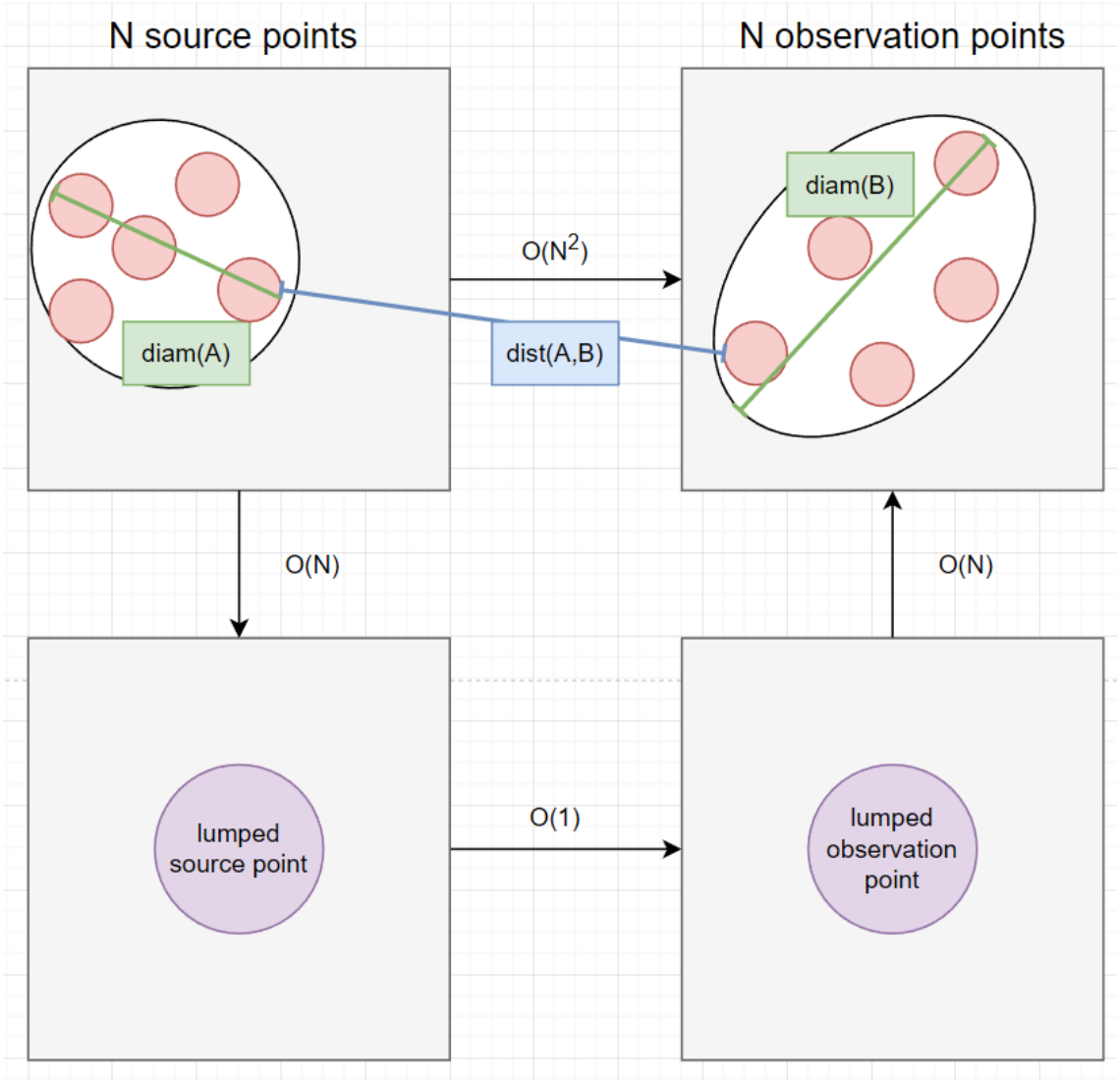

Figure 1.2: Schematic compute process of the FMM.

A downside of the FMM is that it is a physics-based technique. Generally they are more efficient than purely algebraic techniques (ex. hierarchical matrices), but they need to be formulated and implemented for a specific case. For example, a free-space FMM solver cannot be used for problems involving layered media. A multi-layered Green's function needs to be used. In addition, a high-frequency FMM solver cannot be used to solve DC or low frequency problems. This is an advantage that algebraic methods have, as only minor modifications need to be made before applying the solver to a wide range of problems.

Chapter 2

Theory

2.1 Problem Formulation for Electrostatic Problem

This thesis focuses on the electrostatic problem. Given an electrostatic potential $V(\mathbf{r})$, we wish to compute the surface charge distribution $\rho_s(\mathbf{r})$, that caused the potential. The capacitance can then be compared from the charge distribution. Each differential surface charge element can be treated individually. The electrostatic potential from a single point is

$$dV(r) = \frac{q}{4\pi\epsilon r} dr, \tag{2.1}$$

where dV is the differential voltage, q is the charge at the charge element, and r is the distance between the observation point of the potential and the charge element.

To compute the total charge, we integrate over the entire surface charge distribution ρ_s described by the surface $\mathbf{S}$ using differential area integrating element $d\vec{\mathbf{S}}'$. The position vector $\vec{\mathbf{r}}$ is the target point where the potential is measured. The primed position vector $\vec{\mathbf{r}}'$ is the source point on the surface charge distribution that contributes to the potential. The integral takes the familiar electrostatic form

$$V(\vec{\mathbf{r}}) = \int_{\mathbf{S}} \frac{\rho_s(\vec{\mathbf{r}}')}{4\pi\epsilon|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} d\vec{\mathbf{S}}'. \tag{2.2}$$

The analytical solution integrates over the continuously varying surface distribution. But for the purpose of computer computations, we need to discretize the surface. This means breaking the continuous distribution into smaller parts and taking the Riemann sum This is done by meshing the geometry of the problem to produce a 2D surface mesh.

The resulting mesh is composed of flat 2D polygons (usually triangles) that tile the original surface. The mesh should be fine enough that the surface charge density at any

point within the same mesh element (i.e. the same triangle) is almost uniform. In other words in an ideal setup, each mesh element has the same charge density throughout. The charge density only changes when moving to another mesh element. Note that the mesh element size does not need to be constant. In fact, it's better to have large mesh elements when the surface charge density is expected to not change much (ex. on flat surfaces) and to change to small mesh elements when the surface charge density is expected to vary quickly (ex. at edges and corners). This way we reduce the total number of mesh elements which speeds up computation, without sacrificing accuracy. We now represent the continuous surface charge in a discrete form

$$\rho_s(\vec{\mathbf{r}}) = \sum_{i=1}^{N} a_i f_i(\vec{\mathbf{r}}), \tag{2.3}$$

where $f_i(\vec{\mathbf{r}})$ is a set of basis functions, a_i are the coefficients for the corresponding basis function, and N is the total number of mesh elements. The domain of each basis function is a single discrete mesh element. Many types of basis functions can be used such as piece-wise triangular, piece-wise sinusoidal, or the commonly used Rao-Wilton-Glisson (RWG) basis function [11]. For the following analysis, the pulse function will be used. It is simple to understand and sufficient for this analysis. The pulse function "turns on" when it is evaluated over a specific mesh element. Otherwise, it is "off". In this way, the basis $f_i(\vec{\mathbf{r}})$ is selecting the coefficient a_i for the corresponding mesh element. Over all other surface elements, $f_i(\vec{\mathbf{r}})$ evaluates to 0. The basis function is written in piecewise form

$$f_i(\vec{\mathbf{r}}) = \begin{cases} 1, & \text{if } \vec{\mathbf{r}} \in \mathbf{S_i} \\ 0, & \text{otherwise,} \end{cases} \tag{2.4}$$

where $\mathbf{S_i}$ is the i-th mesh element. This way of expanding the problem into undetermined coefficients is known as the Method of Moments [1]. The solution to the problem will be encoded in the coefficients a_i. We can examine the units of the equation. We are solving for $\rho(\vec{\mathbf{r}})$ surface charge distribution which has units of charge per area. Since the basis function is unit-less and evaluates to 1 when on the particular mesh element, the units of the coefficients a_i must also be charge per area.

We put this expansion of the surface charge distribution back into the governing equation (2.2) to get

$$V(\vec{\mathbf{r}}) = \int_{\mathbf{S}} \frac{\sum_{i=1}^{N} a_i f_i(\vec{\mathbf{r}}')}{4\pi\epsilon|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} d\vec{\mathbf{S}}'. \tag{2.5}$$

At this point, we have one equation and N unknowns. To be able to solve for the coeffi-

cients a_i, we need at least N equations. We can produce the N equations by left-multiplying and integrating both sides of the equation by some testing functions. Mathematically, this is the same as computing an inner product between the charge density and a testing function. The common choice of testing functions are the basis functions that were previously used. If this choice is made, then this technique is called the Galerkin method of the Method of Galerkin. Each j-index will be a new equation.

$$\int_{\mathbf{S}} f_j(\vec{\mathbf{r}}) \cdot V(\vec{\mathbf{r}}) d\vec{\mathbf{S}} = \int_{\mathbf{S}} f_j(\vec{\mathbf{r}}) d\vec{\mathbf{S}} \cdot \int_{\mathbf{S}\in S} \frac{\sum_{i=1}^{N} a_i f_i(\vec{\mathbf{r}}')}{4\pi\epsilon|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} d\vec{\mathbf{S}}' \tag{2.6}$$

We can further manipulate this expression by taking summation out of the right-side integral, since it has no dependence on the integrating variable $d\vec{\mathbf{S}}'$.

$$\int_{\mathbf{S}} f_j(\vec{\mathbf{r}}) \cdot V(\vec{\mathbf{r}}) d\vec{\mathbf{S}} = \sum_{i=1}^{N} \int_{\mathbf{S}} f_j(\vec{\mathbf{r}}) d\vec{\mathbf{S}} \cdot \int_{\mathbf{S}\in S} \frac{a_i f_i(\vec{\mathbf{r}}')}{4\pi\epsilon|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} d\vec{\mathbf{S}}' \tag{2.7}$$

This has a matrix form of $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}}$ with matrix elements

$$b_j = \int_{\mathbf{S}} f_j(\vec{\mathbf{r}}) \cdot V(\vec{\mathbf{r}}) d\vec{\mathbf{S}}, \tag{2.8}$$

$$A_{ij} = \frac{1}{4\pi\epsilon} \int_{\mathbf{S}} f_j(\vec{\mathbf{r}}) d\vec{\mathbf{S}} \cdot \int_{\mathbf{S}\in S} \frac{f_i(\vec{\mathbf{r}}')}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} d\vec{\mathbf{S}}', \tag{2.9}$$

$$x_i = a_i. \tag{2.10}$$

We can simplify the matrix elements by using the definition of the basis function. The basis function we choose was the pulse function (2.4). The only time it evaluates to 1 is when the position vector $\vec{\mathbf{r}}$ is inside the domain of the i-th mesh element $\mathbf{S_i}$. Otherwise it will be 0. Looking at b_j, we observe that the only time the integrand is non-zero is when integrating over the j-th mesh element $\mathbf{S_j}$.

$$b_j = \int_{\mathbf{S_j}} V(\vec{\mathbf{r}}) d\vec{\mathbf{S}} \tag{2.11}$$

If the mesh elements are sufficiently small, then it is reasonable to assume that the potential $V(\vec{\mathbf{r}})$ is constant throughout the mesh element $\mathbf{S_j}$. The constant potential can be brought out of the integral. Let the constant potential in the j-th mesh element be V_j. Then the integral is only integrating a constant 1 over the domain of $\vec{\mathbf{S}} \in \mathbf{S_j}$. This is the area of the j-th mesh element. Let the area of the j-th mesh element be called T_j. The letter T is chosen because triangular mesh elements are very common. So T is for triangle.

$$b_j = V_j \int_{\mathbf{S_j}} 1 \cdot d\vec{\mathbf{S}} = V_j \cdot T_j \tag{2.12}$$

The same basis function simplification applies to A_{ij} as well, where $\mathbf{S_i}$ is the i-th mesh element.

$$A_{ij} = \frac{1}{4\pi\epsilon} \int_{\mathbf{S_j}} d\vec{\mathbf{S}} \cdot \int_{\mathbf{S_i}} \frac{1}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} d\vec{\mathbf{S}}' \tag{2.13}$$

The first integral in A_{ij} is integrating 1 over the domain of $d\vec{\mathbf{S}} \in \mathbf{S_j}$. This is the area of the j-th mesh element, which we previously called T_j

The second integral has a similar form to the first integral, but there is a distance term $|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|$ in the denominator. The target position vector $\vec{\mathbf{r}}$ comes from $V(\vec{\mathbf{r}})$ in the b_j expression, where $\vec{\mathbf{r}} \in \mathbf{S_j}$. Previously we assumed that the potential was constant across the same mesh element $\mathbf{S_j}$. So we can approximate the target position vector $\vec{\mathbf{r}}$ as pointing to the center of the j-th mesh element. Let this new approximated target position vector be $\vec{\mathbf{r_j}}$.

Likewise, the source position vector $\vec{\mathbf{r}}'$ comes from $\vec{\mathbf{r}}' \in S_i$. Let this be approximated as pointing to the center of the S_i mesh element. Call this approximated source position vector $\vec{\mathbf{r_i}}$.

If the mesh elements are small and well separated, then the integral can be approximated as the area of the i-th mesh element divided by the distance between the target $\vec{\mathbf{r}}$ and source $\vec{\mathbf{r}}'$ points. Using the above approximations for the position vectors, the denominator can be rewritten as $|\vec{\mathbf{r}} - \vec{\mathbf{r}}'| \approx |\vec{\mathbf{r_j}} - \vec{\mathbf{r_i}}|$.

$$A_{ij} \approx \frac{1}{4\pi\epsilon} \frac{1}{|\vec{\mathbf{r_j}} - \vec{\mathbf{r_i}}|} \cdot T_j \cdot T_i \tag{2.14}$$

With these simplifications, we can write out the explicit matrix equation $\mathbf{A}\vec{\mathbf{x}} = \vec{\mathbf{b}}$.

$$\begin{bmatrix} T_1 \cdot V_1 \\ T_2 \cdot V_2 \\ \vdots \\ T_N \cdot V_N \end{bmatrix} \approx \frac{1}{4\pi\epsilon} \begin{bmatrix} \frac{T_1 T_1}{|\vec{\mathbf{r_1}} - \vec{\mathbf{r_1}}|} & \frac{T_1 T_2}{|\vec{\mathbf{r_1}} - \vec{\mathbf{r_2}}|} & \cdots & \frac{T_1 T_N}{|\vec{\mathbf{r_1}} - \vec{\mathbf{r_N}}|} \\ \frac{T_2 T_2}{|\vec{\mathbf{r_2}} - \vec{\mathbf{r_2}}|} & \frac{T_2 T_2}{|\vec{\mathbf{r_2}} - \vec{\mathbf{r_2}}|} & \cdots & \frac{T_2 T_N}{|\vec{\mathbf{r_2}} - \vec{\mathbf{r_N}}|} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{T_N T_1}{|\vec{\mathbf{r_N}} - \vec{\mathbf{r_1}}|} & \frac{T_N T_2}{|\vec{\mathbf{r_N}} - \vec{\mathbf{r_2}}|} & \cdots & \frac{T_N T_N}{|\vec{\mathbf{r_N}} - \vec{\mathbf{r_N}}|} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \tag{2.15}$$

There are a few important observations to note from the $\mathbf{A}$ matrix:

1. The matrix is fully dense. All matrix entries are important and no entries can be ignored. This is unlike the sparse matrix systems that are produced from finite volume methods.

2. The matrix is symmetric. The matrix entry only depends on distance between points, so distance from point $|\mathbf{r}_i - \mathbf{r}_j|$ is the same as $|\mathbf{r}_j - \mathbf{r}_i|$.

3. The diagonal terms have a singularity because $|\mathbf{r}_i - \mathbf{r}_i| = 0$. Generally one must analytically integrate over these singularities using the full form of (2.6). The assumptions of the source and target points being well separated become worse when evaluating matrix terms close to the diagonal.

Looking at (2.13) that computes the matrix entry, each entry has a common inverse distance operation. This operation that is computed at all matrix entries is known as the kernel. In literature, this specific kernel is called the single-layer potential operator κ.

$$\kappa(\vec{\mathbf{r}}, \vec{\mathbf{r}}') = \frac{1}{4\pi|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} \tag{2.16}$$

It is the DC or zero frequency case of the 3-D Green's function kernel G

$$G(\vec{\mathbf{r}}, \vec{\mathbf{r}}') = \frac{e^{-jk|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|}}{4\pi|\vec{\mathbf{r}} - \vec{\mathbf{r}}'|} \tag{2.17}$$

where k is the wavenumber given as $k = 2\pi/\lambda$ and j is the unit imaginary number. At zero frequency, the wavenumber k tends to 0, which collapses the exponential in the numerator to 1. Thus, the single-layer potential operator is recovered.

The big drawback of the standard Method of Moments is the fully dense matrix. Let N be the total number of mesh elements. To solve the system directly using Gaussian elimination, it will take $O(N^3)$ operations to factorize the matrix and $O(N^2)$ to back-substitute to find the solution. This results in a total time complexity $O(N^3)$. To solve the system using iterative methods such as the conjugate gradient method, it takes $O(N^2)$ per iteration [10]. Matrix-vector products will take $O(N^2)$ operations. The matrix storage requirement scales with $O(N^2)$. These costly scaling relations prove too cumbersome for large-scale problems. As much as possible, researchers strive to reduce this to linear storage and linear time complexity.

2.2 Gaussian Quadrature Integration

Gaussian quadrature integration is a numerical integration technique that is used to compute the matrix entries $\mathbf{A}$, as described by (2.13). It approximates an integral by a weighted sum of function evaluations at non-uniformly spaced locations. For the case of this thesis work, integrals over 2-D triangles in 3-D space are computed. They are of the form

$$\int_{\mathbf{S_i}} f(\vec{\mathbf{r}}) d\vec{\mathbf{S}} \approx T_i \sum_{j=1}^{N_{quad}} w_j f(\alpha_j, \beta_j, \gamma_j), \tag{2.18}$$

where S_i is the i-th triangular mesh, $f(\vec{\mathbf{r}})$ is the surface integral function to be evaluated, N_{quad} is the number of Gaussian integration points, T_i area of the i-th triangle, w_j are the weights, and $(\alpha_j, \beta_j, \gamma_j)$ are the 3-D coordinates of the chosen quadrature points. The coordinates are given by Barycentric coordinates and the weights w_j are proportional to the triangle area. See Fig 2.1. The Barycentric coordinates are defined as

$$\alpha = \frac{A_1}{T};\ \beta = \frac{A_2}{T};\ \gamma = \frac{A_3}{T}, \tag{2.19}$$

where T is the area of the large triangle, and A_1, A_2, A_3 are the areas of the sub-divided triangles.

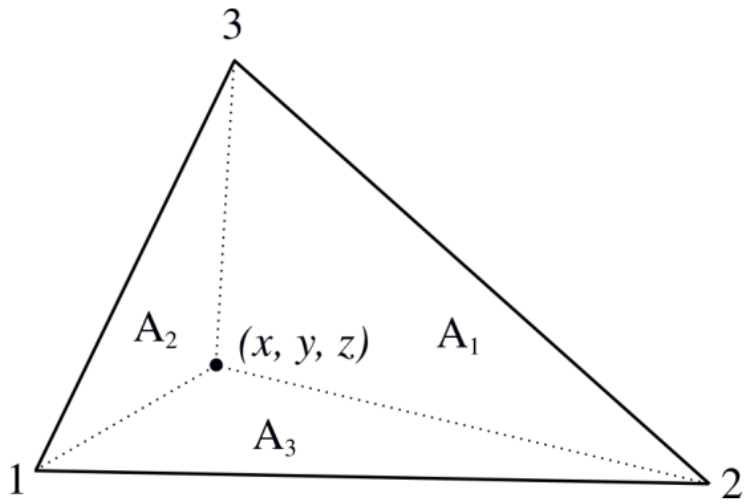

Figure 2.1: A point (x, y, z) is within triangular mesh element. The mesh element is split into three smaller triangles with areas A_1, A_2, A_3. Figure taken from Fig 2.6 in [12]

.

2.3 Low-Rank Approximations

A k-rank approximation of $\mathbf{A} \in \mathbb{R}^{mxn}$ is defined as

$$\mathbf{A} = \mathbf{BC}, \tag{2.20}$$

where $\mathbf{B} \in \mathbb{R}^{m \times k}$ and $\mathbf{C} \in \mathbb{R}^{k \times n}$. The explicitly expanded form is

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \vdots & \ddots \\ A_{m1} & A_{d2} & \dots & A_{mn} \end{bmatrix} \approx \begin{bmatrix} C_{11} & \dots & C_{1k} \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ C_{m1} & \dots & C_{mk} \end{bmatrix} \begin{bmatrix} C_{11} & \dots & \dots & \dots & C_{1n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{k1} & \dots & \dots & \dots & C_{kn} \end{bmatrix}, \tag{2.21}$$

where A_{ij}, B_{ij}, C_{ij} are the matrix entries for their respectively matrices. We see that the cost of storing $\mathbf{B}$ and $\mathbf{C}$ matrices is $O((m \times k) + (k \times n)) = O(k(m+n))$. The cost of storing the original $\mathbf{A}$ matrix is $O(m \times n)$.

If k is sufficiently small, then it consumes less memory to only store the $\mathbf{B}$ and $\mathbf{C}$ matrices. In practice, k will always be $k < n$ and $k < m$. This makes the first matrix $\mathbf{B}$ in the product a tall matrix since it has more rows than columns. And the second matrix $\mathbf{C}$ in the product will be wide since it has more columns than rows. This "tall" multiplied by "wide" form it referred to in this thesis as an outer product.

Since we wish to store the matrix from (2.15) in low-rank forms, the question is how to find these $\mathbf{B}$ and $\mathbf{C}$ matrices?

From the spectral theorem, we know for any arbitrary matrix $\mathbf{A} \in \mathbb{R}^{mxn}$ with $m \geq n$, there are orthogonal matrices $\mathbf{U} \in \mathbb{R}^{mxm}$ and $\mathbf{V} \in \mathbb{R}^{nxn}$ such that

$$\mathbf{A} = \mathbf{U_{SVD}}\, \mathbf{\Sigma}\, \mathbf{V_{SVD}}^T \text{ with } \mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix}, \tag{2.22}$$

where $\mathbf{U_{SVD}} \in \mathbb{R}^{mxm}$, $\mathbf{\Sigma} \in \mathbb{R}^{mxn}$, and $\mathbf{V_{SVD}} \in \mathbb{R}^{nxn}$. The $\mathbf{U_{SVD}}$ and $\mathbf{V_{SVD}}$ matrices are orthogonal as they are constructed from a linearly independent set of column vectors. The columns of $\mathbf{U_{SVD}}$ are $\mathbf{u_{SVD,1}}, \cdots, \mathbf{u_{SVD,m}}$ are called the left singular vectors of $\mathbf{A}$. The columns of $\mathbf{V_{SVD}}$ are $\mathbf{v_{SVD,1}}, \cdots, \mathbf{v_{SVD,n}}$ are called the right singular vectors of $\mathbf{A}$. The $\mathbf{\Sigma}$ matrix is a diagonal matrix with the diagonal entries typically ordered as $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$. The σ_i are called the singular values of $\mathbf{A}$. Corresponding, this decomposition is called the singular value decomposition of $\mathbf{A}$ (SVD). The SVD can be thought of as a

generalized eigenvalue decomposition that works for any sized matrix instead of only square matrices. Note that the spectral theorem is actually a stronger theorem and this SVD is a special case of the spectral theorem.

Knowing that an SVD will always exist, we can conveniently use the Eckart-Young theorem. It states that the "best" rank-k approximation of $\mathbf{A} \in \mathbb{R}^{mxn}$ will be the truncated SVD. The truncated SVD computed as the same form as the SVD, but only taking the first σ_k singular values and zeroing all subsequent singular values. After doing the matrix multiplication, this also has the effect of zeroing the corresponding left and right singular vectors. This has the form

$$\mathbf{A} = \mathbf{U_{SVD}}\, \mathbf{\Sigma}\, \mathbf{V_{SVD}}^T \text{ with } \mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \\ & \mathbf{0} & \end{bmatrix}, \tag{2.23}$$

where $\mathbf{A_k}$ is the rank-k truncated SVD approximation of $\mathbf{A}$.

The "best" approximation is defined as the one that minimizes the norm of the difference. It's been widely proved for many definitions of norms. But the one that is perhaps most intuitive is the Frobenius norm defined as

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |a_{ij}|^2}, \tag{2.24}$$

where a_{ij} are the matrix entries of $\mathbf{A}$. Minimizing the Frobenius norm can be thought of as minimizing the root mean square error (RMSE) between two matrices. Thus the best possible low-rank approximation of matrix $\mathbf{A}$ is the truncated SVD approximation $\mathbf{A_k}$ of the largest k value such that it satisfies a user specified error threshold

$$||\mathbf{R}||_F = ||\mathbf{A} - \mathbf{A_k}||_F \leq \epsilon ||\mathbf{A}||_F, \tag{2.25}$$

where $\mathbf{R}$ is the error matrix and ϵ is the truncation error threshold. Typically an error threshold of 10^{-3} is used. It's observed that such a threshold is provides method of moments results within 1% RMSE of theoretical results.

Unfortunately, the time complexity to compute the full SVD is $(m^2n + n^3)$. The time complexity to compute the truncated SVD is $(m \times n^2)$. Since the method of moments formulation produces a square matrix of size $N \times N$ where N is the number of mesh elements, it can be said that the time complexity to compute a rank-k SVD approximation $\mathbf{A_k}$ is $O(N^3)$. This is too costly for to scale to large problems. The research effort is thus to find ways to

approximate $\mathbf{A_k}$ with improved time complexity. There are many ways to do this such as tensor product interpolation and Taylor series expansion of the kernel [13].

2.4 Adaptive Cross Approximation

One of the more popular ways to do this approximation is called the Adaptive Cross Approximation (ACA). It was first presented by Bebendorf in [14] and first applied to electromagnetic problems by Zhao [6]. It's a way to construct a low-rank approximation of a matrix $\mathbf{A}$ from well-chosen rows and columns of $\mathbf{A}$. It is written as

$$\tilde{\mathbf{A}} = \mathbf{U}\,\mathbf{V} = \sum_{i=i}^{k} \mathbf{u_i} \mathbf{v_i}^T, \tag{2.26}$$

where $\tilde{\mathbf{A}} \in \mathbb{R}^{m\times n}$ is the low-rank approximation of $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{U} \in \mathbb{R}^{m\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times n}$ are the rank-k matrices, and $\mathbf{u_i} \in \mathbb{R}^{m\times 1}$ and $\mathbf{v_i} \in \mathbb{R}^{1\times n}$ are the row and column vectors of $\mathbf{U}$ and $\mathbf{V}$ respectively. These row and column vectors are selected from the original $\mathbf{A}$

ACA is an iterative process. One begins with only picking one row and one column from the $\mathbf{A}$ matrix and seeing if the outer product of the rows and column vectors can sufficiently minimize the error to within the user set threshold. If the error is too large, then select another set of row and column vectors from $\mathbf{A}$. The algorithm succeeds when convergence is reached

$$||\mathbf{R}||_F = ||\mathbf{A} - \tilde{\mathbf{A}}||_F \leq \epsilon ||\mathbf{A}||_F, \tag{2.27}$$

where $\mathbf{R}$ is the error matrix and ϵ is the error threshold.

This is called a "cross" approximation because the selected columns and rows interpolate the original $\mathbf{A}$ matrix entries exactly where they cross or overlap. See Fig 2.2.

A critical part of the algorithm is picking set row and column vectors to add to the approximation. MATLAB notation will be used for the following section. Let $\mathbf{I} = [I_1 \cdots I_k]$ and $\mathbf{J} = [J_1 \cdots J_k]$ be the arrays containing a k number of orderly selected row and column indices from $\mathbf{A}$. In other words, $\mathbf{A}(\mathbf{I}, \mathbf{J})$ is an $k \times k$ sub-matrix of $\mathbf{A}$. In Fig 2.2, this would be if the dark blue entries were merged into one matrix. It was shown in [3] that a quasi-optimal row and column selection is to choose the indices such that $\det |\mathbf{A}(\mathbf{I}, \mathbf{J})|$ is maximal. Unfortunately this is still a computationally difficult problem being NP-hard. So optimal selection of this is not possible.

Instead, greedy algorithms are used to iteratively construct the sub-matrix $\mathbf{A}(\mathbf{I}, \mathbf{J})$ where the $\det |\mathbf{A}(\mathbf{I}, \mathbf{J})|$ is maximized at each step. In Algorithm 1, the approximation starts with

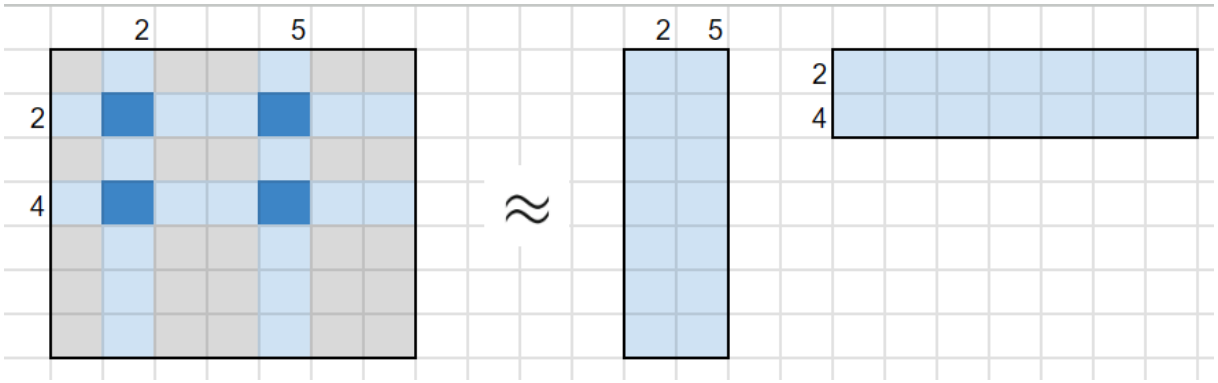

Figure 2.2: The left 7x7 matrix ix approximated by the outer product of a 7x2 and 2x7 matrix. The approximation is rank 2. The dark blue entries are "crossed" in the outer product and thus exactly recovered in the approximation.

an empty matrix so the error matrix $\mathbf{R}$ is the entire matrix $\mathbf{A}$. Then with each iteration, more row and column vectors are added until the error threshold is reached.

Algorithm 1 ACA with full pivoting.

1: Set $R_0 := \mathbf{A}$, $\mathbf{I} := , \mathbf{I} := , k := 0$
2: **repeat**
3: $k = k + 1$
4: $(i^*, j^*) := \text{argmax}_{ij} |\mathbf{R_{k-1}}|(i, j)$ ▷ find max value for full pivoting
5: $\mathbf{I} = \mathbf{I} \cup i^*$, $\mathbf{J} = \mathbf{J} \cup j^*$ ▷ update row and column index set
6: $\delta_k = \mathbf{R_{k-1}}(i^*, j^*)$ ▷ get the pivot value
7: $\mathbf{u_k} = \mathbf{R_{k-1}}(:, j^*)$, $\mathbf{v_k} = \mathbf{R_{k-1}}(i^*, :)/\delta_k$ ▷ slice into the matrix to get the row and column vectors, but divide the column vectors by the pivot
8: $\mathbf{R_k} = \mathbf{R_{k-1}} - \mathbf{u_k}\mathbf{v_k}^T$ ▷ update error matrix
9: **until** $||\mathbf{R_k}||_F \leq \epsilon ||\mathbf{A}||_F$ ▷ error threshold is reached

The algorithm as given still requires the $\mathbf{A}$ matrix to be computed fully before ACA can start. But there are improved versions of the algorithm where complete knowledge of $\mathbf{A}$ is not required, such as in Gibson [5]. It computes rows and columns on the fly. See section 3.1 for more details about the particular ACA used.

An additional step that is possible after ACA is to further recompress the low-rank approximation by the SVD. The rows and columns of $\mathbf{U}$ and $\mathbf{V}$ are almost orthogonal, but generally not exactly orthogonal. They can be made exactly orthogonal by computing the QR and SVD decomposition, then assembling the exactly orthogonal versions of $\mathbf{U}$ and $\mathbf{V}$ using those factorizations. The idea is that the additional cost of this step is paid off during the direct LU factorization in the solve step, as the low-rank matrices will be smaller and thus matrix operations require fewer steps. This additional step is not done in this thesis work.

2.5 Hierarchical Matrices

Now that we have sufficient background, we can finally discuss hierarchical matrices. They were first introduced by Wolfgang Hackbusch in 1999 [15]. It builds on the work of Stefan Sauter who proposed a panel-clustering method [16].

Hierarchical matrices are a data structure that allows a dense matrix to be represented in both data-dense and data-sparse sub-matrices. We've seen from both the FMM and AIM techniques that the far-field interactions do not need to be represented on the same level of detail as the near-field interactions. Thus all three of these MoM acceleration methods can be thought as different ways to accelerate the far-field regions. The FMM used a lumping technique of source and observation points to reduce the time complexity of the far-field region. The AIM used clever FFTs and convolutions to speed-up far-field matrix operations. The insight that hierarchical matrices add is to approximate the far-field region using purely algebraic techniques. This allows the acceleration to be kernel independent, thus making the formulation far more robust. It can be applied without major modification to a wide range of problems such as DC, high frequency, free space, and layered media.

Hierarchical matrices seek to use low-rank approximations to compress the far-field regions of the dense A matrix. The particular method studied in this thesis is the ACA low-rank approximation. It's noted that there other exists other methods to low-rank approximations such as tensor product interpolation [17] which uses Lagrange polynomials and hybrid cross approximation [13] which combines both the tensor product interpolation and ACA.

A key concept in hierarchical matrices is to know which blocks of the $\mathbf{A}$ matrix to approximate and which blocks to leave alone. This falls back to the same admissibility condition that was used in the FMM. Refer to (1.5). If a cluster is determined to be admissible, then it will be stored as in low-rank approximation. If a cluster is not admissible, it will be stored in a fully dense form. The admissibility condition works when the mesh structure is already grouped into clusters. This can be achieved by running a clustering algorithm like k-means clustering or cobblestone clustering (referred to as cobblestone spatial grouping in [4]) before the MoM formulation.

The clustering method that is used in this thesis is an recursive method called geometric bisection. The starting point of the technique is to group all the mesh elements as one cluster. Then the longest axis is determined. This is the axis that connects the two most spatially separated mesh elements. The cluster is split at some plane along this axis into two clusters, such that each sub-cluster has an equal number of mesh elements. This process is continued until the smallest cluster, referred to as leaf clusters, have reached some minimum

number of mesh elements.

A visualization of four levels of the geometric bisection clustering method is shown in Fig 2.3 and the corresponding mock-up of the resulting hierarchical matrix is shown in Fig 2.4.

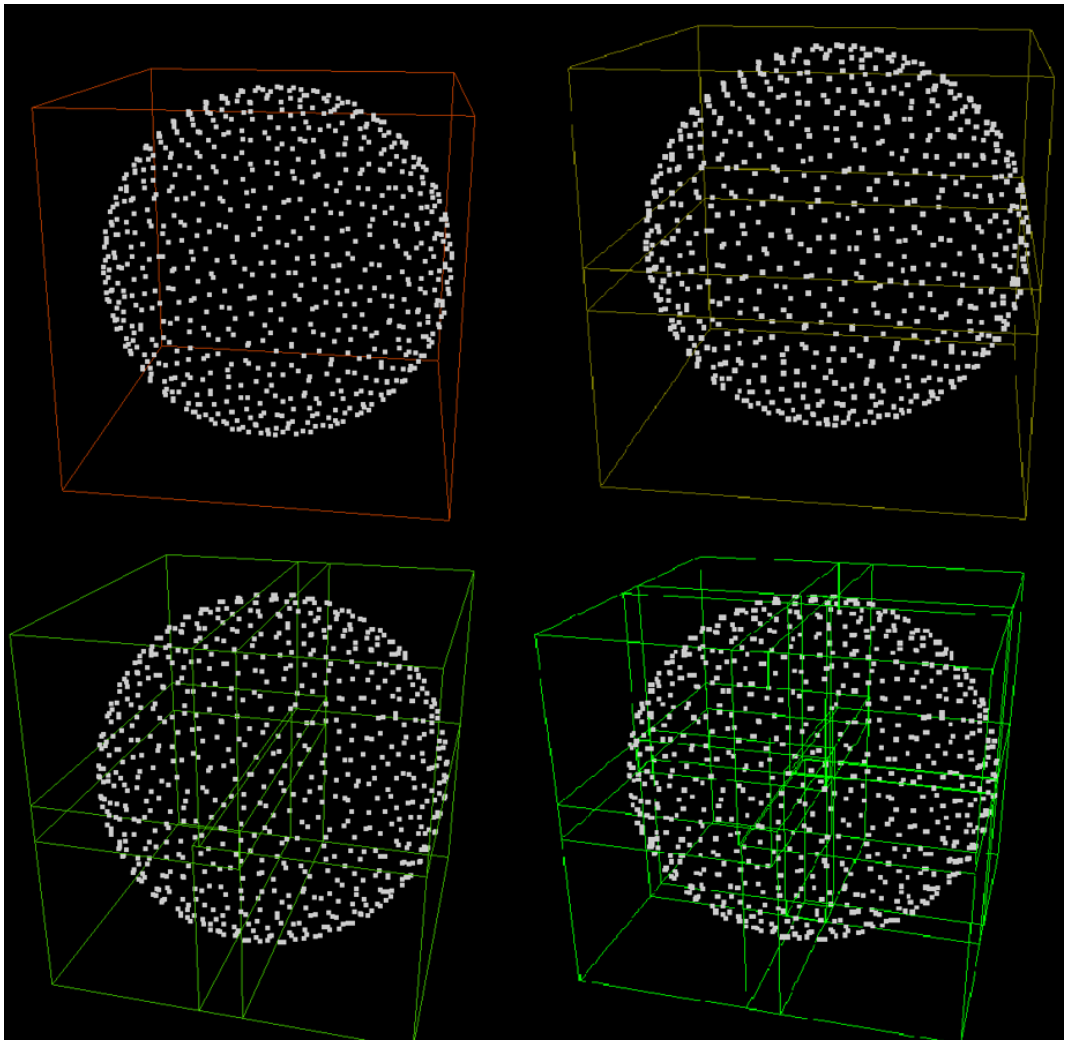

Figure 2.3: Given a spherical mesh, the various levels of refinement are shown. Top left is base level. Top right is level 1, with two blocks. Bottom left is level 2, which four blocks. Bottom right is level 3, with eight blocks.

In Fig 2.4, let each numbered cell represent exactly 1/8 of the spherical mesh. The x, y, and z axis are each bisected, producing eight leaf clusters. In the first level of geometric bisection, the entire spherical mesh is treated as one cluster. When applying the admissibility condition, there is no second cluster B so the $dist(A, B)$ is infinite, which means the cluster is admissible. Thus the entire matrix is colored green.

Then after applying once bisection, the eight clusters are split into two hemispheres each containing four clusters. The interactions within the same hemispheres are do not pass the

admissibility condition and thus are colored red. But interactions across hemispheres are admissible and thus colored green. This leads to two 4x4 cell of admissible blocks and two 4x4 cells of inadmissible blocks. The process is continued until the sphere is divided into eight quadrant clusters. Refinement can continue beyond this point, but it was stopped for illustrative clarity.

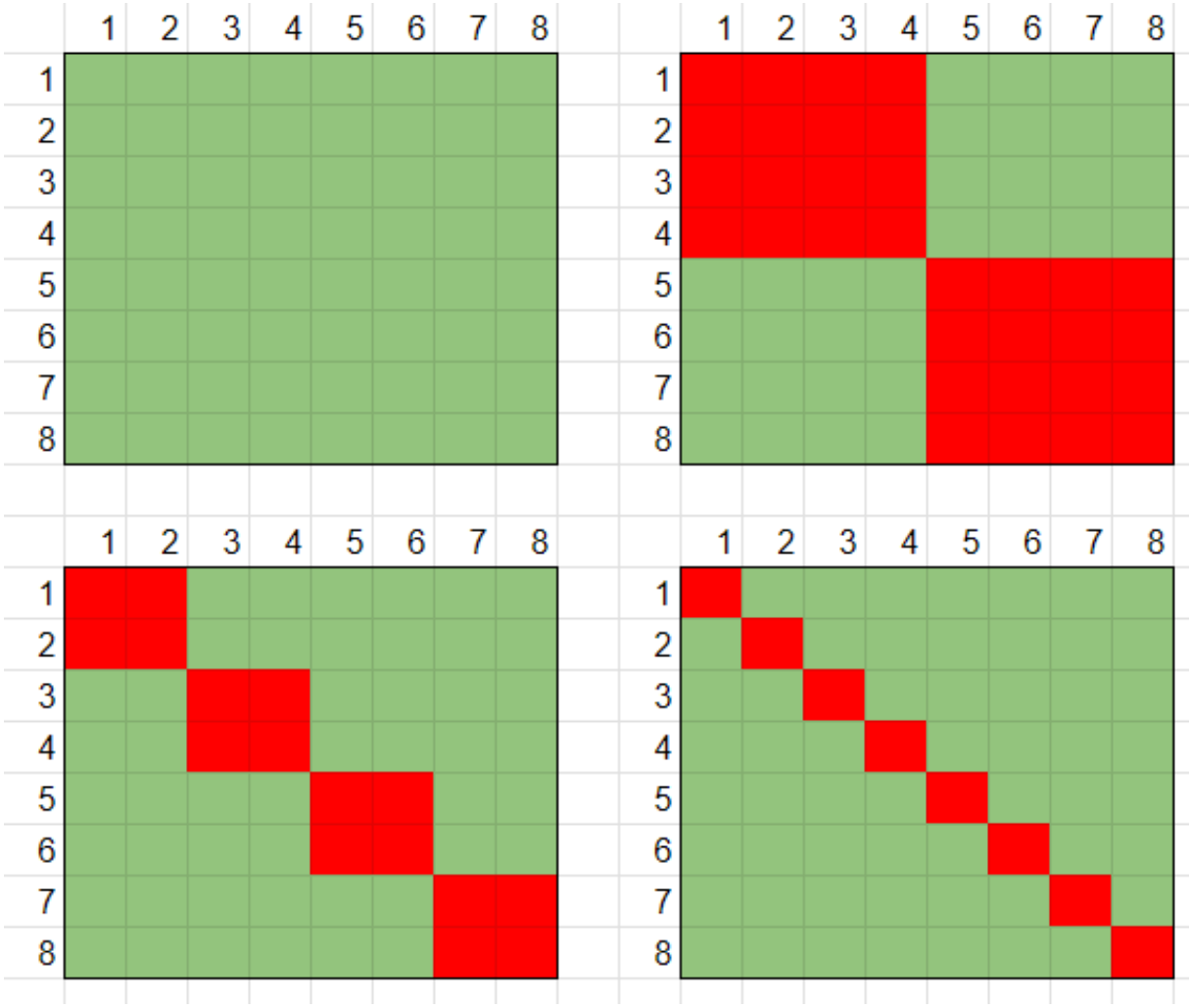

Figure 2.4: Graphical representation of admissible (green) and in-admissible (red) blocks of an hierarchical matrix. Increasing levels of refinement moving left-to-right, then top-down.

The structure of the hierarchical matrix is tracked using recursive index sets. An index set is an array of indices, that are partitioned into groupings that represent the clusters. The index sets select the rows and columns of $mathbf{A} \in \mathbb{R}^{m \times n}$ that are clustered. Let $\mathbf{I} = [I_1 \cdots I_m]$ and $\mathbf{J} = [J_1 \cdots J_n]$ be the arrays containing m and n number of orderly selected row and column indices from $\mathbf{A}$. In other words using MATLAB notation, $\mathbf{A}(\mathbf{I}, \mathbf{J})$ is slicing into matrix $\mathbf{A}$ to pull out the rows and columns required to form a sub-matrix using the

index sets. Due to **I** selecting rows and **J** selecting columns, they are referred to as the row and column clusters according. See figure 2.5.

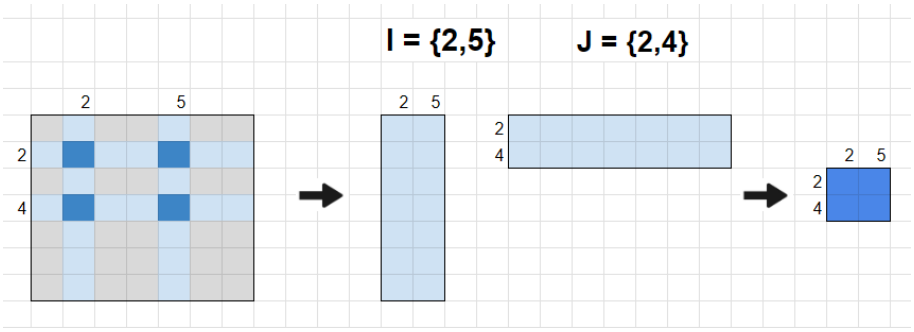

Figure 2.5: The row and column index sets overlap to select the matrix elements (dark blue).

At each level of the hierarchical matrix, the row and column clusters are constructed. So the index sets needs to be tracked at all levels. This is done with a tree structure called a cluster tree. An additional note is that all indices of the **A** must be used as all blocks must be assigned as admissible or inadmissible. In practice this means that at all levels of the cluster tree, the index sets contain all indices from 1 to m or n depending on if it is a row or column cluster respectively. The only thing that changes is how the indices are partitioned.

For the spherical example in Fig 2.3 and Fig 2.4, the cluster tree would look like Fig 2.6. For the final hierarchical matrix construction, there is one recursive cluster tree representing the rows and another recursive cluster tree representing the columns. Together, they are referred to as the cluster block of a hierarchical matrix.

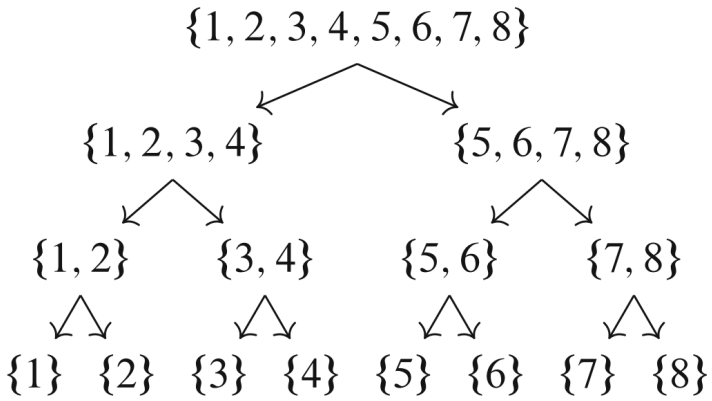

Figure 2.6: An example of a cluster tree showing the row and column clusters representing index sets, for the spherical refinement in Fig 2.3. Note that both the row and column cluster trees would look like this. Figure taken from Fig 6 of [3]

.

Chapter 3

Methods

3.1 Implementing Hierarchical Matrices

An existing C library for hierarchical matrices called H2Lib was used. It is available from: `https://github.com/H2Lib/H2Lib`h2lib.

The library was developed by Steffen Borm and his Scientific Computing Group at the University of Kiel [18]. Borm is a student of Hackbusch, who is the original inventor of the hierarchical matrix. The library provides data structures for storing hierarchical matrices, support for common matrix operations, and parsers for standard mesh file formats. This library was selected because it is a fully-featured hierarchical matrix library with detailed documentation.

No modifications to the library functions were required. Out-of-the-box it supported pulse function basis and the single layer potential kernel operator, which is needed for the electrostatic formulation. The library functions were used to construct an electrostatic problem of arbitrary mesh and applied voltages.

The key parts of the code are shown below.

```
int main(int argc, char **argv) {
    ...
    // 1. load mesh into geometry
    psurface3d mesh = read_gmsh_surface3d(filename);

    // 2. prepare kernel operator
    uint q_reg = 2; // number of integration points for regular intervals
    uint q_sing = q_reg + 2;    // number of points for singular intervals
    basisfunctionbem3d row_basis = BASIS_CONSTANT_BEM3D;
    basisfunctionbem3d col_basis = BASIS_CONSTANT_BEM3D;
    pbem3d bem_slp = new_slp_laplace_bem3d(mesh, q_reg, q_sing, row_basis,
```

```
    col_basis);

    // 3. construct cluster tree from geometry
    uint m = 4;     // number of interpolation points
    uint clf =  clf_user;   // minimum leaf cluster size
    pcluster root = build_bem3d_cluster(bem_slp, clf, row_basis);

    // 4. construct block tree from cluster tree
    real eta = eta_user; // admissibilty parameter eta
    pblock broot = build_nonstrict_block(root, root, &eta,
    admissible_2_cluster);

    // 5. construct skeleton hierarchical matrix from block tree
    setup_hmatrix_aprx_aca_bem3d(bem_slp, root, root, broot, accur);

    // 6. assemble (also called fill-in) matrix values
    phmatrix V = build_from_block_hmatrix(broot, m*m*m);

    // 7. construct right-hand side vector
    pavector b_constant = new_avector(mesh->triangles);
    for (int i = 0; i < b_constant->dim; i++) {
        // set all points to 1V
        b_constant->v[i] = mesh->g[i]/2 *epsilon_0 * 1;
    }

    // 8. solve the system with LU-decomposition and back-substition
    pctruncmode tm = new_releucl_truncmode();
    lrdecomp_hmatrix(V, tm, eps_solve);

    // solution vector overwrites the right-hand side vector
    lrsolve_hmatrix_avector(false, V, b_constant);

    ...
}
```

The ACA algorithm from the H2Lib library was used. There is are full pivoting and partial pivoting versions, but focus was spent on the full pivoting version. No systematic failures or edge cases as described by Gibson were encountered, so no modifications to the library code was needed. The accuracy threshold for ACA approximation used throughout all experiment runs was 10^{-3}. This threshold was used by Gibson [5] and Zhao [6]. When verifying the correctness of the code with a PEC sphere, it was found that this error threshold provided results within 1% RMSE of theoretical results.

3.2 Meshing

Surface meshing of geometries was done using Gmsh [19] at varying levels of mesh refinement. The structures were created with Gmsh's scripting language. A control parameter called characteristic length controls the size of the mesh. A smaller characteristic length produces smaller individual mesh elements, resulting in a denser mesh.

The mesh file was saved in an older `.msh` format in order to be compatible with the integrated H2Lib mesh parser.

An example Gmsh script is given below. It produces a spherical mesh with 30 division along the circumference is given below. This is done by setting the characteristic length to 1/30 of the circumference.

```
SetFactory("OpenCASCADE");

scale = 1.0;

r1 = 0.5*scale;

x1 = 0.0;
y1 = 0.0;
z1 = 0.0;

Sphere(1) = {x1, y1, z1, r1, -Pi/2, Pi/2, 2*Pi};

Physical Surface("Object0") = {1};

lc = 2*Pi*r1/30;
Characteristic Length {1:10000} = lc;

Mesh.Algorithm = 6;
Mesh 2;
```

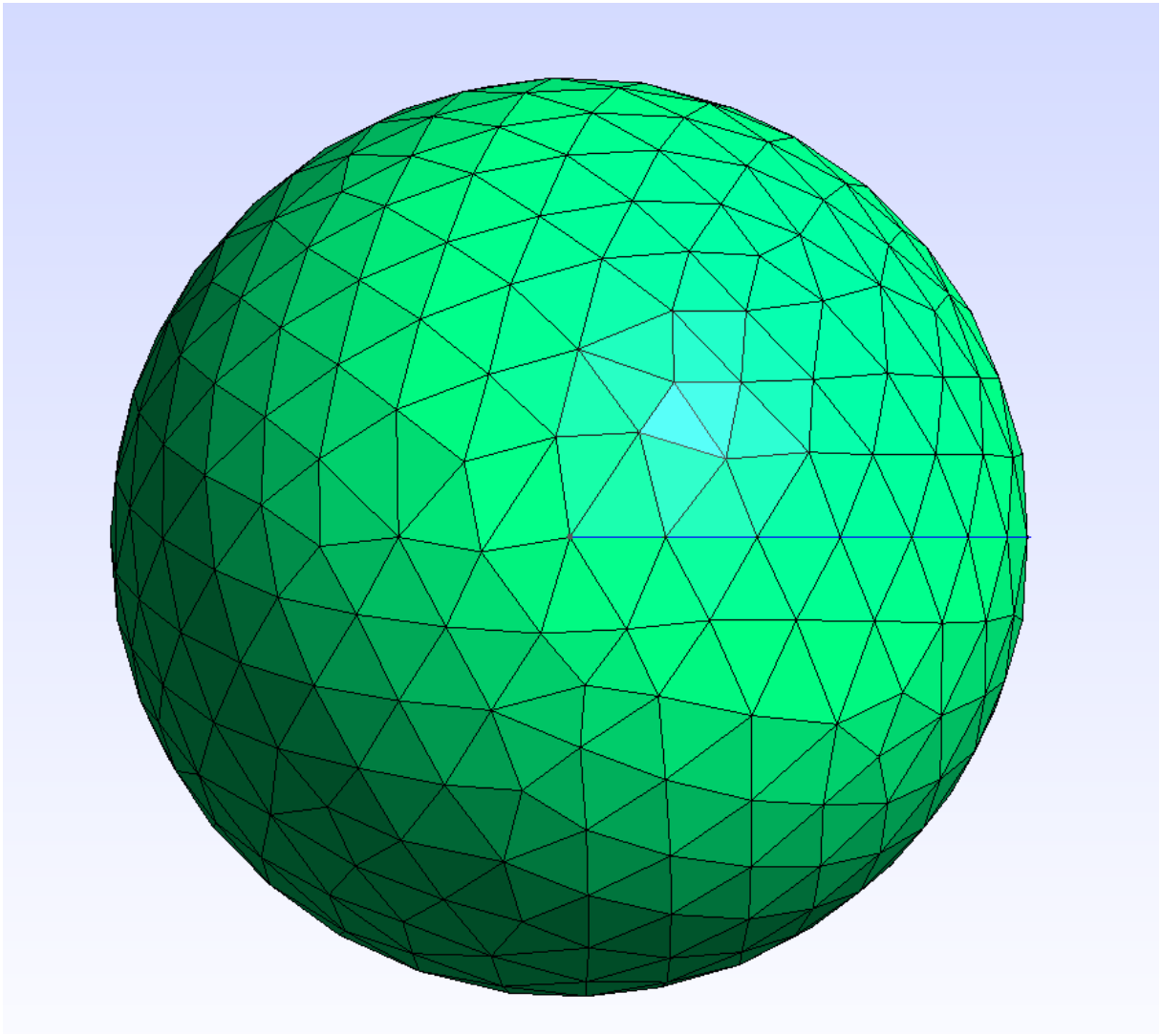

Figure 3.1: An example spherical mesh produced by Gmsh, with 30 divisions along the circumference.

3.3 Test Methodology

All experimental runs were completed on an up-to-date Ubuntu 22.04 distribution running within Windows Subsystem for Linux, on a Windows 10 Server 2022 version of Microsoft Windows. The test machine was equipped with Intel Core i7-9700K CPU and 32 GB of RAM. The processor has 8 cores and 8 threads, with a base frequency of 3.6 GHz and a maximum single-core boost frequency of 4.9 GHz. The RAM is DDR4 3200MB/s in a 2x16GB arrangement.

The C test program was compiled using without any GCC `-o` optimization flags and with debugging enabled. The GCC optimizations were avoided to aid debugging, as optimizations may introduce unexpected behavior between the source code and compiled executable. Other optimizations were used. The BLAS/LAPACK libraries were used to accelerate linear algebra operations such as matrix-vector product and LU decomposition. OpenMP was used to parallelize the matrix entry fill-in step.

After the C test program was compiled, a Python wrapper script was used to automatically run results with different parameters and mesh structures. The results were logged as text files, which were later analyzed and plotted.

Attempts were made to limit the effect of confounding variables. These were background processes and disk thrashing.

Regarding background processes, firstly the test machine is a shared server. Other users of the machine were notified when experiment runs were taking place and asked to avoid using the machine. In addition, the test runs were scheduled overnight during off-hours to avoid the chance of someone forgetting about the warning and using the machine. Looking at the Windows server connection logs, the test account was the only account active during all experiment runs. Secondly during the experimental run, Windows search indexing and updates were disabled. Background processes were also monitored in both Windows task manager and Ubuntu system monitor, to ensure no unexpected heavy background tasks were competing for compute resources during the run.

Disk thrashing is when the system runs out of RAM and must load/store matrix elements on the hard-drive. It significantly slows down program execution since the CPU has to go to the hard-drive for memory access, which is much slower than RAM. This would have artificially added solve time to the runs. The Ubuntu 22.04 virtual machine was configured to access the 30 GB of RAM, leaving 2 GB for Windows overhead. The test runs were tuned to keep the memory usage well under 30 GB. The largest consumer of RAM in the C test program was the hierarchical matrix structure. The largest memory usage of this data structure during all experiment runs was 20.2 GB. This is well below the 30 GB RAM limit

with enough margin for the Ubuntu OS and other processes. Thus disk thrashing is unlikely to have occurred.

Unless otherwise specified, the ACA error threshold ϵ in (2.27) used was 10^{-3}. It was observed that this error threshold allowed the PEC sphere to match the theoretical surface charge distribution within 1% RMSE. This was the value used by Gibson [5] and Zhao [6]. It is noted that Zhao [7] used an error threshold of 10^{-2} to 10^{-6} and observed relative error below $10^{(}-5)$ throughout.

Chapter 4

Results and Discussion

4.1 Verification of Correctness

4.1.1 Sphere

To verify the correctness of the code, a perfect electric conductor (PEC) spherical shell was simulated. The sphere was chosen because its capacitance is known to have an analytic solution. The capacitance of two concentric spheres in free space is

$$C = \frac{4\pi\epsilon_0}{\left(\frac{1}{a} - \frac{1}{b}\right)}, \tag{4.1}$$

where a is the inner radius and b is the outer radius. With an isolated sphere in free space, the capacitance is obtained by setting $b \to \infty$. The capacitance is then

$$C = 4\pi\epsilon_0 R, \tag{4.2}$$

where $R = a$ is the radius of the sphere. See Fig 4.1

The potential of spherical shell was set to a uniform 1 V at all points. Then the total charge Q can be calculated as

$$Q = C/V = C/1 = 4\pi\epsilon_0 R. \tag{4.3}$$

Since the potential is uniform across the sphere, the charge distribution is also uniform. So dividing by the surface area of $4\pi R^2$, we obtain that the theoretical surface charge distribution is

$$\rho_s = \frac{Q}{\text{surface area}} = \frac{4\pi\epsilon_0 R}{4\pi R^2} = \epsilon_0/R, \tag{4.4}$$

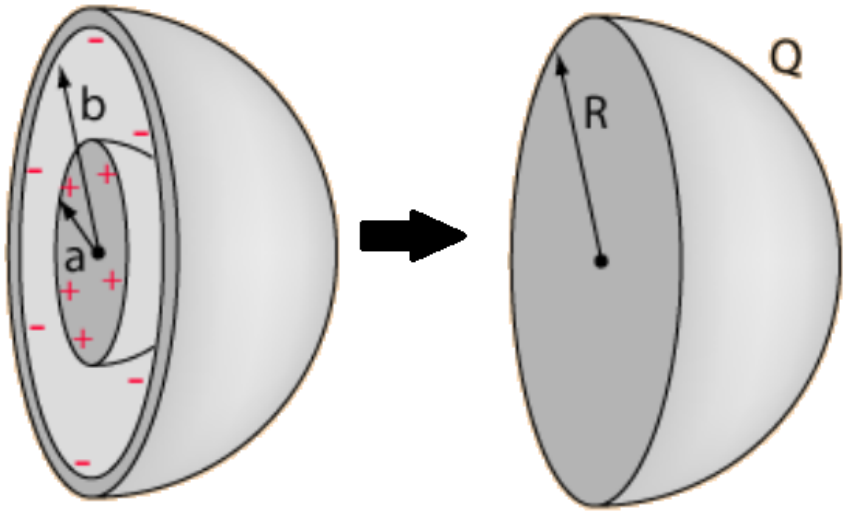

Figure 4.1: The capacitance of an isolated spherical shell in free space is obtained by taking the outer radius $b \to \infty$

where ρ_s is the surface charge density. In the spherical mesh used, the radius R was set to 0.5 m, so the theoretical surface charge density is $\rho_s = 17.7\, pC/m^2$.

The normalized RMSE was used as a figure of merit. It is computed as

$$\text{RMSE}_{\text{normalized}} = \sqrt{\frac{1}{N}\sum_{i}^{N}\left(\frac{\rho_{s,\text{theory}} - \rho_{s,i}}{\rho_{s,\text{theory}}}\right)^2}, \tag{4.5}$$

where N is the total number of mesh elements, $\rho_{s,\text{theory}}$ is the theoretical surface charge density of $17.7 pC/m^2$, and $\rho_{s,i}$ is the computed surface charge density at the i-th mesh element.

We see that in Fig 4.2, the normalized RMSE drops to below 1% when more than 1200 mesh elements are used.

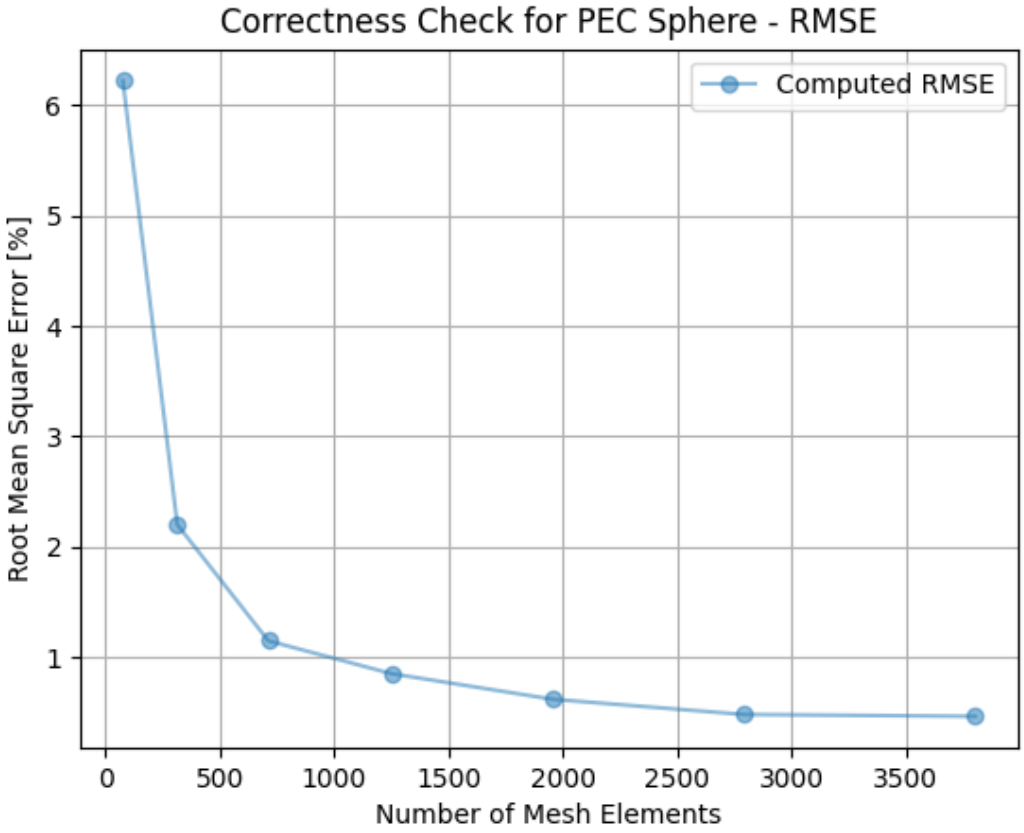

Figure 4.2: Normalized RMSE of the computed surface charge density for a PEC sphere.

In Fig 4.3, we see some patchy areas where the computed surface charge density for each triangular mesh element is not exactly the same.

In Fig 4.4, we see that the computed surface charge density is not uniform for all mesh triangles. The maximum and minimum charge densities are plotted. Using finer meshes beyond 1000 mesh elements doesn't seem to improve the margin by which the minimum and maximum charge densities approach the theoretical limit. This is likely due to improperly tuning a parameter such the admissibility parameter η or the minimum leaf cluster size.

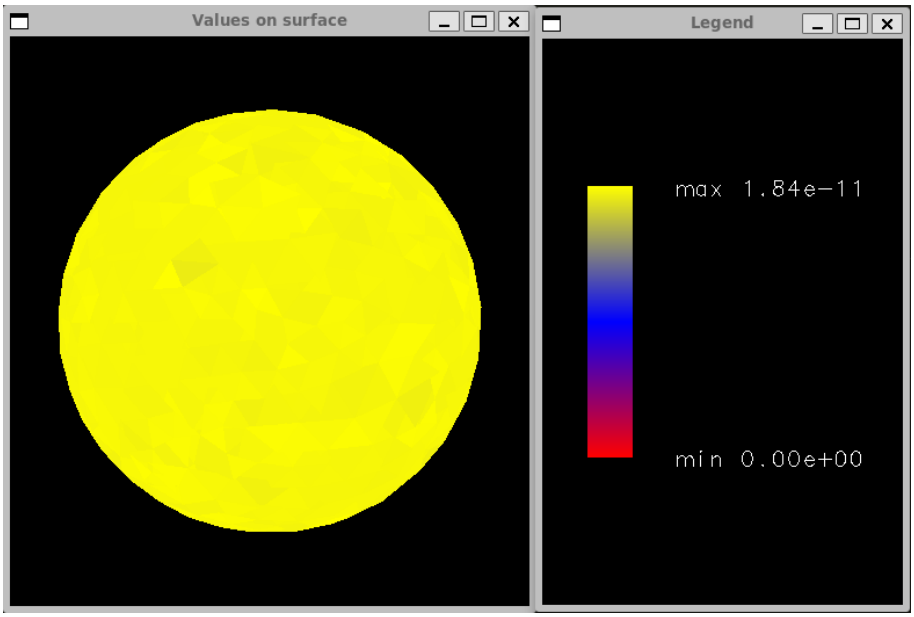

Figure 4.3: Visualization of the computed surface charge density using 714 triangular mesh elements

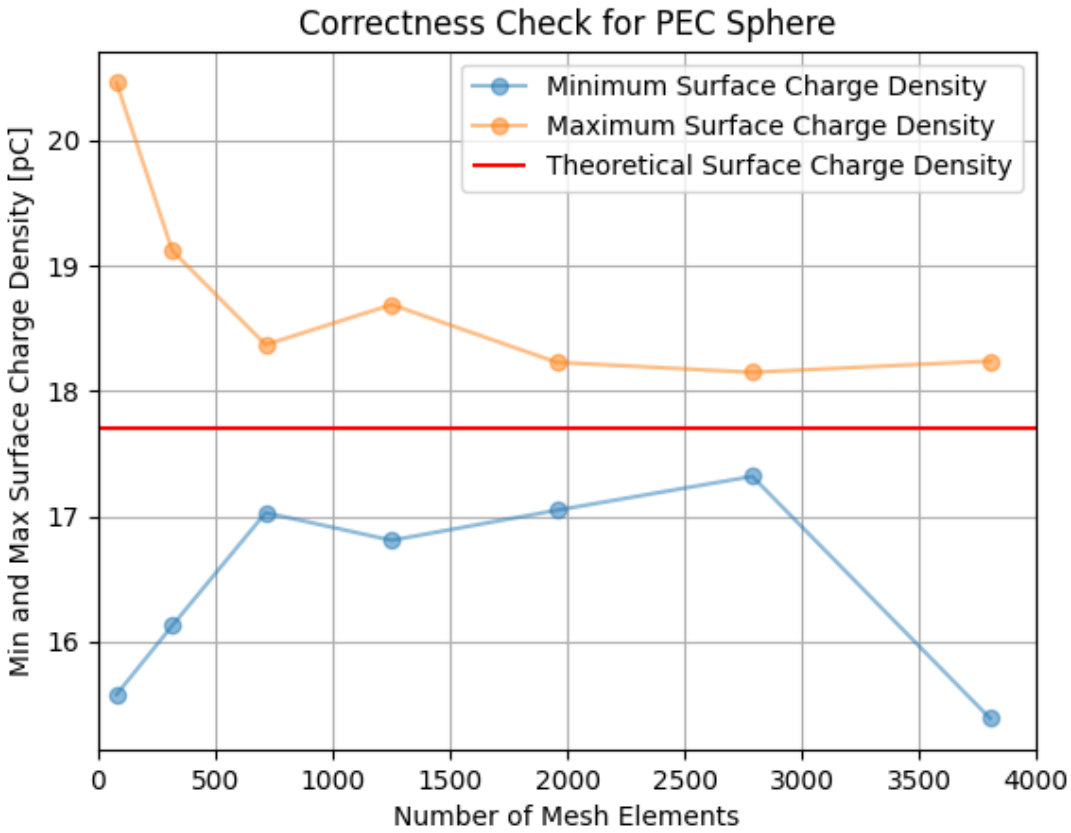

Figure 4.4: The maximum and minimum surface charge densities across all mesh elements.

4.1.2 Other Geometries

The DC charge distribution of other structures are examined. Since there is no closed form analytic solution to these shapes, only the qualitative aspects of the solution will be discussed.

In Fig 4.5, both spheres are set to the same potential of 1 V. Since they have the same potential, there should be the same amount of positive charge on both spheres. But the positive charge repel each other where the spheres are closest. This leads to a local region where the surface charge density is slightly lower. And conversely, the positive charges will accumulate the furthest away from the other sphere. It is hard to see, but there is a slight local increase in surface charge density is slightly higher on the furthest ends of the spheres.

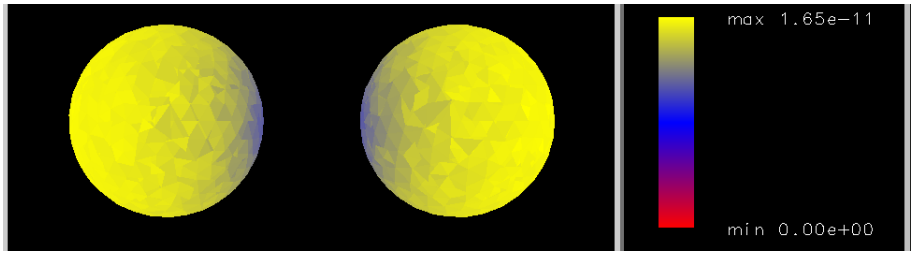

Figure 4.5: Two spheres separated by a center to center distance of three radii. Left sphere set to 1 V. Right sphere set to 1 V.

In Fig 4.6, the left sphere is set to 1 V but the right sphere is set to 0V. We see that the positively charged left sphere causes negative charges to accumulate where they are closest.

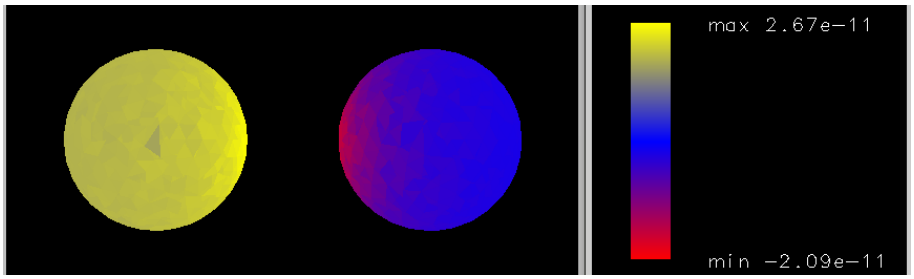

Figure 4.6: Two spheres separated by a center to center distance of three radii. Left sphere set to 1 V. Right sphere set to 0 V.

In Fig 4.7, the left sphere is set to 1 V but the right sphere is set to -1V. We see that there is a gradual transition from positive surface charge density to negative surface charge density as we move left to right.

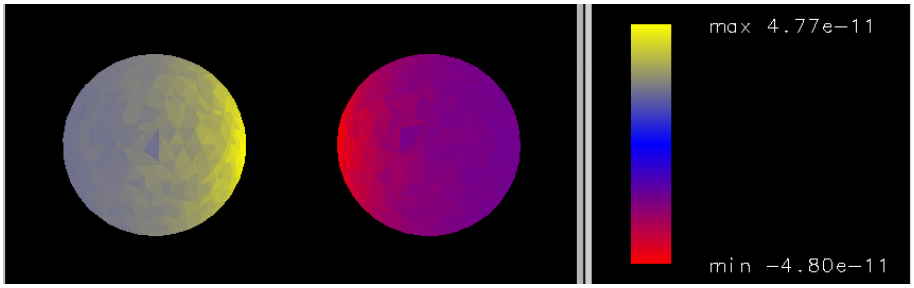

Figure 4.7: Two spheres separated by a center to center distance of three radii. Left sphere set to 1 V. Right sphere set to -1 V.

In Fig 4.8, both wires are set to 1 V. The charges accumulate along the sharp points of the geometry along the edges and corners, as expected.

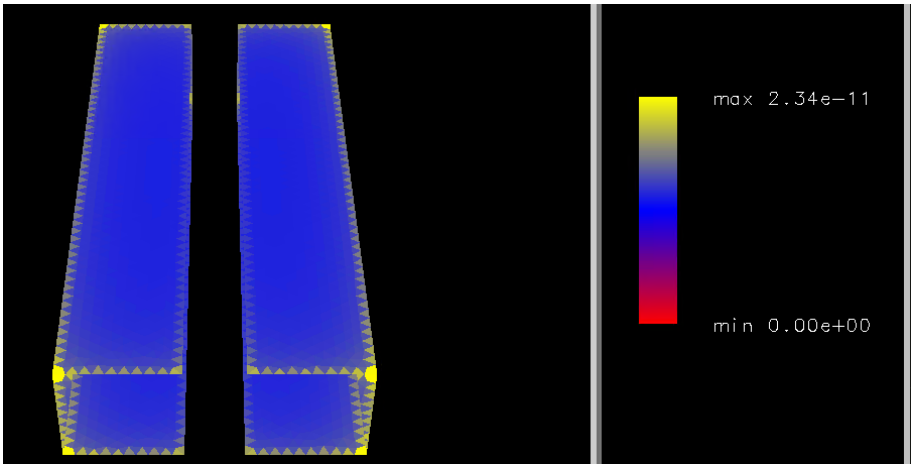

Figure 4.8: Two thin wires both set to 1 V.

In conclusion, the quantitative check against the known PEC sphere surface charge density and the qualitative checks match the expected electrostatic charge behavior. So the code is confirmed to be working correctly.

4.2 Profiling

The test case used to profile the characteristics of the ACA hierarchical matrix solver is an array of cubes (see Fig). This was chosen because there are many opportunities for the clustering algorithm to form near-field and far-field regions. So it is expected there will be large opportunity for low-rank compression to occur.

Since no analytic solution exists, the surface charge density per mesh triangle (weighted by triangle area) is used as the metric for accuracy. This value quickly converged to the same value within 3 decimal places after the mesh had more than 5000 mesh elements. This is to say that all test runs using more than 5000 mesh elements are deemed sufficiently accurate. So only the memory and time complexity is examined.

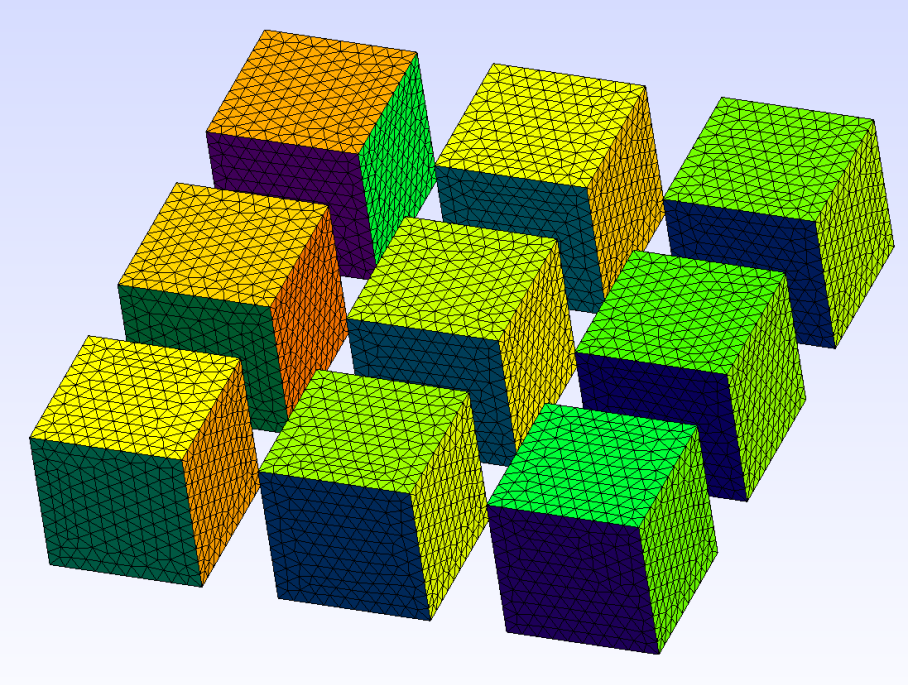

Figure 4.9: The test case for profiling the solver is an array of cubes.

4.2.1 Memory Usage

Test memory usage of the ACA method was measured to be $O(N^{1.412} \log N)$. This is close to the value reported by Zhao in [6] of $O(N^{4/3} \log N)$ and the value reported by Gibson in [5] of $O(N^{1.39} \log N)$. The uncompressed matrix should scale with space complexity $O(N^2)$ as no compression is applied. The observed scaling of $O(N^{1.412})$ is slightly better than predicted quadratic scaling.

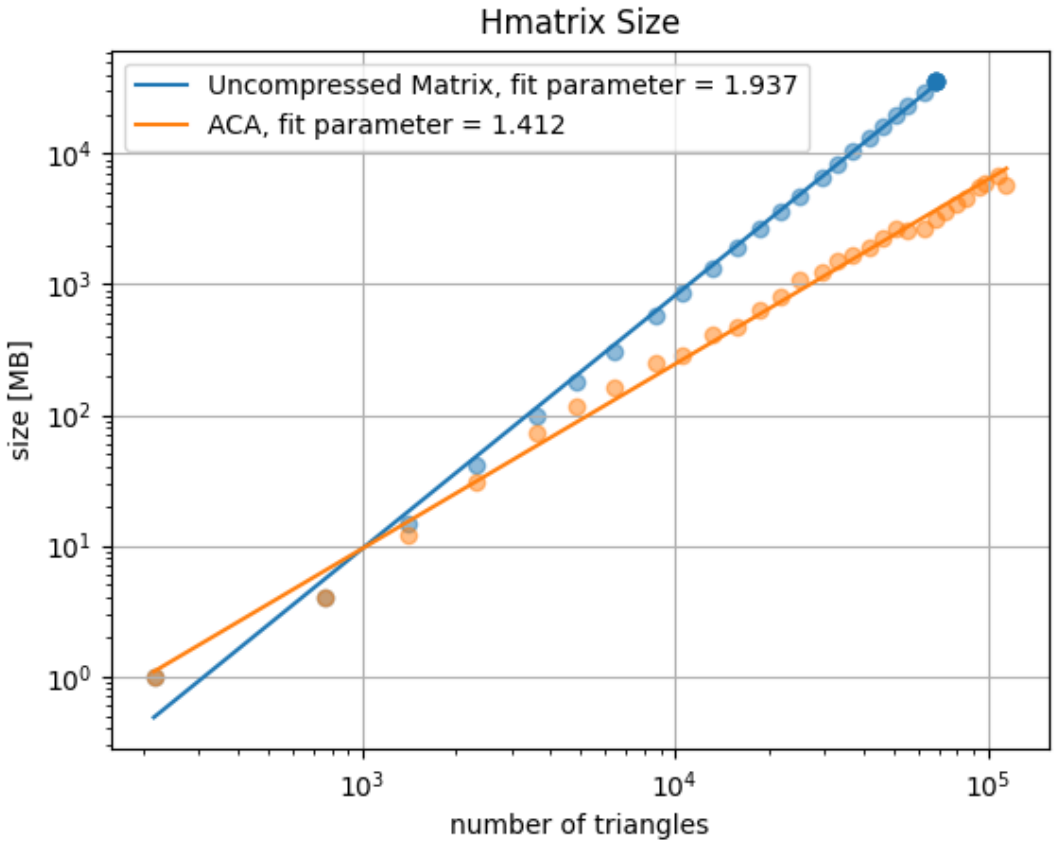

Figure 4.10: Memory usage of the hierarchical ACA method versus the uncompressed dense matrix. The ACA method was fit with $O(N^p \log N)$ and the uncompressed method was fit with $O(N^p)$, where p is the fitting parameter

The compression ratio CR was used as a figure of merit. It is defined as

$$\mathrm{CR} = (1 - \mathbf{A}_{\mathrm{ACA}}/\mathbf{A}_{\mathrm{uncompressed}}) \cdot 100\%, \tag{4.6}$$

where $\mathbf{A}_{\mathrm{ACA}}$ and $\mathbf{A}_{\mathrm{uncompressed}}$ are the sizes of the ACA and uncompressed matrices respectively. We see in Fig 4.11 the ACA method is far superior in memory usage. It offers up to 91% matrix size compression. The compression generally gets better the larger the problem becomes. The compression only starts when the mesh contains more than 1000 mesh elements. This is due to the low-rank approximation not being aggressive enough. The individual mesh elements are larger, so only clusters of points very far away are admissible. This means little low-rank approximation is taking place and the matrix is essentially still dense.

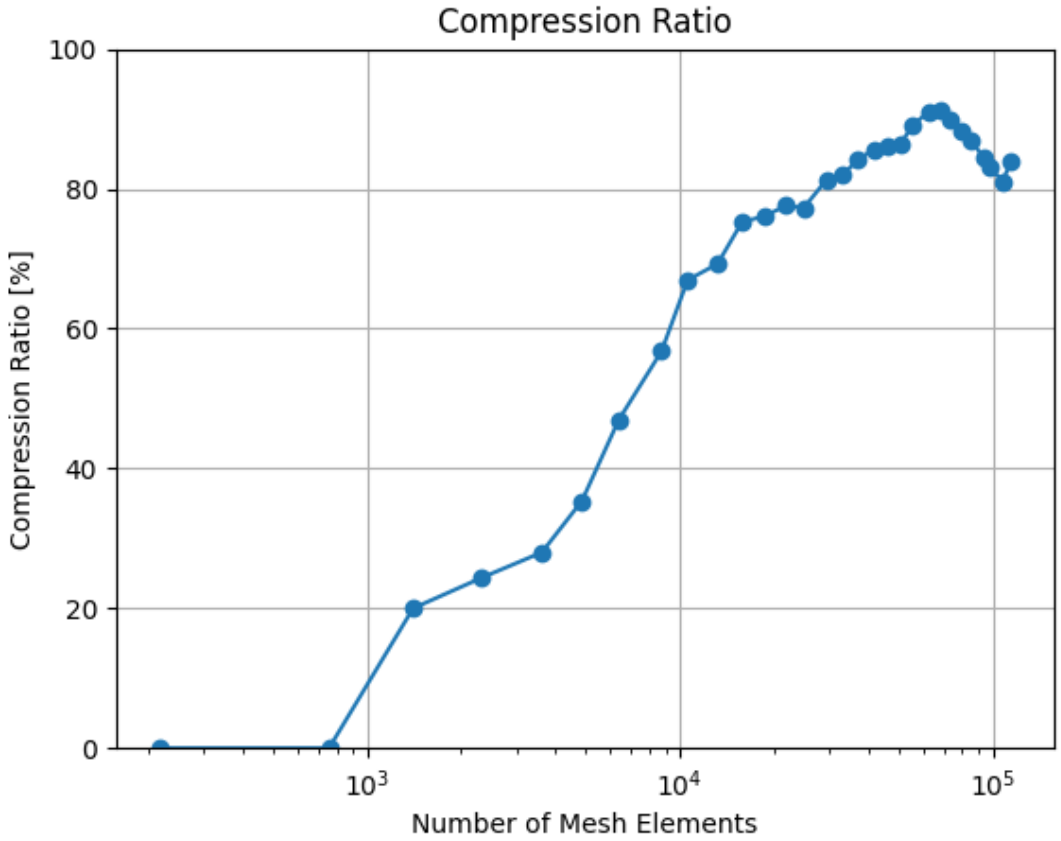

Figure 4.11: The ACA method allows up to 91% matrix size compression compared to the uncompressed version

4.2.2 Matrix Assemble Time

The matrix assemble time of ACA and the uncompressed matrix scale with virtually the same time complexity. The uncompressed matrix time complexity scales with $O(N^{1.788})$ while the ACA scales with $O(N^{1.822})$. The assemble time of the uncompressed matrix is faster, as expected. But surprisingly, the ACA is only slightly slower and is growing at roughly the same rate as the uncompressed version. This is desirable as the little additional time spent during assembly to compute the low-rank approximations will pay off in the preceding LU factorization step.

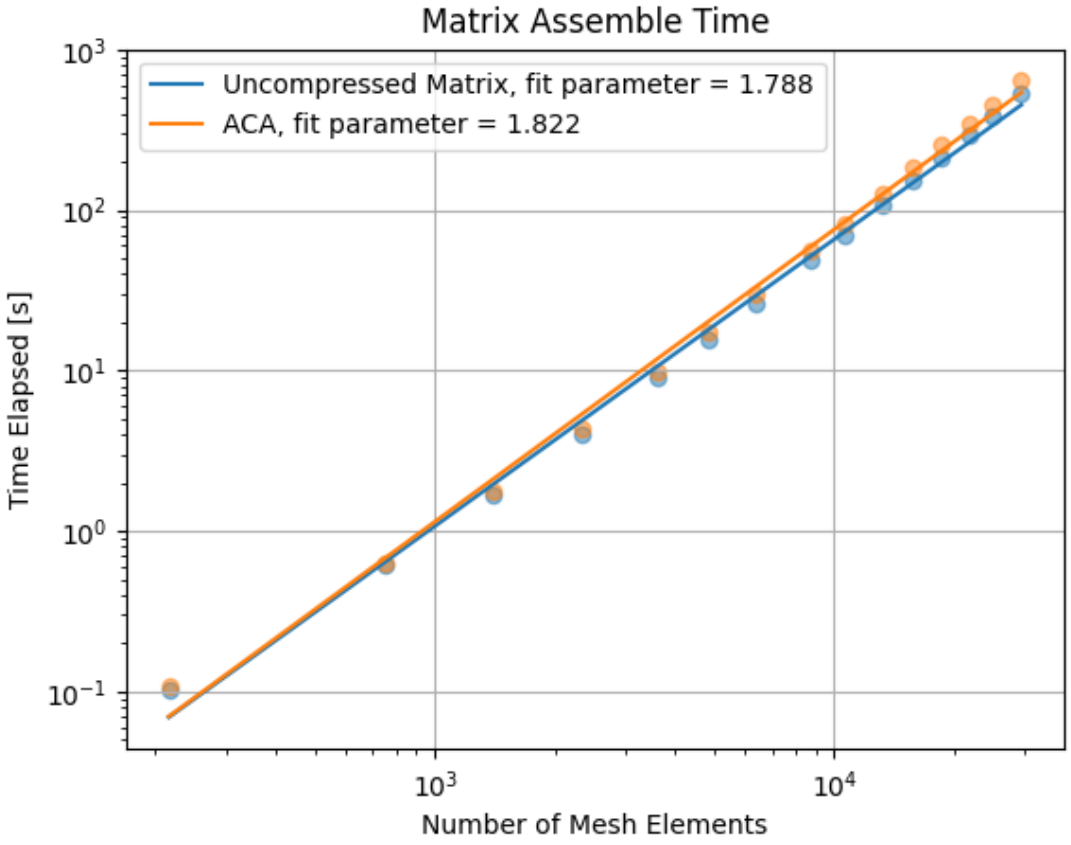

Figure 4.12: The time to assemble the matrix scales with roughly the same power relation.

4.2.3 Matrix Solve Time

The matrix is solved via a direct LU method. A standard LU factorization has time complexity $O(N^3)$. And the back-substitution step has time complexity $O(N^2)$. Since the limiting factor is the factorization step, only this step is studied. It's noted that during experiments runs, the factorization would take hundreds or thousands of seconds, but the back-substitution step rarely took more than 1 second.

The theoretical $O(N^3)$ time complexity is exactly seen in the uncompressed matrix as the fit is $O(N^{3.005})$. Using ACA compression, the number of computations required to perform the LU factorization is reduced by exploiting the low-rank block structure. Thus the time complexity is improved to $O(N^{2.332})$.

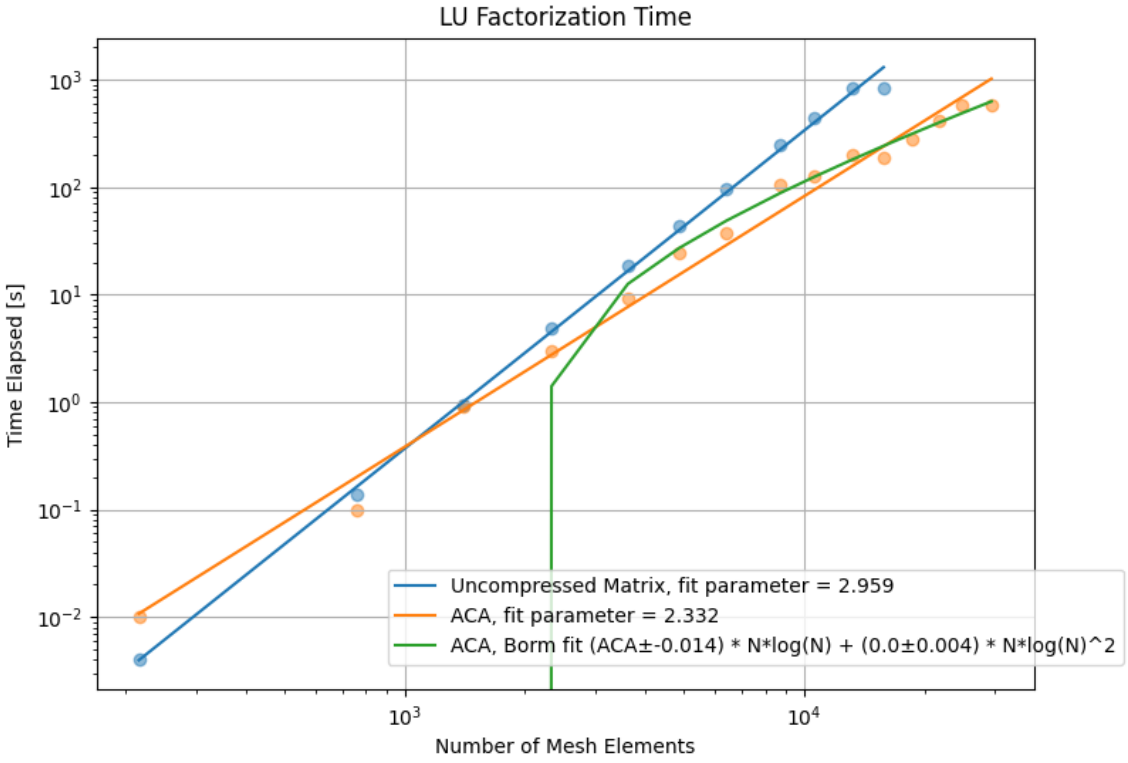

Figure 4.13: The ACA method allows a faster LU factorization because the low-rank approximations enable less matrix entries to be operated on, thus reducing the number of computations needed.

However closely looking at Fig 4.13, the LU factorization times in the large mesh region start to slightly curve downward. The polynomial fit overfits the large mesh region and underfits the small mesh region. In fact at the smallest mesh size, the LU factorization is actually slower than no ACA compression at all. The time complexity depends on the exact

implementation of LU factorization, but the time complexity reported in [17] by Borm (who also wrote the H2Lib matrix being used) that it is not larger than $O(k^3 N \log N + k^2 N \log^2 N)$. This is the green fit on Fig 4.13. It is unable to fit the smaller mesh size region.

Chai reported in [2] they were able to achieve linear time complexity if a nested basis is used to further compress the hierarchical matrix. In literature these new structures are called $\mathbb{H}^2$ matrices, but it is beyond the scope of this thesis work.

4.3 Parameter Optimization

Besides the ACA error threshold which was always set to 10^{-3}, there are two other parameters that can be tuned. They are the minimum leaf size and η admissibility control parameter. For the previous data, the minimum leaf size was 128 and $\eta = 1$. But these two parameters control how aggressively the hierarchical matrices are clustered, thus affecting how aggressively the blocks are approximated in low-rank form.

4.3.1 Minimum Leaf Size

The minimum leaf size refers to the minimum number of mesh elements in a cluster before the geometric bisection clustering algorithm stops. A lower number means more the cluster refinement will continue longer and the size of the clusters when stopped will be smaller. We see in Fig 4.14 that the size of the hierarchical matrix experiences a local minimum when the minimum leaf size is 16.

For this data collection, the admissibility parameter was set to $\eta = 1$.

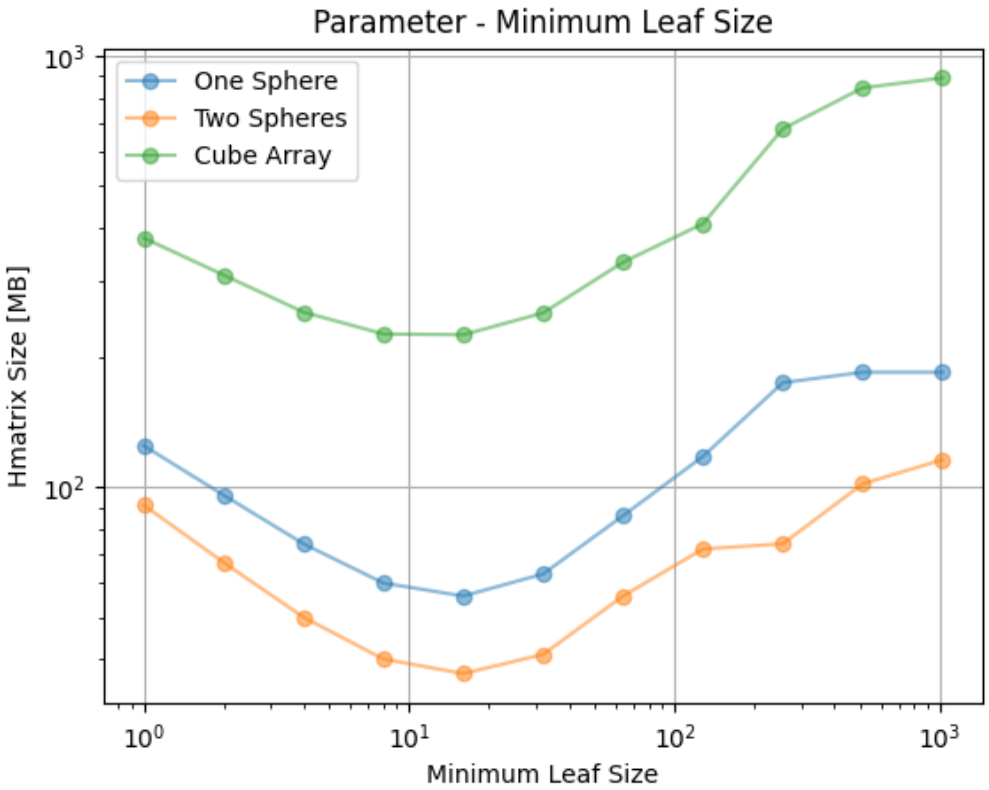

Figure 4.14: There is an ideal minimum leaf size that doesn't seem to vary between mesh geometries.

However this is at a direct trade-off with accuracy. With larger minimum leaf size, the lowest level clustering stays coarse. There are less admissible blocks. Refer to Fig 2.4. The refinement level is stopped when there are more red in-admissible blocks. These blocks are not approximated but kept as dense and calculated exactly. Thus it is more accurate. This likely needs to be tuned for each geometry. But for the PEC sphere, it seems the minimum leaf size should not be decreased beyond 64, or else it will suffer a serious loss in accuracy.

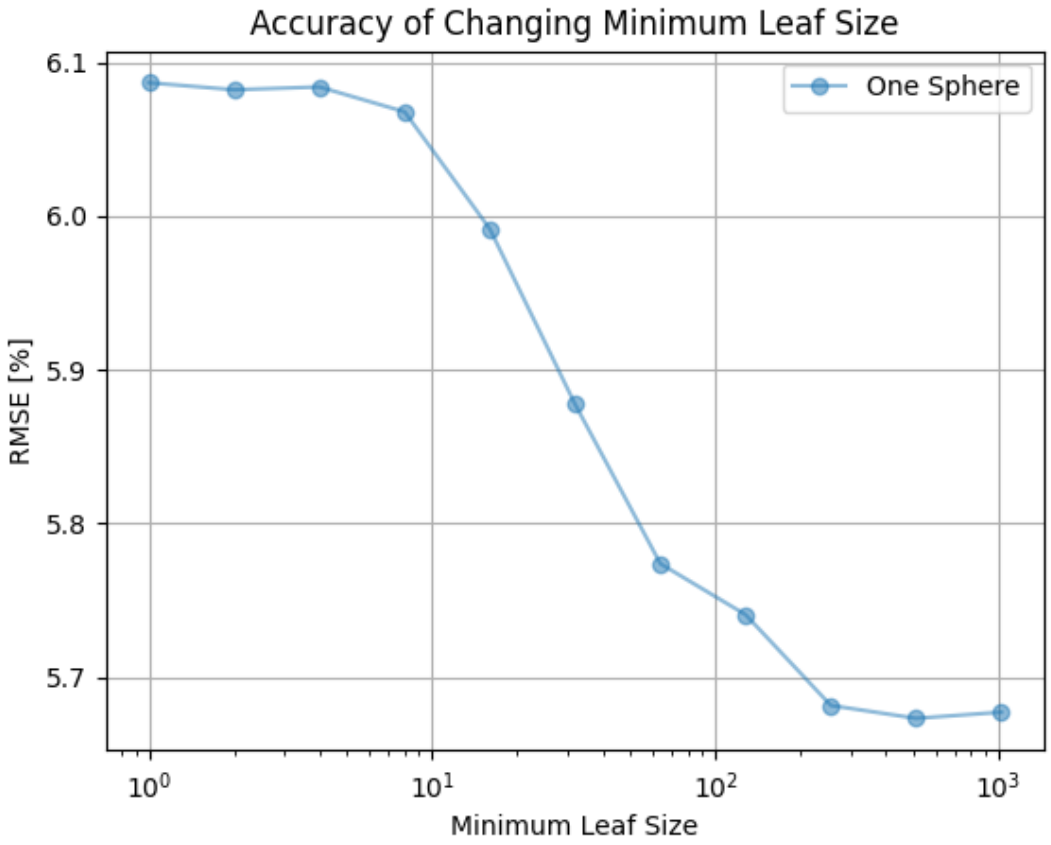

Figure 4.15: Decreasing the minimum leaf size almost always lowers accuracy. For the perfect sphere where the theoretical result is known, a large drop in accuracy occurs when η is decreased beyond 64.

4.3.2 Admissibility Control Parameter

Refer to (1.5) for the definition of η. In general, a larger η allows regions that are closer together to be low-rank approximated. But the validity of the low-rank approximation breaks down when the two cluster of points are in the near-field of each other. Thus although increasing η increases the compression and accelerates the LU factorization, it is at the expense of accuracy. See Fig 4.18.

For this data collection, the minimum leaf size was set to 16, as from the previous experiment it was observed 16 is the best value for reducing the size of compressed matrix.

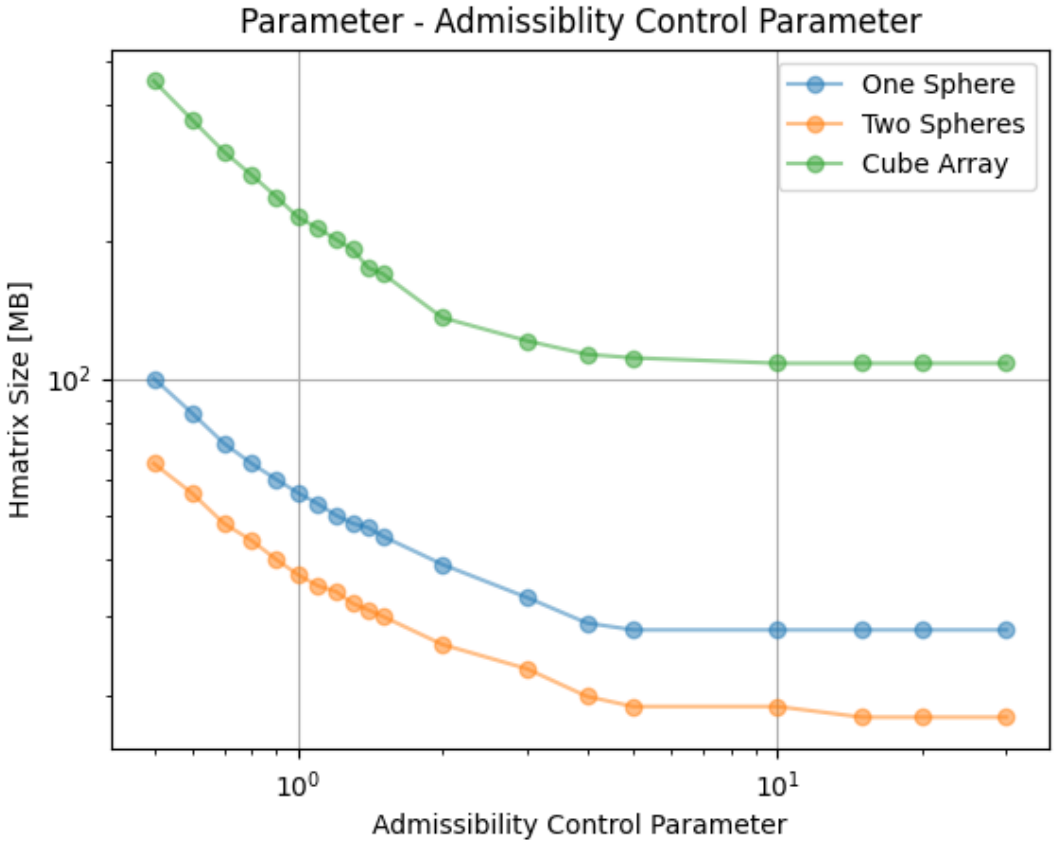

Figure 4.16: ACA compression is higher when η is higher, corresponding to more aggressively allowing admissible blocks to be low-rank approximated.

Zooming in on the log-log linear section near low η values, we see the fit is approximately $O(N^{-0.7})$ to $O(N^{-0.876})$. That means increasing η by 2x around $\eta \approx 1$ results in a 1.6-1.8x decrease in ACA compressed size.

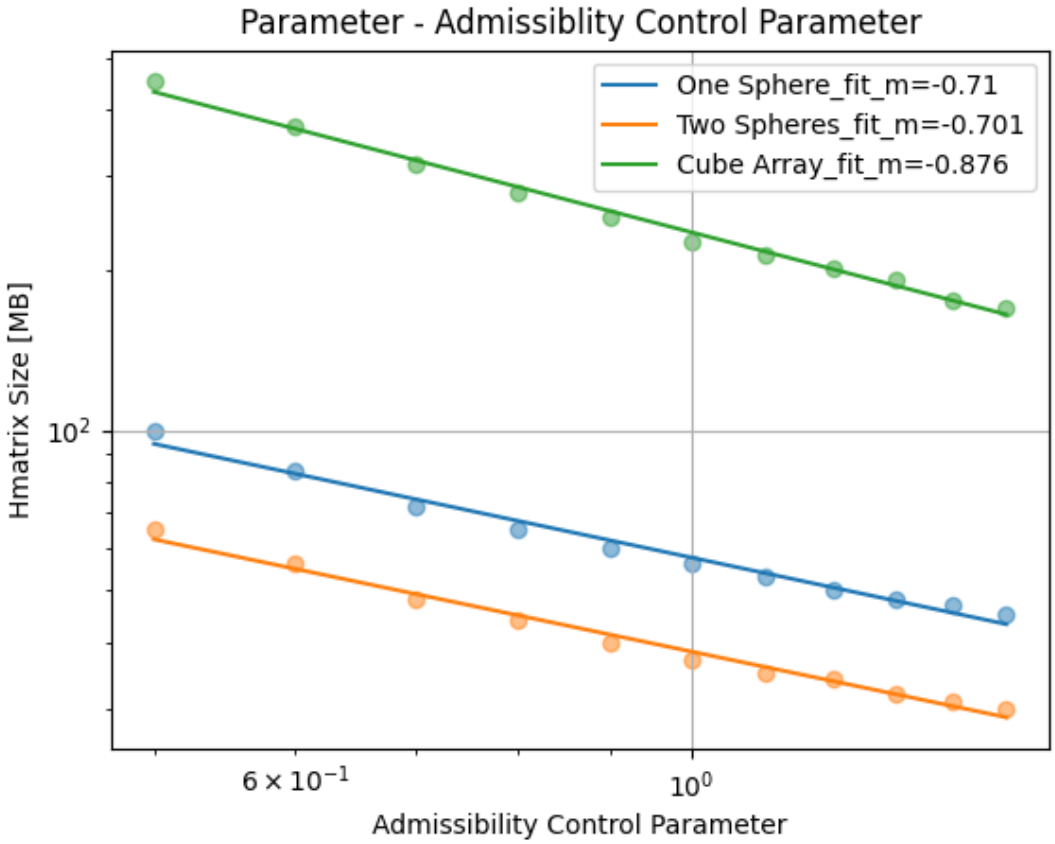

Figure 4.17: Slightly changing the value of η around $\eta \approx 1$ can provide large increases in ACA compression.

To be on the safe side, the η should be kept to the default value of 1. And it should only be carefully increased if memory savings are crucial. Fig 4.18 shows that a large drop-off in accuracy occurs if η is increased beyond 2.

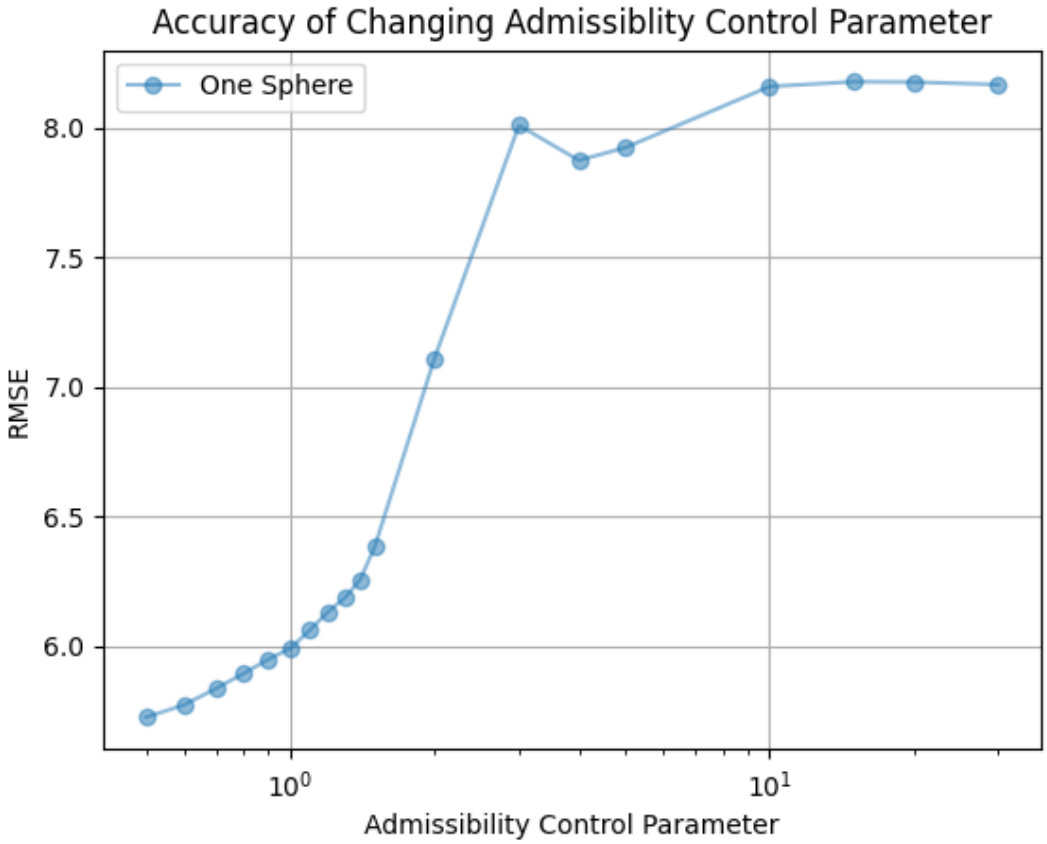

Figure 4.18: Increasing η almost always lowers accuracy. For the perfect sphere where the theoretical result is known, a large drop in accuracy occurs when η is increased beyond 2.

Chapter 5

Future Work and Conclusion

5.1 Future Work

Only a very narrow slice of the research field was studied. At every step, there are many possible algorithms that can be substituted. For example, the ACA method is one way to approximate the low-rank blocks. Other methods such as tensor product interpolation and a hybrid ACA have been proposed in [13] and [17]. Further characterizing these methods would be useful.

Another direction could be to extend the formulation to a general frequency case. Right now the electrostatic kernel is used, which is the simplest case and does not use complex numbers. But since these algebraic methods are kernel independent, it is theoretically possible to extend the code to enable a generalized Green's function kernel that works at all frequencies. And one should still be able to recover the DC results presented in this thesis.

Lastly, larger multi-layered structures could be tried. This would more rigorously test the algebraic clustering and ACA technique to see if the memory and performance savings hold up to designs more typical of modern semiconductor layouts.

5.2 Conclusion

Hierarchical matrices are a mathematical container that allows the dense Method of Moments matrix to be selectively represented in low-rank compressed blocks and uncompressed blocks. The Adaptive Cross Approximation method was used to compute the low-rank approximations. It is show to have significant memory savings of up to 90% compared with the uncompressed matrix. This compressed matrix format was able to speed up the direct LU factorization from $O(N^3)$ to $O(N^{2.332})$. However, care must be taken to tune the minimum

leaf size and η admissibility parameters to provide a good balance between accuracy and compute resources.

References

[1] W. C. Gibson, *The Method of Moments in Electromagnetics*, 2 edition. Boca Raton: Chapman and Hall/CRC, Jul. 10, 2014, 450 pp.

[2] W. Chai and D. Jiao, "Fast $\cal h$ -matrix-based direct integral equation solver with reduced computational cost for large-scale interconnect extraction," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 3, no. 2, pp. 289–298, Feb. 2013, Conference Name: IEEE Transactions on Components, Packaging and Manufacturing Technology.

[3] J. Ballani and D. Kressner, "Matrices with hierarchical low-rank structures," in *Exploiting Hidden Structure in Matrix Computations: Algorithms and Applications : Cetraro, Italy 2015*, ser. Lecture Notes in Mathematics, M. Benzi, D. Bini, D. Kressner, *et al.*, Eds., Cham: Springer International Publishing, 2016, pp. 161–209.

[4] J. Shaeffer, "Low rank matrix algebra for the method of moments," *The Applied Computational Electromagnetics Society Journal (ACES)*, pp. 1052–1059, 2018.

[5] W. C. Gibson, *The Method of Moments in Electromagnetics*, 3rd ed. New York: Chapman and Hall/CRC, Sep. 6, 2021, 510 pp.

[6] K. Zhao, M. Vouvakis, and J.-F. Lee, "The adaptive cross approximation algorithm for accelerated method of moments computations of EMC problems," *IEEE Transactions on Electromagnetic Compatibility*, vol. 47, no. 4, pp. 763–773, Nov. 2005, Conference Name: IEEE Transactions on Electromagnetic Compatibility.

[7] Y. Zhao and J. Mao, "Equivalent surface impedance-based mixed potential integral equation accelerated by optimized $\cal h$ -matrix for 3-d interconnects," *IEEE Transactions on Microwave Theory and Techniques*, vol. 66, no. 1, pp. 22–34, Jan. 2018, Conference Name: IEEE Transactions on Microwave Theory and Techniques.

[8] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, no. 5, pp. 1225–1251, Sep. 1996, Conference Name: Radio Science.

[9] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, Dec. 1, 1987.

[10] N. Engheta, W. Murphy, V. Rokhlin, and M. Vassiliou, "The fast multipole method (FMM) for electromagnetic scattering problems," *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 6, pp. 634–641, Jun. 1992, Conference Name: IEEE Transactions on Antennas and Propagation.

[11] S. Rao, D. Wilton, and A. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 3, pp. 409–418, May 1982, Conference Name: IEEE Transactions on Antennas and Propagation.

[12] E. S. Øyre, "Electromagnetic scattering calculations for arbitrarily shaped closed surfaces using the method of moments," Accepted: 2021-09-28T18:40:39Z, Master thesis, NTNU, 2021.

[13] S. Börm and L. Grasedyck, "Hybrid cross approximation of integral operators," *Numerische Mathematik*, vol. 101, no. 2, pp. 221–249, Aug. 1, 2005.

[14] M. Bebendorf, "Approximation of boundary element matrices," *Numerische Mathematik*, vol. 86, no. 4, pp. 565–589, Oct. 1, 2000.

[15] W. Hackbusch, "A sparse matrix arithmetic based on $\cal h$-matrices. part i: Introduction to ${\cal h}$-matrices," *Computing*, vol. 62, no. 2, pp. 89–108, Apr. 1, 1999.

[16] S. A. Sauter, "The panel clustering method in 3-d bem," in *Wave Propagation in Complex Media*, G. Papanicolaou, Ed., red. by A. Friedman and R. Gulliver, vol. 96, Series Title: The IMA Volumes in Mathematics and its Applications, New York, NY: Springer New York, 1998, pp. 199–224.

[17] S. Borm, L. Grasedyck, and W. Hackbusch. "WS_hmatrices.pdf." (2005), [Online]. Available: `https://www.mis.mpg.de/scicomp/Fulltext/WS_HMatrices.pdf`.

[18] S. Börm and S. C. G. at Kiel University. "H2lib, a library for hierarchical matrices." (), [Online]. Available: `http://www.h2lib.org/index.html`.

[19] C. Geuzaine and J.-F. Remacle. "Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities." (), [Online]. Available: `https://gmsh.info/`.