Chapter 11

Fast Multipole Method

In the previous two chapters, we applied the ACA and the MLACA to a direct solution of the MoM system matrix in compressed form using LU factorization. Overall, the compression levels were very good, with both methods having a storage complexity that scaled approximately as $O(N^{4/3} \log N)$. Compared to the ACA, MLACA was found to yield a much greater amount of compression, particularly for larger problems, at the expense of additional compute time in compressing the matrix blocks.

In this chapter, we introduce the Fast Multipole Method (FMM), which also exploits the rank-deficient nature of the MoM system matrix but approaches the solution in a different way. Whereas the ACA and MLACA compute and store all matrix blocks, with off-diagonal blocks in compressed form, in the FMM only the diagonal and a fraction of the off-diagonal blocks are computed and stored directly. The remaining matrix elements are instead evaluated on the fly in aggregated form as part of the matrix-vector product in an iterative solver. In this way, the FMM achieves an $O(N^{3/2})$ complexity [1], at the expense of requiring an iterative solution. If the geometry is divided in a recursive, hierarchical way, the result is a Multi-Level Fast Multipole Algorithm (MLFMA), which has an improved $O(N \log N)$ complexity [2] but requires a more complex software implementation. It is the MLFMA to which we devote much of our attention in this chapter.

The treatment that follows is novel in that it fuses several elements of the ACA with MLFMA, in particular its use of a block matrix approach and SVD-based compression of the off-diagonal matrix blocks and the FMM radiation and receive functions. This approach, which we call the MLFMA/SVD, retains the $O(N \log N)$ storage complexity but yields greater memory savings than the standard MLFMA.

11.1 The N-Body Problem

Consider the N-body problem, which is encountered in many areas of applied physics. A classic example is the motion of celestial objects interacting through gravitational attraction. For example, if we wish to know the position and velocity of a star at some point in time, the differential equations of motion must be numerically integrated in

time. This requires a calculation of the gravitational potential due to all other nearby stars at each time step in the integration. Similarly, as the other stars are also moving, the gravitational forces on them, and their positions and velocities, must also be computed. This results in an $N \times N$ problem at each point in time, where N is the total number of stars. When N is small, a brute-force pairwise calculation of the forces can be carried out at each time step. However, when N becomes very large, the number of calculations in the brute-force approach may grow so large as to be impractical or impossible.

The Fast Multipole Method (FMM), originally introduced by Greengard and Rokhlin in [3], was successful in reducing the complexity of this N-body problem. In this method, the gravitational potentials of a group of closely spaced particles are rewritten in terms of their multipole expansions. These expansions are combined into a single function representing the potential of the entire group. The interaction between that group and any particles far away from that group can then be computed using the multipole expansion. Thus, the FMM allows for grouping together many nearby particles and treating them as if they are a single particle. The MoM system matrix can be viewed as another type of N-body problem, where each matrix element comprises the field radiated by one basis function and received by another.

As we will see, the FMM can be applied to vector electromagnetic problems. However, the multipole expansions are specific to the Green's function and must be derived for the electromagnetic case. This differs from the ACA approach, which is method-agnostic and requires no knowledge of the underlying problem.

11.2 Matrix-Vector Product

Let us first consider a matrix-vector product in the context of the Method of Moments, written as

$$\mathbf{x} = \mathbf{Z}\mathbf{y} \,, \tag{11.1}$$

where the system matrix $\mathbf{Z}$ represents the interactions (fields) between source and testing functions, and the right-hand side vector $\mathbf{y}$ comprises the excitation of each basis function. The rows of the result vector $\mathbf{x}$ comprise the coherent sum of the fields "received" by each testing function due to the fields "radiated" by all basis functions. Noting that the amplitude of the interaction between a source and testing pair diminishes quickly versus distance, let us divide the space around each basis function into two regions: a *near*-field region having stronger interactions, and a *far*-field region with weaker interactions. This is illustrated in Figure 11.1, where point A is at the center of a shaded circular region of diameter d. All points inside the circular region are said to lie in the *near* region, and those outside in the *far* region. Note that though they are similarly named, these should not be confused with the near and far field regions described in Section 3.5.

Consider next row m of $\mathbf{Z}$, which we denote as $\mathbf{z}_{m*}$. The product of this row with the column vector $\mathbf{y}$ yields the column vector element x_m given by

$$x_m = \mathbf{z}_{m*}\, \mathbf{y} \,, \tag{11.2}$$

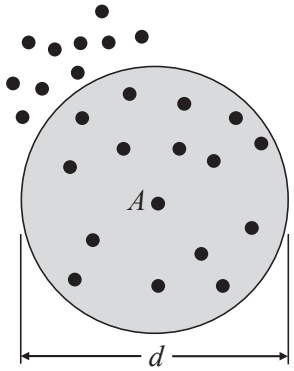


Figure 11.1: Near and Far Points

which comprises the fields received by testing function m due to all basis functions. As some basis functions are in the near region of testing function m and the others are in the far region, the two vectors can be re-ordered and divided so that (11.2) now reads as

$$x_m = \mathbf{z}_{m*}\,\mathbf{y} = \mathbf{z}_{m*}^{near}\,\mathbf{y}^{near} + \mathbf{z}_{m*}^{far}\,\mathbf{y}^{far}, \tag{11.3}$$

where the elements in the sub-vectors $\mathbf{z}_{m*}^{near}$ and $\mathbf{y}^{near}$ are due to the source functions in the region near to testing function m, and those in $\mathbf{z}_{m*}^{far}$ and $\mathbf{y}^{far}$ are due to those in the far region. Extending this concept, if we now write the system matrix as

$$\mathbf{Z} = \mathbf{Z}^{near} + \mathbf{Z}^{far}, \tag{11.4}$$

where $\mathbf{Z}^{near}$ and $\mathbf{Z}^{far}$ are sparse, the matrix-vector product can be written as the sum of a *near product* and a *far product*, yielding

$$\mathbf{x} = \mathbf{x}^{near} + \mathbf{x}^{far} = \mathbf{Z}^{near}\mathbf{y} + \mathbf{Z}^{far}\mathbf{y}, \tag{11.5}$$

where $\mathbf{x}^{near}$ and $\mathbf{x}^{far}$ are sparse vectors. In the FMM, the matrix-vector product comprises two steps: a *near* product with $\mathbf{Z}^{near}$, where the elements are computed and stored explicitly, and a *far* product with $\mathbf{Z}^{far}$, which is computed in a different way on the fly. We will discuss the practical implementation of these steps throughout the rest of this chapter. As we will see, one of the primary benefits of the FMM is that $\mathbf{Z}^{near}$ is extremely sparse, greatly reducing the memory requirements. In addition, we will use the SVD to compress some of the off-diagonal blocks in $\mathbf{Z}^{near}$, reducing the memory footprint even further.

In the rest of this section, we will consider the far part of the matrix-vector product in more detail. First we will address the concepts of the addition theorem and wave translation, which allow us to treat groups of basis functions as a single group. Following this, we will then develop the FMM-specific expressions used to compute the far matrix elements.

11.2.1 Addition Theorem

Consider again the three-dimensional Green's function (3.39), written as

$$G(\mathbf{r},\mathbf{r}') = \frac{e^{-jk|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|}\,, \tag{11.6}$$

where $\mathbf{r}'$ and $\mathbf{r}$ are the source and field points, respectively. If we now add to $\mathbf{r}$ an offset $\mathbf{x}$, (11.6) can be written as

$$\frac{e^{-jk|\mathbf{r}-\mathbf{r}'+\mathbf{x}|}}{4\pi|\mathbf{r}-\mathbf{r}'+\mathbf{x}|} = \frac{e^{-jk|\mathbf{R}+\mathbf{x}|}}{4\pi|\mathbf{R}+\mathbf{x}|}\,, \tag{11.7}$$

where $\mathbf{R} = \mathbf{r}-\mathbf{r}'$. Provided that $|\mathbf{x}| < |\mathbf{R}|$, we now define the *addition theorem* for spherical waves, given by [1, 2]

$$\frac{e^{-jk|\mathbf{R}+\mathbf{x}|}}{4\pi|\mathbf{R}+\mathbf{x}|} = -\frac{jk}{4\pi}\sum_{l=0}^{\infty}(-1)^l(2l+1)j_l\big(k|\mathbf{x}|\big)h_l^{(2)}\big(k|\mathbf{R}|\big)P_l\big(\hat{\mathbf{x}}\cdot\hat{\mathbf{R}}\big)\,, \tag{11.8}$$

where $j_l(x)$ is a spherical Bessel function of the first kind, $P_l(x)$ is a Legendre polynomial of order l, and $h_l^{(2)}(x)$ is a spherical Hankel function of the second kind, i.e.,

$$h_l^{(2)}(x) = \sqrt{\frac{\pi}{2x}}H_{l+1/2}^{(2)}(x) = j_l(x) - jn_l(x)\,, \tag{11.9}$$

where $j_l(x)$ and $n_l(x)$ are spherical Bessel functions of the first and second kinds, respectively. Examining (11.8) closely, we see that it is a way of writing a wave radiating from a point as if it were radiating from a different point nearby. Following Stratton [4], we can convert (11.8) to a surface integral on the unit sphere via

$$4\pi(-j^l)j_l\big(k|\mathbf{x}|\big)P_l\big(\hat{\mathbf{x}}\cdot\hat{\mathbf{R}}\big) = \oint_1 e^{-jk\hat{\mathbf{k}}\cdot\mathbf{x}}P_l\big(\hat{\mathbf{k}}\cdot\hat{\mathbf{R}}\big)\,dS\,, \tag{11.10}$$

where $\hat{\mathbf{k}}$ is the radially oriented unit vector on the sphere. Using this expression allows us to then write

$$\frac{e^{-jk|\mathbf{R}+\mathbf{x}|}}{4\pi|\mathbf{R}+\mathbf{x}|} = \frac{k}{(4\pi)^2}\oint_1 e^{-jk\hat{\mathbf{k}}\cdot\mathbf{x}}\sum_{l=0}^{\infty}(-j)^{l+1}(2l+1)h_l^{(2)}\big(k|\mathbf{R}|\big)P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{R}})\,dS\,, \tag{11.11}$$

where the integration and summation have been exchanged. This exchange will be legitimate provided that we truncate the summation at some finite order L, which limits the maximum value of $|\mathbf{R}|$. Doing so, we can now write (11.11) as

$$\frac{e^{-jk|\mathbf{R}+\mathbf{x}|}}{4\pi|\mathbf{R}+\mathbf{x}|} = \oint_1 e^{-jk\hat{\mathbf{k}}\cdot\mathbf{x}}\,T_L(k,\hat{\mathbf{k}},\mathbf{R})\,dS\,, \tag{11.12}$$

where

$$T_L(k,\hat{\mathbf{k}},\mathbf{R}) = \frac{k}{(4\pi)^2}\sum_{l=0}^{L}(-j)^{l+1}(2l+1)h_l^{(2)}(k|\mathbf{R}|)P_l(\hat{\mathbf{k}}\cdot\hat{\mathbf{R}})\,. \tag{11.13}$$

We refer to $T_L(k, \hat{\mathbf{k}}, \mathbf{R})$ as the *transfer function*, as it allows us to convert outgoing spherical waves at a source point to a set of incoming spherical waves at a field point. In practice, the integration on the unit sphere in (11.12) is carried out numerically with quadratures in the θ and ϕ dimensions. We will discuss the types of quadratures and required sampling rates in more detail in a later section.

11.2.2 Wave Translation

Let us illustrate the addition theorem by calculating the Green's function between a source point $\mathbf{r}'$ and observation point $\mathbf{r}$. We first define the vector

$$\mathbf{v} = \mathbf{r} - \mathbf{r}', \tag{11.14}$$

and then introduce two points $\mathbf{a}$ and $\mathbf{b}$ that are very close to $\mathbf{r}$ and $\mathbf{r}'$, respectively, as shown in Figure 11.2. We can now write (11.14) as

$$\mathbf{v} = (\mathbf{r} - \mathbf{a}) + (\mathbf{a} - \mathbf{b}) - (\mathbf{r}' - \mathbf{b}) \ , \tag{11.15}$$

or

$$\mathbf{v} = \mathbf{r}_{ra} + \mathbf{r}_{ab} - \mathbf{r}_{r'b} \ . \tag{11.16}$$

Using (11.16), we can now write (11.12) as

$$\frac{e^{-jk|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} = \int_1 e^{-jk\hat{\mathbf{k}}\cdot(\mathbf{r}_{ra}-\mathbf{r}_{r'b})} T_L(k, \hat{\mathbf{k}}, \mathbf{r}_{ab}) \, dS \ , \tag{11.17}$$

where

$$T_L(k, \hat{\mathbf{k}}, \mathbf{r}_{ab}) = \frac{k}{(4\pi)^2} \sum_{l=0}^{L} (-j)^{l+1} (2l+1) h_l^{(2)}\big(k|\mathbf{r}_{ab}|\big) P_l\big(\hat{\mathbf{k}} \cdot \hat{\mathbf{r}}_{ab}\big) \ . \tag{11.18}$$

This is an important result, as the transfer function depends only on $\mathbf{r}_{ab}$. If we were to move $\mathbf{r}$ and $\mathbf{r}'$ a small distance from their previous location, the transfer function remains unchanged. Therefore, we can compute the interaction between *any* two points near $\mathbf{a}$ and $\mathbf{b}$ using the *same* transfer function. Let us now write (11.17) as

$$\frac{e^{-jk|\mathbf{r}-\mathbf{r}'|}}{4\pi|\mathbf{r}-\mathbf{r}'|} = \int_1 R(\mathbf{r}, \hat{\mathbf{k}}) \, T_L(k, \hat{\mathbf{k}}, \mathbf{r}_{ab}) T(\mathbf{r}', \hat{\mathbf{k}}) \, dS \ , \tag{11.19}$$

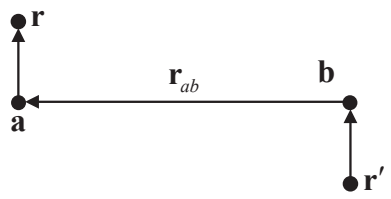


Figure 11.2: Wave Translation

where at the source point $\mathbf{r}'$, we define the *radiation function*

$$T(\mathbf{r}', \hat{\mathbf{k}}) = e^{jk\hat{\mathbf{k}}\cdot\mathbf{r}_{r'b}}, \tag{11.20}$$

and at the field point $\mathbf{r}$, we define the *receive function*

$$R(\mathbf{r}, \hat{\mathbf{k}}) = e^{-jk\hat{\mathbf{k}}\cdot\mathbf{r}_{ra}}. \tag{11.21}$$

If we move $\mathbf{r}'$ or $\mathbf{r}$ to another location, only the corresponding radiation or receive function changes. As a result, we can now compute a sum of the Green's functions evaluated at the field point $\mathbf{r}$ due to *many* source points $\mathbf{r}'_n$ located close to $\mathbf{b}$. Using (11.19), this can be written as

$$\sum_{n=1}^{N} \frac{e^{-jk|\mathbf{r}-\mathbf{r}'_n|}}{4\pi|\mathbf{r}-\mathbf{r}'_n|} = \int_1 R(\mathbf{r}, \hat{\mathbf{k}})\, T_L(k, \hat{\mathbf{k}}, \mathbf{r}_{ab}) \sum_{n=1}^{N} T_n(\mathbf{r}'_n, \hat{\mathbf{k}})\, dS\,, \tag{11.22}$$

where the individual radiation functions are

$$T_n(\mathbf{r}'_n, \hat{\mathbf{k}}) = e^{jk\hat{\mathbf{k}}\cdot\mathbf{r}_{r'_n b}}\,. \tag{11.23}$$

It is (11.22) that allows us to quickly compute the far matrix-vector product. The radiation functions for all source points are coherently added (*aggregated*) to create a local field at $\mathbf{b}$. This field is then *transmitted* via the transfer function, yielding a local field at $\mathbf{a}$. The local field is then multiplied with the receive function at the field point and integrated over the unit sphere (*disaggregated*), yielding the desired sum. In a practical FMM implementation, the basis and testing functions are clustered into groups. The radiation and receive functions for all basis and testing functions, as well as the transfer functions linking all groups, are precomputed. The far part of the matrix-vector product then comprises aggregations, transfers, and disaggregations between well-separated groups.

11.2.2.1 Complex Wavenumbers

In lossy dielectric regions having complex-valued wavenumbers, the transfer function (11.18) remains valid. However, special numerical routines are needed to compute the spherical Hankel function for non-integer orders and complex arguments. The subroutines by Amos [5] are ideal for this purpose, in particular the routines `cbesh` and `zbesh` for single- and double-precision complex variables, respectively. These routines are available via the Netlib repository[1], and are often integrated in software libraries for computing special functions.

11.2.3 Far Matrix Elements

Having discussed the addition theorem and the concepts of wave translation, we will now derive expressions for the radiation and receive functions used to compute the

[1] http://www.netlib.org/amos

MoM matrix elements (3.188–3.193). Though the examples in this chapter use a three-dimensional approach with the RWG basis and testing functions, the expressions presented in this section are independent of basis function. We remind the reader that though the radiation and receive functions are derived in the context of a single source-field pair, the FMM matrix-vector product operates on aggregated (summed) radiation functions and individual matrix elements are not formed explicitly.

Note that we will allow the points **a** and **b** to remain undefined for now. They will be specified later when we discuss clustering the basis functions into groups.

11.2.3.1 EFIE

The EFIE (3.181) contains $\mathcal{L}$ and $\mathcal{K}$ operators, which were discretized and tested via the MoM in (3.188) and (3.189). We will derive the corresponding radiation and receive functions in this section.

11.2.3.1.1 $\mathcal{L}$ Operator

Let us focus our attention on the second term on the right in (3.188). As the FMM interactions are not in the near zone, we do not need to redistribute the differential operators as in Section 3.6.2.3. Instead, we follow Section 3.4.4 and move them so that they operate only on the Green's function. This yields in region R_l matrix elements given by

$$Z_{mn}^{EJ(l)} = \mathrm{L}(\mathbf{f}_m, \mathbf{f}_n) \,, \tag{11.24}$$

where

$$\mathrm{L}(\mathbf{f}_m, \mathbf{f}_n) = j\omega\mu_l \int_{\mathbf{f}_m} \mathbf{f}_m(\mathbf{r}) \cdot \int_{\mathbf{f}_n} \mathbf{f}_n(\mathbf{r}') \Big[1 - \frac{1}{k_l^2} \nabla\nabla'\Big] G(\mathbf{r}, \mathbf{r}') \, d\mathbf{r}' \, d\mathbf{r} \,. \tag{11.25}$$

We next apply the differential operations to the Green's function in (11.17). This is accomplished by noting that the $T_L(k, \hat{\mathbf{k}}, \mathbf{r}_{ab})$ is not a function of r or r', and the vectors $\mathbf{r}_{ra}$ and $\mathbf{r}_{r'b}$ are radially directed vectors in field and source coordinates, respectively, with **a** and **b** as the respective origins. Inserting the result into (11.25) yields an expression of the form

$$\mathrm{L}(\mathbf{f}_m, \mathbf{f}_n) = \int_1 \mathbf{R}^{\mathcal{L}}(\mathbf{f}_m, \hat{\mathbf{k}}) \cdot T_L(k_l, \hat{\mathbf{k}}, \mathbf{r}_{ab}) \mathbf{T}^{\mathcal{L}}(\mathbf{f}_n, \hat{\mathbf{k}}) \, dS \,, \tag{11.26}$$

where the radiation function for basis function $\mathbf{f}_n(\mathbf{r}')$ is

$$\mathbf{T}^{\mathcal{L}}(\mathbf{f}_n, \hat{\mathbf{k}}) = \Big[1 - \hat{\mathbf{k}}\hat{\mathbf{k}}\Big] \cdot \int_{\mathbf{f}_n} \mathbf{f}_n(\mathbf{r}') e^{jk_l \hat{\mathbf{k}} \cdot \mathbf{r}_{r'b}} \, d\mathbf{r}' \tag{11.27}$$

and the receive function for testing function $\mathbf{f}_m(\mathbf{r})$ is

$$\mathbf{R}^{\mathcal{L}}(\mathbf{f}_m, \hat{\mathbf{k}}) = j\omega\mu_l \int_{\mathbf{f}_m} \mathbf{f}_m(\mathbf{r}) e^{-jk_l \hat{\mathbf{k}} \cdot \mathbf{r}_{ra}} \, d\mathbf{r} \,. \tag{11.28}$$

We note that the $\Big[1 - \hat{\mathbf{k}}\hat{\mathbf{k}}\Big]$ term in (11.27) removes the components of the integral along the radial vector $\hat{\mathbf{k}}$, leaving only $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ components.

11.2.3.1.2 $\mathcal{K}$ Operator

As the source and testing functions are well separated, the supports of $\mathbf{f}_m$ and $\mathbf{g}_n$ do not overlap and we retain only the first term on the right-hand side of (3.189). This yields in Region R_l matrix elements given by

$$Z_{mn}^{EM(l)} = \mathrm{K}(\mathbf{f}_m, \mathbf{g}_n) \,, \tag{11.29}$$

where

$$\mathrm{K}(\mathbf{f}_m, \mathbf{g}_n) = \int_{\mathbf{f}_m} \mathbf{f}_m(\mathbf{r}) \cdot \int_{\mathbf{g}_n} \mathbf{g}_n(\mathbf{r}') \times \nabla' G(\mathbf{r}, \mathbf{r}') \, d\mathbf{r}' \, d\mathbf{r} \,. \tag{11.30}$$

Taking the gradient of (11.17) and rearranging the cross product yields

$$\mathrm{K}(\mathbf{f}_m, \mathbf{g}_n) = \int_1 \mathbf{R}^{\mathcal{K}}(\mathbf{f}_m, \hat{\mathbf{k}}) \cdot T_L(k_l, \hat{\mathbf{k}}, \mathbf{r}_{ab}) \mathbf{T}^{\mathcal{K}}(\mathbf{g}_n, \hat{\mathbf{k}}) \, dS \,, \tag{11.31}$$

where the radiation function for basis function $\mathbf{g}_n(\mathbf{r}')$ is

$$\mathbf{T}^{\mathcal{K}}(\mathbf{g}_n, \hat{\mathbf{k}}) = \int_{\mathbf{g}_n} \mathbf{g}_n(\mathbf{r}') e^{jk_l \hat{\mathbf{k}} \cdot \mathbf{r}_{r'b}} \, d\mathbf{r}' \tag{11.32}$$

and the receive function for testing function $\mathbf{f}_m(\mathbf{r})$ is

$$\mathbf{R}^{\mathcal{K}}(\mathbf{f}_m, \hat{\mathbf{k}}) = jk_l \, \hat{\mathbf{k}} \times \int_{\mathbf{f}_m} \mathbf{f}_m(\mathbf{r}) e^{-jk_l \hat{\mathbf{k}} \cdot \mathbf{r}_{ra}} \, d\mathbf{r} \,. \tag{11.33}$$

The cross product with $\hat{\mathbf{k}}$ in (11.33) results in the receive function having only $\hat{\boldsymbol{\theta}}$ and $\hat{\phi}$ components. An η_0 scale factor must also be applied to the radiation function to account for the magnetic basis function.

11.2.3.2 MFIE

The MFIE (3.182) also has $\mathcal{K}$ and $\mathcal{L}$ operators, which were discretized and tested via the MoM in (3.190) and (3.191). Substitution of the electric basis and magnetic testing functions into (3.190) yields in Region R_l matrix elements given by

$$Z_{mn}^{HJ(l)} = -\mathrm{K}(\mathbf{g}_m, \mathbf{f}_n) \,, \tag{11.34}$$

and substitution of the magnetic basis and testing functions into (3.191) yields

$$Z_{mn}^{HM(l)} = \frac{\epsilon_l}{\mu_l} \mathrm{L}(\mathbf{g}_m, \mathbf{g}_n) \,, \tag{11.35}$$

where the L and K elements were defined in Section 11.2.3.1. An η_0 scale factor must also be applied to the radiation and receive functions to account for the presence of the magnetic basis functions, where appropriate.

11.2.3.3 nMFIE

The nMFIE (3.183) contains the $\hat{\mathbf{n}} \times \mathcal{L}$ and $\hat{\mathbf{n}} \times \mathcal{K}$ operators, which were discretized and tested via the MoM in (3.192) and (3.193). We derive their radiation and receive functions in this section.

11.2.3.3.1 $\hat{\mathbf{n}} \times \mathcal{K}$ Operator

As the basis and testing functions are well separated, the supports of $\mathbf{f}_m$ and $\mathbf{f}_n$ do not overlap and we retain only the first term on the right-hand side in (3.192). This yields in Region R_l matrix elements given by

$$Z_{mn}^{nHJ(l)} = \text{nK}(\mathbf{f}_m, \mathbf{f}_n) \ , \tag{11.36}$$

where

$$\text{nK}(\mathbf{f}_m, \mathbf{f}_n) = -\int_{\mathbf{f}_m} \mathbf{f}_m(\mathbf{r}) \cdot \Big[\hat{\mathbf{n}}_l(\mathbf{r}) \times \int_{\mathbf{f}_n} \mathbf{f}_n(\mathbf{r}') \times \nabla' G(\mathbf{r}, \mathbf{r}') \, d\mathbf{r}'\Big] \, d\mathbf{r} \ . \tag{11.37}$$

Taking the gradient of (11.17) and rearranging the cross products yields

$$\text{nK}(\mathbf{f}_m, \mathbf{f}_n) = \int_1 \mathbf{R}^{\hat{\mathbf{n}}\times\mathcal{K}}(\mathbf{f}_m, \hat{\mathbf{k}}) \cdot T_L(k_l, \hat{\mathbf{k}}, \mathbf{r}_{ab}) \mathbf{T}^{\hat{\mathbf{n}}\times\mathcal{K}}(\mathbf{f}_n, \hat{\mathbf{k}}) \, dS \ , \tag{11.38}$$

where the radiation function for basis function $\mathbf{f}_n(\mathbf{r}')$ is

$$\mathbf{T}^{\hat{\mathbf{n}}\times\mathcal{K}}(\mathbf{f}_n, \hat{\mathbf{k}}) = \int_{\mathbf{f}_n} \mathbf{f}_n(\mathbf{r}') e^{jk_l \hat{\mathbf{k}} \cdot \mathbf{r}_{r'b}} \, d\mathbf{r}' \tag{11.39}$$

and the receive function for testing function $\mathbf{f}_m(\mathbf{r})$ is

$$\mathbf{R}^{\hat{\mathbf{n}}\times\mathcal{K}}(\mathbf{f}_m, \hat{\mathbf{k}}) = -jk_l \, \hat{\mathbf{k}} \times \int_{\mathbf{f}_m} \Big[\mathbf{f}_m(\mathbf{r}) \times \hat{\mathbf{n}}_l(\mathbf{r})\Big] e^{-jk_l \hat{\mathbf{k}} \cdot \mathbf{r}_{ra}} \, d\mathbf{r} \ . \tag{11.40}$$

We note that due to the cross product with $\hat{\mathbf{k}}$, the receive function has only $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ components.

11.2.3.3.2 $\hat{\mathbf{n}} \times \mathcal{L}$ Operator

Substituting the magnetic basis and electric testing functions into (3.193) yields in Region R_l matrix elements given by

$$Z_{mn}^{nHM(l)} = \text{nL}(\mathbf{f}_m, \mathbf{g}_n) \ , \tag{11.41}$$

where

$$\text{nL}(\mathbf{f}_m, \mathbf{g}_n) = j\omega\epsilon_l \int_{\mathbf{f}_m} \mathbf{f}_m(\mathbf{r}) \cdot \Big[\hat{\mathbf{n}}_l(\mathbf{r}) \times \int_{\mathbf{g}_n} \mathbf{g}_n(\mathbf{r}') \Big[1 - \frac{1}{k_l^2} \nabla\nabla'\Big] G(\mathbf{r}, \mathbf{r}') \, d\mathbf{r}'\Big] \, d\mathbf{r} \ . \tag{11.42}$$

Applying the differential operators yields a result similar to the $\mathcal{L}$ operator in Section 11.2.3.1, which is

$$\mathrm{nL}(\mathbf{f}_m, \mathbf{g}_n) = \int_1 \mathbf{R}^{\hat{\mathbf{n}}\times\mathcal{L}}(\mathbf{f}_m, \hat{\mathbf{k}}) \cdot T_L(k_l, \hat{\mathbf{k}}, \mathbf{r}_{ab}) \mathbf{T}^{\hat{\mathbf{n}}\times\mathcal{L}}(\mathbf{g}_n, \hat{\mathbf{k}})\, dS\,, \tag{11.43}$$

where the radiation function for basis function $\mathbf{g}_n(\mathbf{r}')$ is

$$\mathbf{T}^{\hat{\mathbf{n}}\times\mathcal{L}}(\mathbf{g}_n, \hat{\mathbf{k}}) = \left[1 - \hat{\mathbf{k}}\hat{\mathbf{k}}\right] \cdot \int_{\mathbf{g}_n} \mathbf{g}_n(\mathbf{r}') e^{jk_l \hat{\mathbf{k}}\cdot\mathbf{r}_{r'b}}\, d\mathbf{r}' \tag{11.44}$$

and the receive function for testing function $\mathbf{f}_m(\mathbf{r})$ is

$$\mathbf{R}^{\hat{\mathbf{n}}\times\mathcal{L}}(\mathbf{f}_m, \hat{\mathbf{k}}) = j\omega\epsilon_l \int_{\mathbf{f}_m} \left[\mathbf{f}_m(\mathbf{r}) \times \hat{\mathbf{n}}_l(\mathbf{r})\right] e^{-jk_l \hat{\mathbf{k}}\cdot\mathbf{r}_{ra}}\, d\mathbf{r}\,. \tag{11.45}$$

We note that the radiation function has only $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ components. An η_0 scale factor must also be applied to account for the presence of the magnetic basis function.

11.2.4 Unit Sphere Decomposition

In the previous sections we noted that for each matrix element, the radiation or receive function had only $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ components. As a result, the inner product of the radiation and receive functions operates on only those components, and any $\hat{\mathbf{k}}$-oriented components are disregarded. Therefore, when precomputing these functions, only the scalar component in each dimension is stored. For example, when precomputing the value of the receive function $\mathbf{R}^{\mathcal{L}}$ at each quadrature point on the unit sphere, one would calculate and store the scalar values $\mathrm{R}_{\theta}^{\mathcal{L}}$ and $\mathrm{R}_{\phi}^{\mathcal{L}}$, which are

$$\mathrm{R}_{\theta,\phi}^{\mathcal{L}} = (\hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}) \cdot \mathbf{R}^{\mathcal{L}} \tag{11.46}$$

and similarly, one would store the scalar values $\mathrm{T}_{\theta}^{\mathcal{L}}$ and $\mathrm{T}_{\phi}^{\mathcal{L}}$ for the radiation function $\mathbf{T}^{\mathcal{L}}$.

Though the receive functions for the various operators have different forms, a brief review of the radiation functions in Sections 11.2.3.1–11.2.3.3 shows them to be identical when decomposed this way. Thus, only one radiation function is needed for each basis function. These have the form

$$\mathbf{T}^{J}(\mathbf{f}_n, \hat{\mathbf{k}}) = \left[1 - \hat{\mathbf{k}}\hat{\mathbf{k}}\right] \cdot \int_{\mathbf{f}_n} \mathbf{f}_n(\mathbf{r}') e^{jk_l \hat{\mathbf{k}}\cdot\mathbf{r}_{r'b}}\, d\mathbf{r}' \tag{11.47}$$

for the electric basis functions, and

$$\mathbf{T}^{M}(\mathbf{g}_n, \hat{\mathbf{k}}) = \left[1 - \hat{\mathbf{k}}\hat{\mathbf{k}}\right] \cdot \int_{\mathbf{g}_n} \mathbf{g}_n(\mathbf{r}') e^{jk_l \hat{\mathbf{k}}\cdot\mathbf{r}_{r'b}}\, d\mathbf{r}' \tag{11.48}$$

for the magnetic basis functions. In general, $\mathbf{T}^J$ and $\mathbf{T}^M$ are not necessarily the same since $\mathbf{f}_n$ and $\mathbf{g}_n$ may be different, however for RWG basis functions they are identical except for the η_0 scale factor in the magnetic basis function. Thus, we can compute and store only one radiation function for each RWG function.

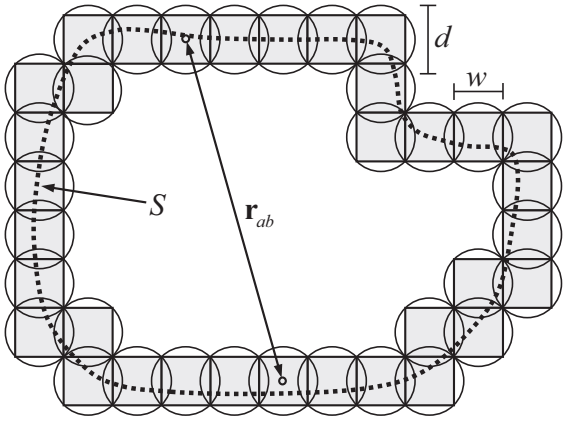


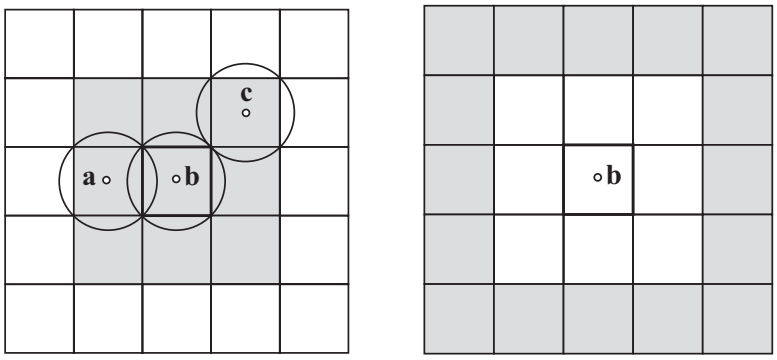
Figure 11.3: Spatial Subdivision and Grouping of Basis Functions

11.3 One-Level Fast Multipole Algorithm

We will now consider the practical implementation details of the matrix-vector product in the context of a one-level Fast Multipole Algorithm (FMA). We will discuss grouping of basis functions into near and far groups, truncation of the transfer function, sampling rates on the unit sphere, and calculation and storage of radiation and receive functions and the near matrix. In Section 11.4, we will then utilize elements of the one-level algorithm in developing the Multi-Level Fast Multipole Algorithm (MLFMA), which is the preferred approach used in virtually all MoM solvers.

11.3.1 Clustering of Basis Functions

For efficient programming and execution, a single sampling rate and number of multipoles should be used to compute and store all radiation, receive, and transfer functions. Consequently, those values will depend on the diameter of the largest group. We also note that since the transfer function depends on the vector $\mathbf{r}_{ab}$ between group pairs, a total of M^2 transfer functions would be needed, assuming M randomly centered groups. Thus, the K-means algorithm that was used to cluster basis functions for the ACA (Section 9.3) is not optimal in this case. Instead, let us enclose the entire surface S in a bounding cube, which is then divided into small, equally sized cubes with sides of length w and bounding spheres of diameter $d = \sqrt{3w^2}$. We set w to some fraction of the free-space wavelength, such as $\lambda_0/2$. Basis functions are sorted into cubes by testing each cube's bounding box against the center of the edge to which a function is assigned. When complete, the nonempty cubes and their basis functions comprise the groups, and all empty cubes are discarded, resulting in an arrangement similar to that depicted in Figure 11.3. The points **a** and **b**, left undefined in Section 11.2.3, now comprise the centers of the cubes. As a result of this regularly spaced grid, there are now a limited number of unique transfer vectors $\mathbf{r}_{ab}$, greatly reducing the number of transfer functions that must be stored.



(a) Groups Near To Group **b** (b) Groups Far From Group **b**

Figure 11.4: Near and Far Groups

11.3.1.1 Classification of Near and Far Groups

To classify pairs of groups as *near* groups or *far* groups, we must define what it means to be *well separated* in the context of the FMM. Consider two groups with bounding boxes centered at **a** and **b**. For the addition theorem of (11.8) to remain valid, the center-to-center distance $|\mathbf{r}_{ab}|$ between groups must be at least one diameter d [6]. Whether two groups are well separated is now determined by the distance between their bounding spheres. Consider the groups shown in Figure 11.4a, where the group at **b** comprises the box in the middle. Boxes sharing an edge with this one, such as the one at **a**, are separated by less than d. Those that share a corner, such as the one at **c**, are separated by exactly d. Thus, we label as *near* all boxes that share an edge or a corner. For the box at **b**, this includes itself and all the boxes it touches, which comprise the shaded boxes in Figure 11.4a. The *far* groups are those that are separated by at least one box, and comprise the shaded boxes in Figure 11.4b. The near-zone boxes that lie between the group at **b** and the far groups are often referred to as *buffer boxes*.

Following this clustering and classification strategy, the MoM matrix now comprises the sum of two sparse block matrices, $\mathbf{Z}^{near}$ and $\mathbf{Z}^{far}$. Basis functions can now be reindexed following the approach in Section 9.3.

11.3.2 Near Matrix

Because $\mathbf{Z}^{near}$ is a block matrix, it can be computed using the same software routines used to fill the ACA and MLACA block matrices, with a few modifications, as discussed in Section 11.6.1. Compression of the off-diagonal blocks is discussed next.

11.3.2.1 Compression of Near Matrix

The diagonal blocks of $\mathbf{Z}^{near}$ are not compressible, and are computed and stored in full form. However, it is possible to compress some of the off-diagonal blocks via the SVD approach (Section 9.1) and store them in compressed form. The difference is that the

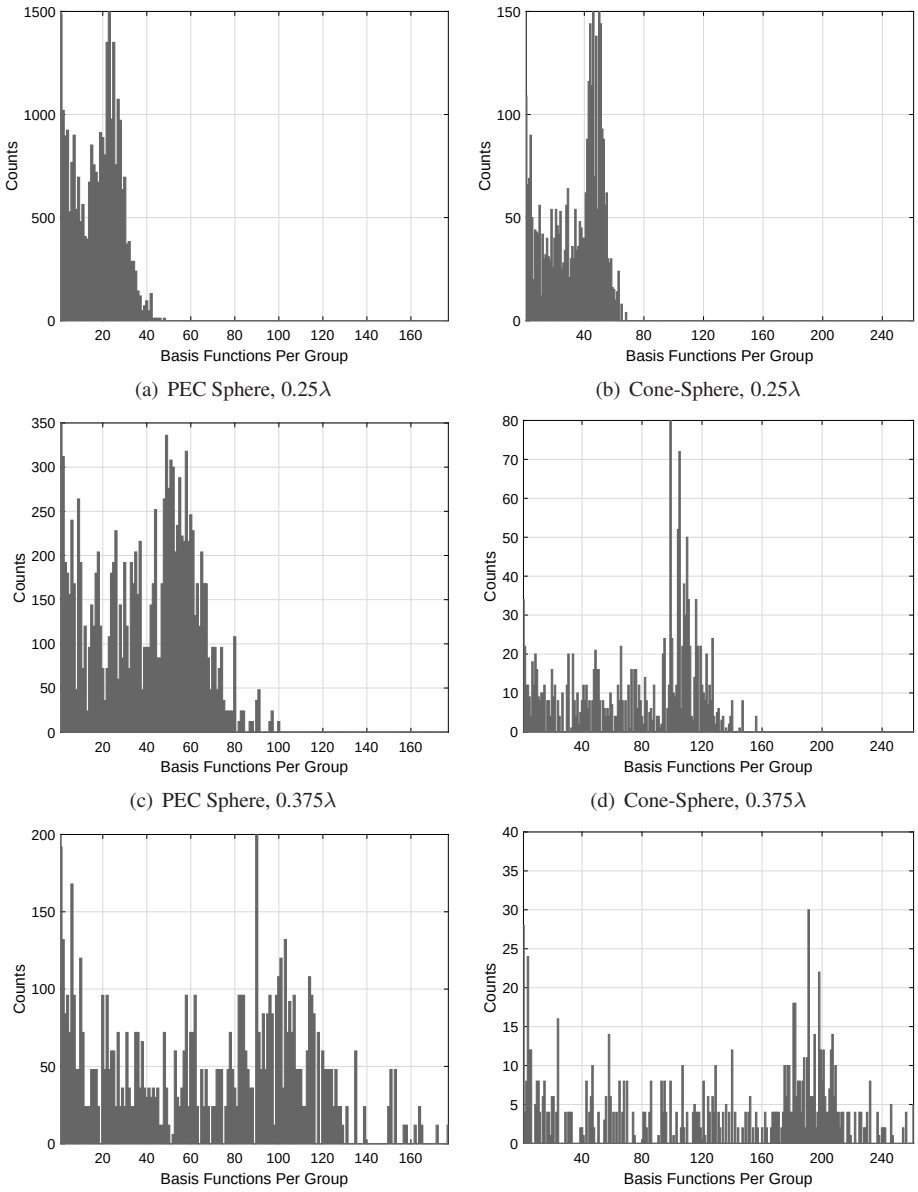
number of basis functions per group in the FMM is often no more than a few hundred, versus thousands in the ACA, and so the SVD compression is not effective for all off-diagonal blocks. That is, for some matrix blocks of size $m \times n$, $k_{SVD}(m+n) \geq mn$ for a given τ_{SVD}.

To illustrate, let us consider two examples from Chapter 9: the PEC Sphere at 6.0 GHz (Section 9.7.3.1) with 491520 basis functions, and the EMCC Cone-Sphere at 9 GHz (Section 9.7.4.3) with 119244 basis functions. Figures 11.5a–11.5f show histograms of the number of basis functions per group for cube sizes of 0.25λ, 0.375λ, and 0.5λ. We see that at 0.25λ, a large portion of the groups have less than 20 basis functions. Off-diagonal blocks involving these groups are typically not compressible. As the cube size increases, we see that the average number increases, and at 0.5λ there are many groups with over 100 basis functions. For each case, we now compute all near matrix off-diagonal blocks, and perform an SVD-based compression to determine their compressibility. In Figures 11.6a–11.6f are shown corresponding histograms of the compression of the blocks, in percent, for $\tau_{SVD} = 10^{-4}$. We note that in all cases, while there are many non-compressible blocks, this number decreases with increasing cube size, and the compressibility of individual blocks improves significantly. Table 11.1 summarizes the compression of the near matrix, taking into account the non-compressible diagonal blocks. It is small at 0.25λ, but significant gains are obtained at 0.375λ and 0.5λ. Though not as substantial as in the ACA and MLACA, which often yield over 98% compression, the near matrix size is still greatly reduced. This will be even more significant in Section 11.5, where we use the SVD to compress off-diagonal preconditioner matrix blocks, but with a less restrictive τ_{SVD}.

In practice, our approach for computing off-diagonal blocks is to fill them completely, and then compress them through direct application of the SVD. Because the blocks are typically very small, this approach is more efficient than applying the ACA+QR/SVD technique. If $k_{SVD}(m+n) < mn$ for the given τ_{SVD}, we retain the compressed version, otherwise the full version is stored.

Table 11.1: Near Matrix Percent Compression

	PEC Sphere			Cone-Sphere		
Cube Size (λ)	0.25	0.375	0.5	0.25	0.375	0.5
Compression (%)	0.3	15.6	35.6	9.4	38.3	53.5



(a) PEC Sphere, 0.25λ

(b) Cone-Sphere, 0.25λ

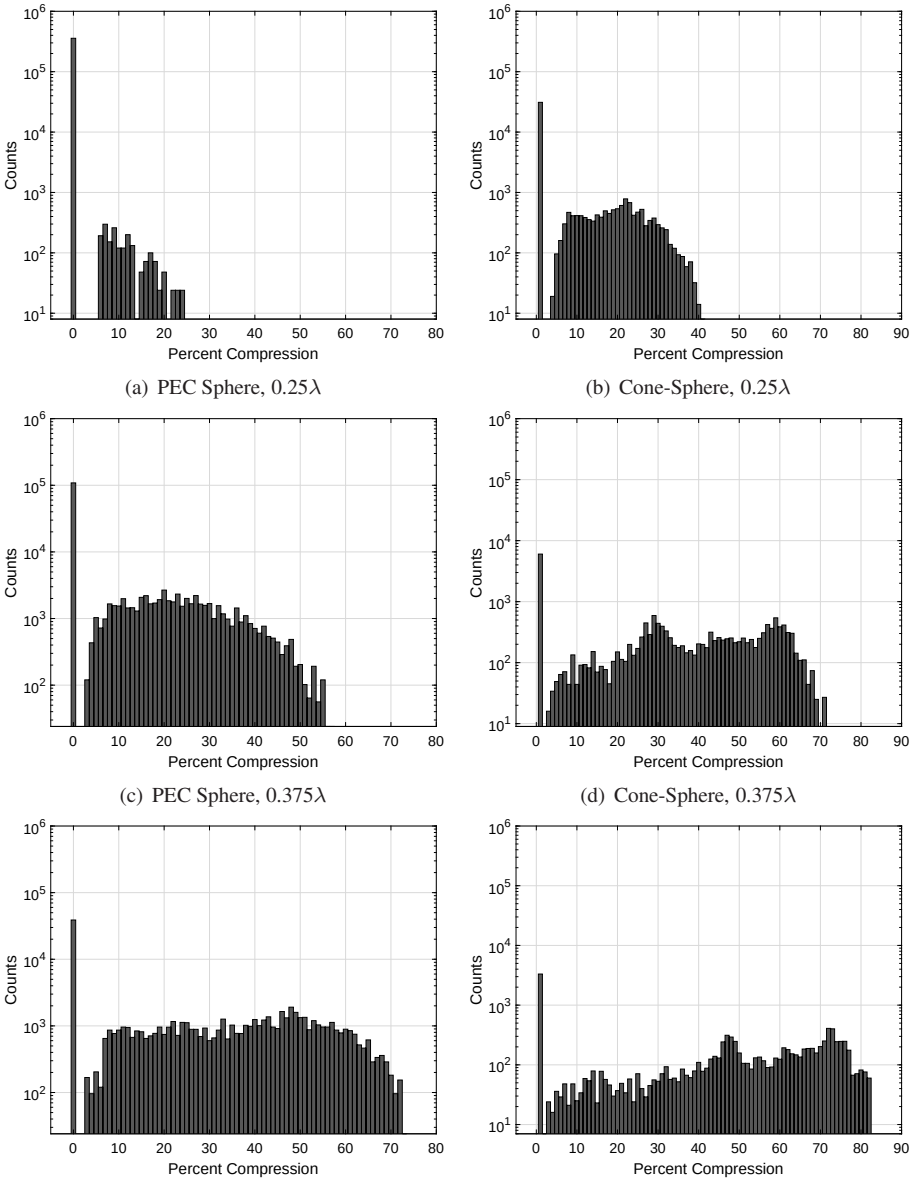
(c) PEC Sphere, 0.375λ

(d) Cone-Sphere, 0.375λ

(e) PEC Sphere, 0.5λ

(f) Cone-Sphere, 0.5λ

Figure 11.5: Basis Functions Per Group For Different Cube Sizes



(a) PEC Sphere, 0.25λ

(b) Cone-Sphere, 0.25λ

(c) PEC Sphere, 0.375λ

(d) Cone-Sphere, 0.375λ

(e) PEC Sphere, 0.5λ

(f) Cone-Sphere, 0.5λ

Figure 11.6: Compression of Off-Diagonal Blocks For Different Cube Sizes

11.3.3 Number of Multipoles

An accurate evaluation of the integral (11.12) requires truncation of the transfer function (11.13) at some limit L, which we refer to as the number of *multipoles*. This limit depends on the size of the groups and the wavenumber, and expressions for it have been determined empirically. For real-valued k, Rokhlin prescribes for single precision calculations the formula [1]

$$L = kD + 5\log(kD + \pi) \,, \tag{11.49}$$

and for double precision calculations,

$$L = kD + 10\log(kD + \pi) \,, \tag{11.50}$$

where $D = |\mathbf{x}|$ in (11.12). The maximum possible value of D to be accounted for occurs when $\mathbf{r} - \mathbf{r}'$ and $\mathbf{a} - \mathbf{b}$ are collinear. With basis functions grouped by cubes, this value is equal to the bounding-sphere diameter d, so we take $D = d$. Another expression was presented later by Chew and Song in [7] and is

$$L = kd + \beta(kd)^{1/3} \,, \tag{11.51}$$

where β is the number of digits of accuracy required. They suggest that $\beta = 6$ is sufficient for most purposes. The same formula is also used in [8] for the optimal sampling of scattered fields on the sphere.

11.3.3.1 Limiting L for Transfer Functions

Although there are many different separations $|\mathbf{r}_{ab}|$ between groups, using a single value of L from (11.51) for all $|\mathbf{r}_{ab}|$ can cause numerical issues. This is because the spherical Hankel function $h_l^{(2)}(x)$ becomes highly oscillatory for fixed x and increasing l. Therefore, we cannot take L to be much larger than the argument of the Hankel function, which is $|k_l \mathbf{r}_{ab}|$ in region R_l. Our approach is to simply limit the value of L to the minimum of either (11.51) or $|k_l \mathbf{r}_{ab}|$ in each region. Though this has the effect of reducing the accuracy of the expansion for groups having the smallest separations, this does not appear to significantly affect the results in practice.

11.3.3.2 L for Complex Wavenumbers

The expressions for L in Section 11.3.3 were derived empirically for real-valued wavenumbers. In lossy dielectric regions having complex wavenumbers, however, the number of terms required in the addition theorem increases, and (11.51) is no longer valid. When the losses are small, L increases only slightly, and the approximation

$$L = |kd| + \beta(|kd|)^{1/3} \tag{11.52}$$

is adequate in most cases. For large losses the increase in L may be much larger, and it must be determined adaptively [9].

11.3.4 Integration on the Sphere

Numerical integration of (11.19) requires the radiation, receive, and transfer functions to be computed and stored at discrete angles in the θ and ϕ dimensions on the unit sphere. We must choose the sampling rate in each dimension carefully, so that memory requirements are minimized while still meeting the Nyquist rate. As these functions are of finite (or *quasi-finite*) bandwidth [10], we need to determine the extent of the bandwidth, how it affects the sampling rate, and how the bandwidth changes when forming products of these functions.

Consider a function $f_1(x)$ having bandwidth A, sampled at a rate N_1, and a similar function $f_2(x)$ of bandwidth B sampled at a rate N_2. To then calculate the integral

$$\int_{x_1}^{x_2} f_1(x) f_2(x)\, dx \tag{11.53}$$

we can use numerical quadrature of the form

$$\int_{x_1}^{x_2} f_1(x) f_2(x)\, dx = \sum_{i=1}^{N_3} w(x_i) f_1(x_i) f_2(x_i)\ , \tag{11.54}$$

where we use a new sampling rate N_3. Because the bandwidth of the product $f_1(x)f_2(x)$ is the sum of the individual bandwidths, the sample rate N_3 must be sufficiently high to capture the bandwidth of the product. This presents a dilemma, because the sampling rate needed to numerically evaluate the Green's function in (11.19) is greater than the rate needed to sample the radiation, receive, and transfer functions individually. We can save memory by storing these functions at a lower sampling rate and then upsampling them to the higher rate when needed. However, interpolation carries with it additional compute time for upsampling the functions, as well as interpolation error. To simplify the present discussion, we will assume that all functions are sampled at the rate required for integration. We will then return to the issue of interpolation when we discuss the MLFMA in Section 11.4.

11.3.4.1 Spherical Harmonic Representation

The optimal sampling rates and numerical quadrature rules for integrating expressions like (11.19) on the unit sphere are related to their spherical harmonic representation. Therefore, let us first consider the decomposition of a bandlimited function in terms of spherical harmonics. A function $f(\theta, \phi)$ is said to be *bandlimited* with bandwidth L if it can be written as [11]

$$f(\theta, \phi) = \sum_{l=0}^{L} \sum_{m=-l}^{l} a_l^m Y_l^m(\theta, \phi)\ , \tag{11.55}$$

where a_l^m are the spherical harmonic coefficients, Y_l^m are the spherical harmonics given by

$$Y_l^m(\theta, \phi) = P_l^m(\cos\theta) e^{jm\phi}, \tag{11.56}$$

and $P_l^m(\cos\theta)$ is the normalized associated Legendre polynomial of degree l and order m, given by

$$P_l^m(x) = \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}}(1-x^2)^{m/2}\frac{d^m}{dx^m}P_l(x)\ . \tag{11.57}$$

The coefficients a_l^m are obtained via the integral

$$a_l^m = \int_0^{2\pi}\int_0^{\pi} f(\theta,\phi)Y_l^{m*}(\theta,\phi)\sin\theta\ d\theta d\phi\ , \tag{11.58}$$

and substituting (11.56) allows us to write (11.58) as

$$a_l^m = \int_0^{2\pi} e^{-jm\phi}\int_0^{\pi} f(\theta,\phi)P_l^m(\cos\theta)\sin\theta\ d\theta d\phi\ . \tag{11.59}$$

It was shown in [11] that for a function $f(\theta,\phi)$ of bandwidth L, the integral in (11.59) can be evaluated optimally using an $N_\theta = L$ point Gauss–Legendre quadrature rule in θ, and an equally spaced $N_\phi = 2L+1$ point quadrature (such as Simpson's rule) in ϕ. However, we note that the bandwidth in the ϕ dimension scales as $\sin\theta$ with a maximum at $\theta = \frac{\pi}{2}$ [10]. Thus, for each sample in θ, it is more efficient to choose $N_\phi(\theta) = \text{Int}(2L\sin\theta)+1$ samples in ϕ, where $\text{Int}(x)$ is the largest integer less than or equal to x.

11.3.4.2 Total Bandwidth

We can now determine the sample rate required to numerically integrate (11.19). Referring to Section 11.3.3, the transfer functions in region R_l have a bandwidth of

$$L_T = k_l d + \beta(k_l d)^{1/3}, \tag{11.60}$$

and similarly, the radiation and receive functions have a bandwidth equal to

$$L_f = k_l d/2 + \beta(k_l d/2)^{1/3}. \tag{11.61}$$

Thus, the total bandwidth of (11.19) is $L_{tot} = L_T + 2L_f$. Following Section 11.3.4.1, all transfer functions, as well as temporary aggregated and transmitted fields, are sampled and stored using $N_\theta = L_{tot}$ points in θ and $N_\phi(\theta) = \text{Int}(2L_{tot}\sin\theta)+1$ samples in ϕ, for a total of N_k directions. Similarly, radiation and receive functions are sampled using $N_\theta = L_f$ points in θ, and $N_\phi(\theta) = \text{Int}(2L_f\sin\theta)+1$ samples in ϕ, for a total of N_f directions. However, the radiation and receive functions are potentially compressible, as we will see in Section 11.3.4.5.

11.3.4.3 Computation and Storage of Transfer Functions

The use of axis-aligned bounding cubes results in a set of vectors whose lengths are multiples of the box edge length w in each cardinal direction. As a result, some vectors will be repeated between group pairs, and in those cases only one transfer function must be stored. Thus, transfer functions for only the unique vectors $\mathbf{r}_{ab}$ in each region are computed and stored in vectors $T_L(k_l, \mathbf{r}_{ab})$, each having length N_k. We also premultiply the transfer functions by the quadrature weights $w_s(\hat{\mathbf{k}})$ used for integration.

11.3.4.4 Computation of Radiation and Receive Functions

The radiation and receive functions in Section 11.2.3 are computed for each region separately. Inside each region, this is done on a *per-cube* basis, where we consider only basis functions having support in that region. As a result, cubes having no support in a region are considered empty and are assigned no radiation or receive functions. As a region may be assigned only a subset of all basis functions, a global-to-local basis function map is constructed, which is used to access the correct elements in the input and output vectors in the matrix-vector product.

We continue using the rules outlined in Section 8.6 to enforce the required integral equations on each interface. In general, there would be separate radiation functions for the electric and magnetic basis functions. When using RWG functions, however, only one radiation function is needed. Each testing function has two receive functions: one to receive fields radiated by electric basis functions, and one to receive fields radiated by magnetic basis functions.

For an individual cube C, the radiation and receive functions are stored in matrices $\mathbf{T}^{J,M}_{\theta,\phi}$ and $\mathbf{R}^{J,M}_{\theta,\phi}$. Each matrix has dimensions $N_f \times N_C$, where N_f is the number of sample points and N_C is the number of basis functions in C, and comprises a full- or reduced-rank outer-product matrix (see Sections 9.1 and 9.6.1.2), depending on its compressibility. As mentioned previously, $\mathbf{T}^{M}_{\theta,\phi} = \eta_l \mathbf{T}^{J}_{\theta,\phi}$ in region R_l when using RWG functions, and only $\mathbf{T}^{J}_{\theta,\phi}$ is actually stored.

11.3.4.5 Compression of Radiation and Receive Functions

We recall from Section 9.5 that when basis functions are clustered into spatially localized groups, and there are many closely-spaced RHS vectors, the RHS matrix blocks are compressible. By analogy, the radiation and receive functions comprise similar "RHS" blocks, where the RHS vectors now span the entire unit sphere. Therefore, it is reasonable to assume that these functions are potentially compressible via the SVD approach (Sections 9.1 and 11.3.2.1). We also note that because these functions represent *far* interactions, it should be possible to store them at reduced accuracy (via a larger τ_{SVD}) without significantly affecting the solution accuracy.

To illustrate, let us again consider the PEC Sphere and EMCC Cone-Sphere from Section 11.3.2.1. We again consider finest-level cube sizes of 0.25λ, 0.375λ, and 0.5λ, and for each cube, we sample the radiation and receive functions, each comprising a matrix of size $N_f \times N_C$. We then perform an SVD compression to determine their compressibility. In Figures 11.7a–11.7f are shown corresponding histograms of the compression, in percent, of the radiation and receive functions in each cube, for $\tau_{SVD} = 10^{-2}$. This τ_{SVD} was selected as it was found to deliver an excellent level of compression without any significant effects on solution accuracy. Numerical examples will demonstrate this later in Section 11.7.

As we saw when compressing the off-diagonal matrix blocks, the functions in some cubes are not compressible as they have a small number of basis functions. However, the compressibility is quite high overall and increases with cube size. Table 11.2 summarizes the total size of the radiation and receive functions (in MB), in both compressed and uncompressed form. We note that for both objects, the compressed size is

relatively flat with respect to cube size. Table 11.3 summarizes the corresponding level of compression in each case (in percent), which increases with cube size.

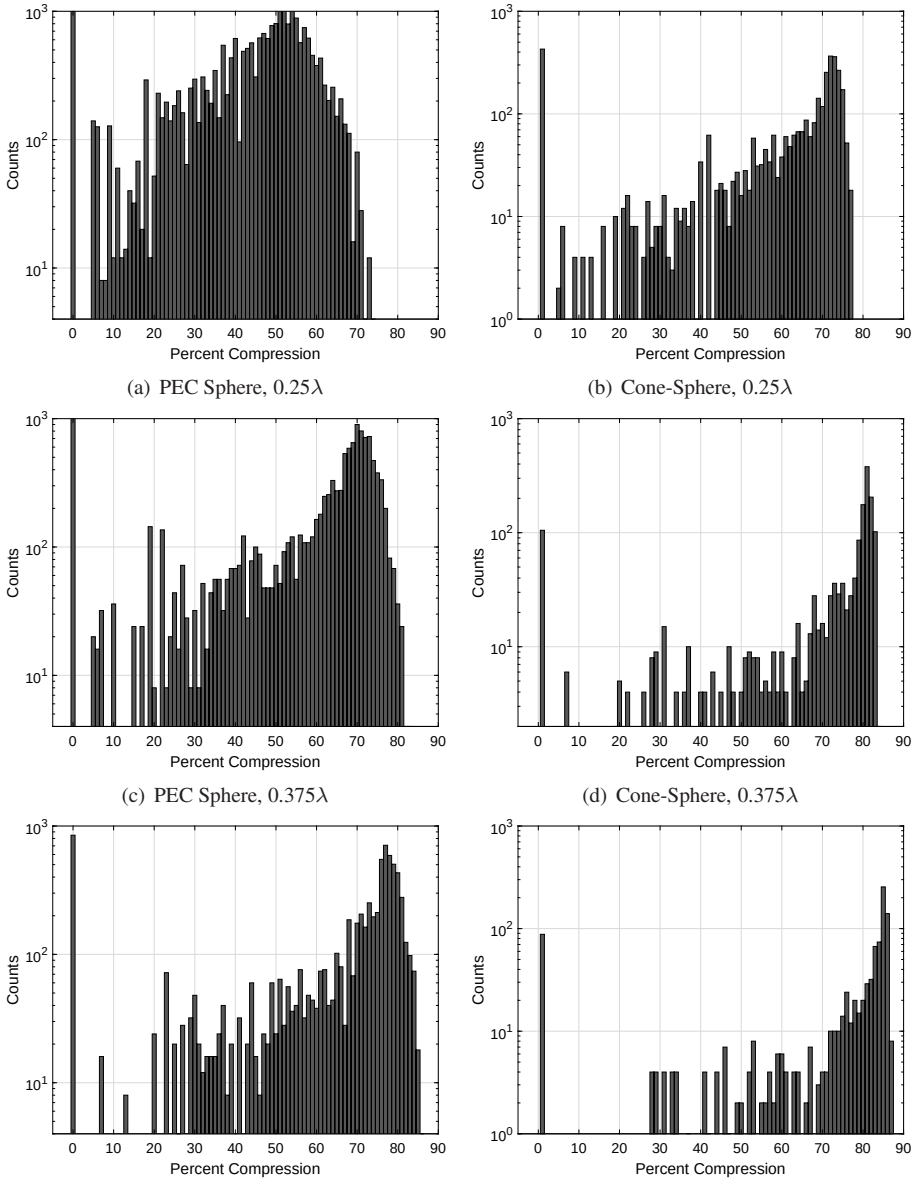
In practice, computation and storage of radiation and receive functions are similar to those of the off-diagonal matrix blocks: the matrices are filled in full and then SVD-based compression is applied. If the compression is successful, we retain the compressed version; otherwise, the full version is stored. Because the dimensions of these matrices are typically small, this step is very fast and does not significantly increase compute time.

Table 11.2: Size of Radiation and Receive Functions (MB)

	PEC Sphere			EMCC Cone-Sphere		
	0.25 λ	0.375 λ	0.5 λ	0.25 λ	0.375 λ	0.5 λ
Uncompressed	1290	1926	2362	313	404	517
Compressed	681	630	589	102	85	87

Table 11.3: Overall Compression of Radiation and Receive Functions (Percent)

PEC Sphere			EMCC Cone-Sphere		
0.25 λ	0.375 λ	0.5 λ	0.25 λ	0.375 λ	0.5 λ
73	83	87	84	90	92



(a) PEC Sphere, 0.25λ

(b) Cone-Sphere, 0.25λ

(c) PEC Sphere, 0.375λ

(d) Cone-Sphere, 0.375λ

(e) PEC Sphere, 0.5λ

(f) Cone-Sphere, 0.5λ

Figure 11.7: Compression of Radiation and Receive Functions versus Cube Size

11.3.5 Matrix-Vector Product

We now discuss computation of the matrix-vector product in (11.1) using the one-level FMM, where the result vector **x** comprises the sum of near and far products in (11.5). It was shown in [2] that the complexity of the one-level FMM is $O(N^{3/2})$.

11.3.5.1 Near Product

The near product comprises a straightforward matrix-vector product between the near matrix $\mathbf{Z}^{near}$ and the right-hand side vector **y**. As the near matrix is stored in block format, this operation is carried out most efficiently using Level 2 BLAS functions. Because fast access to individual matrix blocks is critical, their pointers should be stored in the CSR format (Section 11.6.1.2.1). As $\mathbf{Z}^{near}$ is very sparse, the time spent computing the near product is often less than the far product.

11.3.5.2 Far Product

To compute the far product, two complex vectors of length N_k are allocated for each group. These vectors, denoted as $\mathbf{t}^{\theta}$ and $\mathbf{t}^{\phi}$, store the $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ components of the aggregated radiation functions. Two additional vectors $\mathbf{s}^{\theta}$ and $\mathbf{s}^{\phi}$ of length N_k are allocated to provide temporary storage for the $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\phi}}$ components of the far fields received by a group from all far groups. The one-level far product, summarized in Algorithm 14, comprises an outer loop over all dielectric regions, where for each region R_l there are two phases: an aggregation phase and a transfer/disaggregation phase.

Step 4 comprises a mapping of the global right-hand side vector **y** of length N to a *local* vector $\mathbf{y}_{c_j}$ of length N_c, where P_{c_j} is the global to local basis function mapping for cube c_j. This is performed by simply populating $\mathbf{y}_{c_j}$ with the correct elements from **y**. In steps 14 and 15, the products $T_L(k_l, \mathbf{r}_{ab}) \odot \mathbf{t}^{\theta}_{c_k}$ and $T_L(k_l, \mathbf{r}_{ab}) \odot \mathbf{t}^{\phi}_{c_k}$ comprise term-wise multiplication. Step 18 is a mapping of the local product vector $\mathbf{x}_{c_j}$ of length N_c, to the global product vector **x** of length N. As in the aggregation step, this is performed by simply adding to **x** the correct elements from $\mathbf{x}_{c_j}$.

All operations in the aggregation phase must be completed before moving to the disaggregation phase, and in general, two passes must be executed to yield the far product. In the first pass, the fields due to only the electric basis functions are aggregated and transferred. In the second pass, the fields due to the magnetic basis functions are aggregated and transferred. During the disaggregation step, the appropriate electric or magnetic receive function, respectively, is used for each testing function. In the purely conducting case, however, only one pass is required as there are no magnetic basis functions.

Algorithm 14 One-Level FMM Far Product

for all Regions R_l **do**
  Aggregation Phase:
  for all Non-Empty Cubes c_j in R_l **do**
    $\mathbf{y}_{c_j} = \text{map}(\mathbf{y}, P_{c_j})$
    $\mathbf{t}^{\theta}_{c_j} = \mathbf{T}^{J,M}_{\theta,c_j}\mathbf{y}_{c_j}$
    $\mathbf{t}^{\phi}_{c_j} = \mathbf{T}^{J,M}_{\phi,c_j}\mathbf{y}_{c_j}$
  end for
  Disaggregation Phase:
  for all Non-Empty Cubes c_j in R_l **do**
    $\mathbf{s}^{\theta} = 0$
    $\mathbf{s}^{\phi} = 0$
    for all Non-Empty Cubes c_k far from c_j **do**
      Find the $T_L(k_l, \mathbf{r}_{ab})$ that transfers from c_k to c_j
      $\mathbf{s}^{\theta} = \mathbf{s}^{\theta} + T_L(k_l, \mathbf{r}_{ab}) \odot \mathbf{t}^{\theta}_{c_k}$
      $\mathbf{s}^{\phi} = \mathbf{s}^{\phi} + T_L(k_l, \mathbf{r}_{ab}) \odot \mathbf{t}^{\phi}_{c_k}$
    end for
    $\mathbf{x}_{c_j} = \mathbf{R}^{J,M}_{\theta}\mathbf{s}^{\theta} + \mathbf{R}^{J,M}_{\phi}\mathbf{s}^{\phi}$
    $\mathbf{x} = \mathbf{x} + \text{map}(\mathbf{x}_{c_j}, P^{-1}_{c_j})$
  end for
end for

11.4 Multi-Level Fast Multipole Algorithm (MLFMA)

In the one-level algorithm, the number of transfers grows exponentially with the size of the problem and the number of groups. Although the method offers an $O(N^{3/2})$ complexity and improved memory utilization, a complexity of $O(N \log N)$ can be obtained with a multilevel algorithm [2, 12]. The Multi-Level Fast Multipole Algorithm (MLFMA) extends the one-level algorithm by dividing the geometry recursively into multiple levels with bounding cubes of progressively smaller size. As we will see, clustering basis functions this way greatly reduces the number of transfers, further accelerating the far product. However, there are some differences and subtleties that must be handled properly for the multi-level algorithm to be effective, which we will discuss in detail in this section.

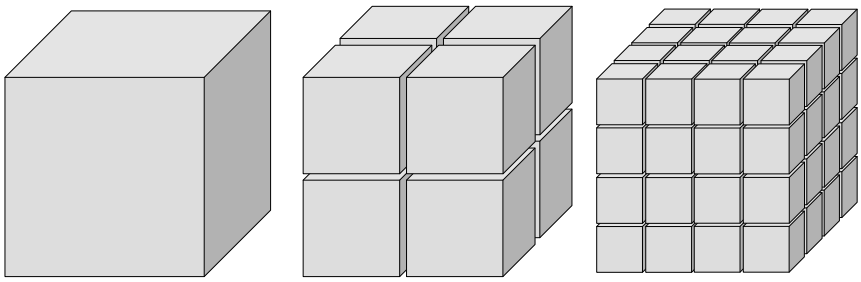
11.4.1 MLFMA/SVD

So far in this chapter, we have paid significant attention to the block near matrix and its compressibility via the SVD. We note that in the literature, though the block structure was recognized by several authors [12, 13], it does not appear that computation and storage of the near matrix was afforded special consideration. Most authors, such as those in [1], note that the near matrix is computed and stored in the "usual way", but provide no other details on how this is done. Our approach herein, where off-diagonal near matrix and preconditioner blocks are compressed via the SVD, is novel in the literature and we denote it as MLFMA/SVD.

11.4.2 Spatial Subdivision and Clustering via Octree

Let us first consider what happens when we increase the size of the cubes in the one-level algorithm. This reduces the number of transfers at the expense of increasing the size of the near matrix, shifting more of the compute time to the near product. The sampling rates on the unit sphere also increase, increasing the storage requirements of the radiation, receive, and transfer functions. If we can devise a way to simultaneously use large and small groups, we can store the radiation and receive functions at a lower sample rate, while simultaneously reducing the number of transfers. The answer to this problem lies in a recursive subdivision of the geometry, based on the cube-based clustering strategy of the one-level FMM in Section 11.3.1. Using a tree-based scheme, larger groups can be divided into recursively smaller groups with a clear parent-child relationship. An excellent choice for this is the octree [14], a data structure commonly used in computer graphics. To generate an octree, the object is first enclosed in a single large cube. This cube is then divided into eight smaller cubes, which are themselves divided into eight smaller cubes, and so on, as illustrated in Figure 11.8. This approach is very similar to the one-level algorithm, where now the basis and testing functions are sorted and assigned to cubes on each level. The assignment process is very fast as only those basis functions assigned to a parent are assigned to its children. The cubes on the finest level of the tree are called *leaf* cubes.

Our approach herein is to fix the size w of the smallest cubes at some fraction of a wavelength in R_0, such as $\lambda_0/2$ or $\lambda_0/4$. This length is then doubled until it exceeds



(a) Octree Level 1 (b) Octree Level 2 (c) Octree Level 3

Figure 11.8: Octree Levels 1–3

the maximum dimension of the object's bounding box, thus defining w for the top-level cube.

On levels 1 and 2, all groups are near groups. Starting at the next coarsest level (level 3), all near and far groups are identified as usual. On the finer levels, only the children of groups in the near zone of a parent group (see Section 11.3.1.1) are considered as candidate near and far groups for that parent's children. This greatly reduces the average number of far groups for each cube, and limits the number of unique transfer vectors to a maximum of 316 on each level. This is especially advantageous as the number of transfers can be very high in larger problems. As in the one-level algorithm, cubes that are non-empty in one region may be empty in another.

11.4.3 Near Matrix and Near Product

In the MLFMA, the nonzero part of the near matrix is defined by the near groups on the finest level of the octree. The near matrix is computed and stored in the same manner as in the one-level FMM in Section 11.3.2. The same is true for the near product, which is carried out as described in Section 11.3.5.1.

11.4.4 Unit Sphere Sampling Rates

Transfer functions are computed for all unique transfer vectors (a maximum of 316) on octree levels $l \geq 3$, where cubes have a bounding diameter d_l, and the number of multipoles L is computed via (11.51) and limited for closer groups as in Section 11.3.3.1. As the number of transfer functions per level is limited, their memory impact is relatively small. Thus, they are sampled and stored at the rate $N_{k,l}$ used for integration on each level, and do not need to be interpolated. Radiation and receive functions are computed and stored at the sample rate N_f on the finest level of the octree following Section 11.3.4.4, where the bandwidth (11.61) depends on the diameter of the leaf cubes. A consequence of this choice is that these functions must be interpolated and phase shifted when moving between levels, as discussed in the following sections. The sample rates in θ and ϕ dimensions are computed following Section 11.3.4.2, with

some oversampling in each dimension to minimize interpolation errors (see Section 11.4.6).

11.4.5 Far Product

The far product in the MLFMA operates on levels $l \geq 3$ in the octree. As in the one-level algorithm, the MLFMA comprises two phases: an aggregation phase and a transfer and disaggregation phase. However, the one-level algorithm must be extended, as the radiation and receive functions are stored on the finest level of the tree. In the aggregation phase, an upward pass is made through the octree, where radiation functions in each group are aggregated and passed up to their parents using interpolation and phase shifting. In the disaggregation phase, a downward pass is made through the octree, where transfers are made between all far groups and the received fields passed down to child groups through a combination of integration on the sphere, phase shifting, and "reverse interpolation" (anterpolation). The transfer scheme in the MLFMA is illustrated in Figures 11.9–11.11. Figure 11.9 depicts the transfers between groups on one of the coarser levels. On the next highest level in Figure 11.10, the far groups are children of groups that were considered near on the previous level. This is further emphasized on the finest level in Figure 11.11, where the number of transfers is now far less than it would have been in a one-level algorithm. We will now describe each phase in detail.

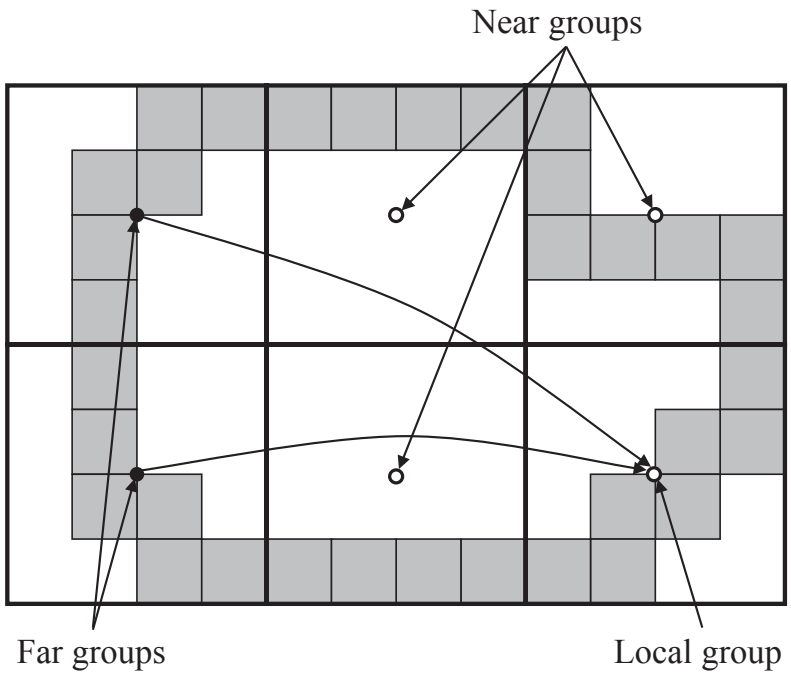


Figure 11.9: Transfers On MLFMA Level $M - 2$

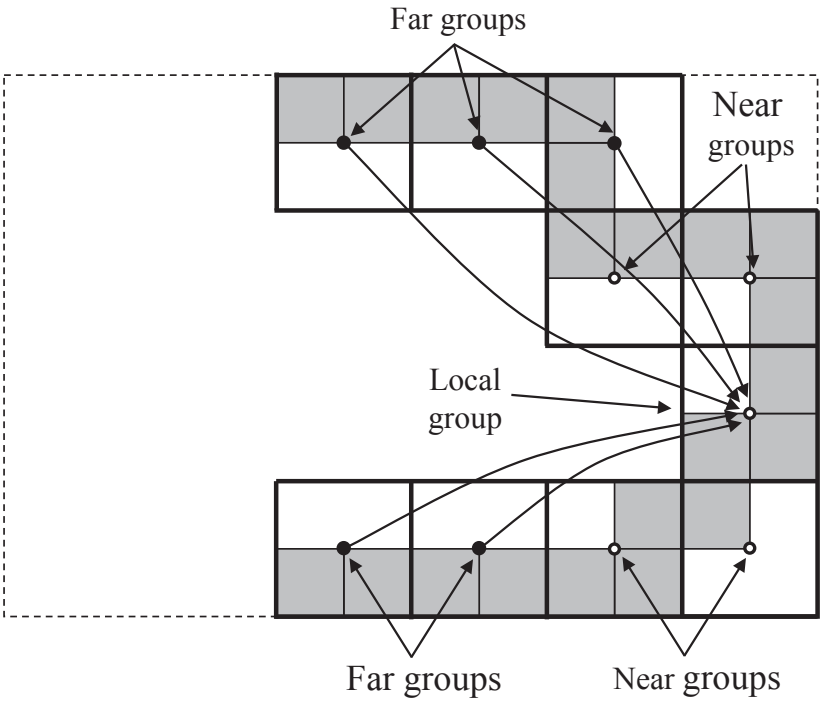


Figure 11.10: Transfers On MLFMA Level $M-1$

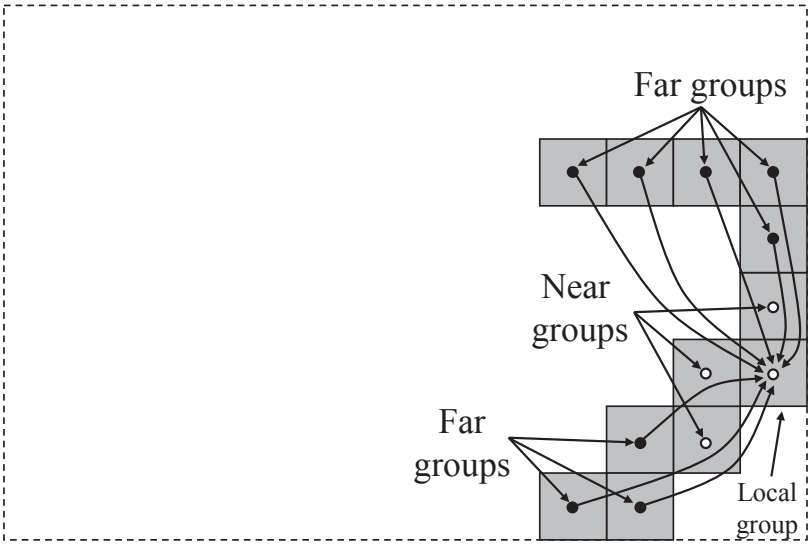


Figure 11.11: Transfers On MLFMA Level M

11.4.5.1 Upward Pass (Aggregation)

The aggregation phase comprises a recursive, upward pass through the tree. We begin on the finest level, where fields are aggregated in all leaf cubes following steps 3-7 in Algorithm 14. The aggregated field vector $\mathbf{t}_c^{\theta,\phi}$ for cube c is computed via the matrix-vector product

$$\mathbf{t}_{c,l}^{\theta,\phi} = \mathbf{W}_{N_f}^{N_{k,l}} \mathbf{T}_{\theta,\phi,c}\, \mathbf{y}_c \,, \tag{11.62}$$

where $\mathbf{W}_{N_f}^{N_{k,l}}$ is an interpolation matrix of size $N_{k,l} \times N_f$ that upsamples from the rate N_f of the radiation/receive functions to the rate $N_{k,l}$ used for integration on level l. Thus, $\mathbf{t}_{c,l}^{\theta,\phi}$, comprises a vector of length $N_{k,l}$. As the θ and ϕ component vectors are handled separately, we will defer that notation until the end of the disaggregation step and simply refer to vector $\mathbf{t}_{c,l}$.

It should be noted that the choice of upsampling to the rate $N_{k,l}$ is not arbitrary. It was found through numerical experimentation that the aggregation phase requires much less compute time than the transfer and disaggregation phase. Thus, immediately upsampling to this rate during the upward pass has little overhead, and avoids multiple interpolations at intermediate sample rates and their corresponding interpolation error.

On coarser levels, the aggregated field in each group comprises a sum of interpolated and phase-shifted aggregated fields from their children. To illustrate, consider a basis function located at $\mathbf{r}'$ and assigned to cube c, centered at $\mathbf{r}_c$. Following (11.23), its radiation function $T_l(\hat{\mathbf{k}})$ on level l can be written as

$$T_l(\hat{\mathbf{k}}) = e^{jk\hat{\mathbf{k}}\cdot(\mathbf{r}'-\mathbf{r}_c)} \,. \tag{11.63}$$

This function can be passed up to the parent cube p on level $l-1$ via a phase shift, yielding a new radiation function $T_{l-1}(\hat{\mathbf{k}})$ given by

$$T_{l-1}(\hat{\mathbf{k}}) = T_l(\hat{\mathbf{k}}) e^{jk\hat{\mathbf{k}}\cdot(\mathbf{r}_c-\mathbf{r}_p)} = T_l(\hat{\mathbf{k}})\; \psi(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}}) \tag{11.64}$$

where the phase shift function $\psi(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}})$ is

$$\psi(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}}) = e^{jk\hat{\mathbf{k}}\cdot(\mathbf{r}_c-\mathbf{r}_p)}. \tag{11.65}$$

Applying this phase shift, however, causes a bandwidth increase. Thus $\mathbf{t}_{c,l}$ must be first upsampled to the rate $N_{k,l-1}$ on level $l-1$, then phase shifted before being added to the aggregated field vector $\mathbf{t}_{p,l-1}$ of parent cube p. Doing so, can now write $\mathbf{t}_{p,l-1}$ as a sum over all nonempty child cubes, yielding

$$\mathbf{t}_{p,l-1} = \sum_c \boldsymbol{\psi}(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}}) \odot \left[\mathbf{W}_{N_{k,l}}^{N_{k,l-1}} \mathbf{t}_{c,l}\right], \tag{11.66}$$

where the phase shift in (11.65) is sampled at $N_{k,l-1}$ directions. As we use the octree in the MLFMA, there are a total of 8 child-to-parent shift vectors per level, which are precomputed and stored with little overhead.

11.4.5.2 Downward Pass (Disaggregation)

The disaggregation phase comprises a recursive, downward pass through the tree. To illustrate, let us first consider an MLFMA with only two levels: a coarse level (level $l-1$) and a fine level (level l). We begin with a single parent group p on the coarse level, and following steps 12-16 in Algorithm 14, we transfer the aggregated fields in all far groups to this one, yielding the local field vector $\mathbf{s}_{l-1}$. We now disaggregate this field with the receive functions $\mathbf{R}_{c,l}$ in a single child cube c on level l. To do so, we must first upsample and phase shift the receive functions on the finer level. Following the same approach as in the upward pass, this operation can be written as

$$\mathbf{x}_{c,l-1} = \Big[\mathbf{W}_{N_{k,l}}^{N_{k,l-1}} \mathbf{W}_{N_f}^{N_{k,l}} \mathbf{R}_{c,l}\Big]^T \big[\boldsymbol{\psi}^*(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}}) \odot \mathbf{s}_{l-1}\big], \tag{11.67}$$

where after transposition the first term on the right-hand side is a matrix of size $N_{c,l} \times N_{k,l-1}$, and we have used the conjugate of (11.65) and re-ordered terms so the phase shift is applied to $\mathbf{s}_{l-1}$. Now consider child cube c, which has its own local field $\mathbf{s}_l$ transmitted from all far groups on level l. The disaggregation step on this level can be written as

$$\mathbf{x}_{c,l} = \mathbf{R}_{c,l}^T \mathbf{s}_l \tag{11.68}$$

and the output vector $\mathbf{x}_c$ for cube c is therefore

$$\mathbf{x}_c = \mathbf{x}_{c,l-1} + \mathbf{x}_{c,l} \ . \tag{11.69}$$

It is clear that if there are many levels, repeated upsampling of the receive functions will yield a very inefficient algorithm. However, if we use the commutative property $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$, the operations in (11.67) can be reordered, yielding

$$\mathbf{x}_{c,l-1} = \mathbf{R}_{c,l}^T [\mathbf{W}_{N_f}^{N_{k,l}}]^T [\mathbf{W}_{N_{k,l}}^{N_{k,l-1}}]^T \big[\boldsymbol{\psi}^*(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}}) \odot \mathbf{s}_{l-1}\big], \tag{11.70}$$

where the matrix operations are carried out pairwise right to left. This operation still upsamples $\mathbf{R}_{c,l}^T$ to the rate $N_{k,l-1}$ on level $l-1$, but with a reversed order of operations. In the literature, this is often referred to as *adjoint interpolation* or *anterpolation* [2], though this author prefers the term *reverse interpolation* in this context.

Following (11.70), the disaggregation can now be done on level l, exclusively, where $\mathbf{x}_c$ can be written as

$$\mathbf{x}_c = \mathbf{R}_{c,l}^T \Big[\mathbf{s}_l + [\mathbf{W}_{N_f}^{N_{k,l}}]^T \mathbf{a}\Big], \tag{11.71}$$

which comprises disaggregation with the received fields $\mathbf{s}_l$ on level l and the *anterpolated* fields $\mathbf{a}$ from the parent group on level $l-1$, given by

$$\mathbf{a} = [\mathbf{W}_{N_{k,l}}^{N_{k,l-1}}]^T \big[\boldsymbol{\psi}^*(\mathbf{r}_p, \mathbf{r}_c, \hat{\mathbf{k}}) \odot \mathbf{s}_{l-1}\big]. \tag{11.72}$$

Finally, we recall from Section 11.4.5.1 that there are θ and ϕ-polarized field vectors, which are processed separately during both passes. Therefore, at the bottom of the

downward pass, there will exist four field vectors $\mathbf{s}_l^{\theta,\phi}$ and $\mathbf{a}^{\theta,\phi}$, which must be multiplied with the appropriate receive functions. Thus, (11.71) becomes

$$\mathbf{x}_c = \mathbf{R}_{\theta,c,l}^T \left[\mathbf{s}_l^{\theta} + [\mathbf{W}_{N_f}^{N_{k,l}}]^T \mathbf{a}^{\theta} \right] + \mathbf{R}_{\phi,c,l}^T \left[\mathbf{s}_l^{\phi} + [\mathbf{W}_{N_f}^{N_{k,l}}]^T \mathbf{a}^{\phi} \right]. \tag{11.73}$$

The disaggregation process can be generalized in a straightforward way to a tree with many levels. We begin on the coarsest level (level 3), and for an individual cube p, we perform transfers from all far groups, and as this is the coarsest level, there are no anterpolated fields. We then recurse into each child cube, where on the next finest level (4) we again perform transfers from all far groups. To the local field we now add the anterpolated fields from the parent group. This recursion continues until the finest level is reached, where (11.73) is evaluated. This yields a local output vector which is remapped and added to the global output vector. This process is then repeated for all other cubes on the coarsest level, completing the downward pass.

11.4.6 Interpolation Algorithms

The interpolation steps in the MLFMA comprise a significant portion of the far multiply time, particularly in the downward pass. Thus, we must make interpolation as fast as possible while minimizing the interpolation error. In this section we will discuss several different interpolation algorithms and their implementation.

11.4.6.1 Statement of the Problem

Given a function $f(\theta, \phi)$ on the unit sphere, let us compute $N_{k_1} = N_{\theta_1} \times N_{\phi_1}$ samples and store them in vector $\mathbf{s}_1$. The interpolation problem involves computing a new vector $\mathbf{s}_2$ at a different sample rate $N_{k_2} = N_{\theta_2} \times N_{\phi_2}$ using only the values from $\mathbf{s}_1$. This interpolation can be written as the matrix-vector product

$$\mathbf{s}_2 = \mathbf{W}\mathbf{s}_1, \tag{11.74}$$

where $\mathbf{W}$ is an interpolation matrix of size $N_{k_2} \times N_{k_1}$. If $N_{k_2} > N_{k_1}$, this operation *upsamples* $\mathbf{s}_1$, and if $N_{k_2} < N_{k_1}$ it *downsamples* $\mathbf{s}_1$.

Interpolation methods fall into one of two general categories: *global* or *local*. In a global interpolation, the matrix $\mathbf{W}$ is full and each sample in $\mathbf{s}_2$ is computed using all samples in $\mathbf{s}_1$. If $f(\theta, \phi)$ is bandlimited, this interpolation can be exact if it is sampled at or above the Nyquist rate. In a local interpolation, $\mathbf{W}$ is sparse and each sample in $\mathbf{s}_2$ is computed from a subset of $\mathbf{s}_1$. Local interpolations do not typically yield exact results, however they are often much faster, and their error can be controlled. In our implementation of the MLFMA, we use a local interpolation based on Lagrange polynomials, but we will also briefly consider global interpolation via spherical harmonics, which is useful in other areas of electromagnetics.

11.4.6.2 Global Interpolation by Spherical Harmonics

As we discussed in Section 11.3.4.1, if $f(\theta, \phi)$ is bandlimited, it can be decomposed into a sum of spherical harmonics (11.55) with the coefficients a_l^m defined in (11.58).

Once the coefficients are calculated, the value $f(\theta_p, \phi_p)$ can be computed as

$$f(\theta_p, \phi_p) = \sum_{l=0}^{L} \sum_{m=-l}^{l} a_l^m Y_l^m(\theta_p, \phi_p) \; . \tag{11.75}$$

If we now substitute (11.58) into (11.75), this yields

$$f(\theta_p, \phi_p) = \sum_{l=0}^{L} \sum_{m=-l}^{l} Y_l^m(\theta_p, \phi_p) \sum_{q=1}^{N_{k_1}} w_q Y_l^{m*}(\theta_q, \phi_q) f(\theta_q, \phi_q) \; , \tag{11.76}$$

where the original function $f(\theta, \phi)$ is sampled at N_{k_1} points, and w_q is the quadrature weight on the unit sphere. Exchanging the order of the summations allows us to write

$$f(\theta_p, \phi_p) = \sum_{q=1}^{N_{k_1}} w_q \sum_{l=0}^{L} \sum_{m=-l}^{l} Y_l^{m*}(\theta_p, \phi_p) Y_l^m(\theta_q, \phi_q) f(\theta_q, \phi_q) \; , \tag{11.77}$$

which can be re-written as

$$f(\theta_p, \phi_p) = \sum_{q=1}^{N_{k_1}} W_{pq} f(\theta_q, \phi_q) \; , \tag{11.78}$$

where the matrix elements W_{pq} are

$$W_{pq} = w_q \sum_{l=0}^{L} \sum_{m=-l}^{l} Y_l^{m*}(\theta_p, \phi_p) Y_l^m(\theta_q, \phi_q) \; . \tag{11.79}$$

As the matrix **W** is full, it is not optimal for interpolation in the MLFMA. For many other applications, however, it is efficient to store the coefficients and perform a reconstruction via (11.75). For example, this method is used in geopotential models of the Earth's gravitational field, allowing a fast and accurate reconstruction of the gravitational potential at any latitude or longitude. Two such models are the World Geodetic System 1984 (WGS84) [15] and the Earth Gravitational Model 1996 (EGM96) [16].

11.4.6.3 Local Interpolation by Lagrange Polynomials

Lagrange polynomials are an effective means of creating a very sparse interpolator with local support. Given N samples of a function $f(x)$, this method generates a unique polynomial of order $N-1$ that interpolates those samples. New values $f(x')$ can then be computed as

$$f(x') = \sum_{i=1}^{N} w_i(x') f(x_i) \; , \tag{11.80}$$

where the weights are

$$w_i(x') = \frac{(x'-x_1)\cdots(x'-x_{i-1})(x'-x_{i+1})\cdots(x'-x_N)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_N)} \; , \tag{11.81}$$

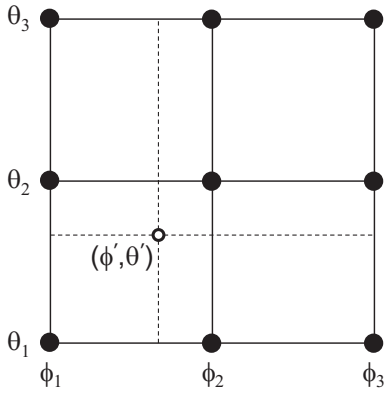


Figure 11.12: 3×3 Lagrange Interpolation Stencil

or

$$w_i(x') = \prod_{\substack{k=1 \\ k \neq i}}^{N} \frac{x' - x_k}{x_i - x_k} \,. \tag{11.82}$$

Interpolation on the unit sphere uses an $N \times N$ interpolation *stencil*, as illustrated in Figure 11.12 for $N = 3$. In this case, the interpolation matrix **W** will have nine nonzero columns per row. This yields a very sparse interpolation, as there may be hundreds (or thousands) of sample points on the unit sphere. The use of a quadratic or cubic interpolator ($N = 3$ and $N = 4$, respectively) can provide reasonable results with small error, provided that the function is oversampled. An error analysis of this method is presented in [17]. In this author's experience, a 3×3 Lagrange stencil, with an oversampling factor of 2 in the θ and ϕ dimensions, yields sufficient accuracy. A similar investigation was also performed by other authors [18]. We also note that while the oversampling of functions increases the operation count in the upward and downward pass, it does not significantly affect the storage requirements of the radiation and receive functions, as they are stored in compressed form.

11.4.6.3.1 Storage of Interpolation Matrix

As the matrix **W** is very sparse, the CSR format (Section 11.6.1.2.1) is an ideal storage container. We note that the interpolation comprises a matrix-vector product of the form $\mathbf{b} = \mathbf{Wa}$, and as the elements in **W** are in row-major format, they can be accessed linearly. This is an ideal memory access pattern. However, the elements in **a** are not accessed linearly because elements comprising the rows of the stencil are not completely contiguous in memory. Thus, cache misses may cause the CPU to stall while waiting for data from RAM. The same also occurs when performing anterpolations using $\mathbf{W}^T$. As the anterpolations in the downward pass comprise a significant portion of the compute time in the far product, special attention should be paid to optimizing the storage format and memory-access patterns for the target processing architecture [19, 20].

11.5 Preconditioners

The concept of preconditioning was introduced in Section 4.2.7. Preconditioners have several nontrivial sources of overhead, including setup time, storage, and solve time per matrix-vector product, all of which should be minimized when possible. As only the near matrix is available explicitly in the FMM, we will consider preconditioners designed specifically for sparse matrices.

As discussed previously in Section 4.2.7, given a matrix **A** the preconditioner matrix **M** should approximate **A** in some way. Our approach herein is based on the idea that as **M** is only an approximation, it does not need to be as accurate as **A** to be effective. Thus, for an MoM matrix **A** with compressed off-diagonal blocks, we will also compress the off-diagonal blocks in **M** but at reduced accuracy. This will reduce the storage requirements for **M**, and accelerate the preconditioner solve time as many blocks will be in compressed form. Numerical examples in Section 11.7 will show the effectiveness of this approach.

11.5.1 Information Content

The FMM near matrix is *very* sparse, representing only the interactions between the closest basis functions. Thus, only a limited amount of information is available for computing a preconditioner. Intuitively, we expect that as we decrease the size of groups on the finest MLFMA level, the information in the near matrix will also be reduced, making the preconditioner less effective. Therefore, the groups must be large enough so that the preconditioner remains effective, while minimizing the storage requirements of the near matrix, as well as the radiation and receive functions.

11.5.2 Diagonal Preconditioner

The diagonal or *Jacobi* preconditioner is the simplest form of preconditioner and is useful for illustrating the preconditioning concept. The elements of the diagonal preconditioner matrix **M** are

$$M_{ij} = A_{ij} \quad i = j \,, \tag{11.83}$$

and

$$M_{ij} = 0 \quad i \neq j \,. \tag{11.84}$$

We note that if we reduce the finest group size to a single basis function, the FMM near matrix now comprises only the diagonal. As noted in Section 11.5.1, the information content of the diagonal is quite small, and in practice this preconditioner is rarely effective.

11.5.3 Incomplete Block LU (ILU) Preconditioners

In this section we will consider several incomplete block LU (ILU) preconditioners. An ILU factorization of a sparse matrix **A** results in a sparse upper triangular matrix **U** and sparse lower unit triangular matrix **L**, with a residual matrix $\mathbf{R} = \mathbf{LU} - \mathbf{A}$ that satisfies certain criteria [21]. In our approach, some off-diagonal blocks of the LU

matrix are compressed using the SVD, following the block LU factorization specified in Algorithm 7. The primary differences between these preconditioners lies in the presumed zero patterns of **A** and the **LU** matrix, as well as the amount of fill-in.

11.5.3.1 Block Diagonal

The block diagonal preconditioner, previously reported in [2, 12], is constructed using only the diagonal blocks of **A**. Thus, for a 3×3 block matrix **A**, the factorization LU(**M**) is simply

$$\mathrm{LU}(\mathbf{M}) = \begin{bmatrix} \mathrm{LU}(\mathbf{M}_{11}) & 0 & 0 \\ 0 & \mathrm{LU}(\mathbf{M}_{22}) & 0 \\ 0 & 0 & \mathrm{LU}(\mathbf{M}_{33}) \end{bmatrix}, \tag{11.85}$$

where each diagonal block is factored independently. Note that this preconditioner also uses very little information from **A** and, as numerical examples will later show, is rarely effective in practice.

11.5.3.2 Block ILU with Zero Fill-In (ILU(0))

We next consider an incomplete block LU factorization with no fill-in, denoted as ILU(0), where the **LU** matrix has the same zero pattern as **A** [21]. This preconditioner can be computed via Algorithm 7, where all fill-in blocks are simply ignored. For the off-diagonal near matrix blocks we would use $\tau_{SVD} = 10^{-4}$, and for the **L** and **U** blocks we instead use $\tau_{SVD} = 10^{-3}$ or 10^{-2}. This choice is based on the idea that an *approximation* of the **LU** matrix produced by ILU(0) will perform almost as well as **LU**, while requiring less storage space.

11.5.3.3 Block ILU with Threshold (ILUT)

We can improve ILU(0) significantly by allowing fill-in blocks. The scalar approach by Saad in [21] computes the **LU** matrix by row, applying a dropping rule to the elements in each row based on their amplitude relative to the 2-norm of the row, as well as limiting the maximum number of elements per row. As Algorithm 7 operates on blocks and not individual elements, this approach cannot be applied directly, but we will use a similar strategy. First, we update all trailing row and column blocks for diagonal b. Pre-existing (non fill-in) blocks are compressed using a similar τ_{SVD} as in the ILU(0), whereas fill-in blocks are compressed using a less accurate $\tau_{SVD} = 10^{-2}$ or 10^{-1} and are added to a temporary storage space. After the update is complete, we compute the matrix norm $\|\mathbf{Z}\|_1$ for all blocks on row b and column b, including the fill-in blocks, and retain the maxima. We then drop all temporary fill-in blocks whose norm is less than a threshold τ_{ILUT} times the maximum row or column norm, where $\tau_{ILUT} = 10^{-4}$ or 10^{-3}. The remaining fill-in blocks are added to secondary CSRMatrix structures (see Section 11.6.1.2), so that they can be accessed quickly during subsequent row and column updates. As these comprise a single trailing row or column of blocks, which progressively moves to the right, they can be appended to an existing CSRMatrix

with little performance penalty. Once the factorization is complete, these blocks are combined with all other **LU** matrix blocks in a new, single CSRMatrix structure.

Our choices of τ_{SVD} and τ_{ILUT} should yield a compressed **LU** matrix that yields roughly the same improvement in iterative solver convergence as an uncompressed matrix, but with a smaller memory footprint. This will be examined and shown to be true in Section 11.7.8. It is also expected that with the added fill-in blocks, preconditioner compute time, memory requirements, and solve time will be greater than ILU(0). As we will see, our choice of τ_{SVD} for the fill-in blocks greatly minimizes the memory requirement.

11.5.4 Sparse Approximate Inverse (SAI)

Let us now consider a preconditioner matrix **M** constructed by minimizing the Frobenius norm of the residual matrix, given by

$$F(\mathbf{M}) = ||\mathbf{I} - \mathbf{AM}||_F^2 \,, \tag{11.86}$$

for right-preconditioning. In this sense, **M** is called a sparse approximate inverse (SAI). The minimization problem can be decoupled into the sum of the squares of the 2-norms of the columns of the residual matrix, which is

$$||\mathbf{I} - \mathbf{AM}||_F^2 = \sum_{n=1}^{N} ||\mathbf{e}_n - \mathbf{Am}_n||_2^2 \,, \tag{11.87}$$

where $\mathbf{e}_n$ and $\mathbf{m}_n$ are the columns of **I** and **M**, respectively. The right-hand side of (11.87) comprises N independent least-squares subproblems, which can be solved in parallel.

As the FMM near matrix is sparse, we must also determine a sparsity pattern for **M**. Some treatments in the literature find this pattern through heuristic algorithms, or iterative refinement. In some cases, the elements in **A** are first down-selected (filtered) based on certain criteria (such as amplitude), yielding a further sparsified matrix $\tilde{\mathbf{A}}$ used to build **M**. Approaches specifically tailored to the FMM problem such as those in [22, 23, 24] use the block structure of the near matrix to determine the pattern. Our approach herein assumes that **M** has the same zero pattern as **A**.

Saad [21] suggests solving for each column in (11.87) using a few iterations of a solver such as GMRES, which can be made fast by using sparse vectors and performing the matrix-vector product in a sparse-sparse mode. As the FMM near matrix is stored in block form, where some off-diagonal blocks are compressed, we will instead use a dense factorization.

11.5.4.1 Dense QR Factorization

We now consider the computation of an SAI preconditioner matrix **M** having the same the sparsity pattern as **A**. For a block matrix with M block rows and columns, this requires solving M least-square problems of the form

$$\mathbf{AM}_{*k} = \mathbf{E}_k \tag{11.88}$$

in the least-squares sense, where $\mathbf{M}_{*k}$ is block-column k of $\mathbf{M}$, and $\mathbf{E}_k$ is a block-column comprising the identity matrix in block row k and zero otherwise. To illustrate, consider a 9×9 sparse block matrix given by

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & 0 & 0 & \mathbf{A}_{15} & 0 & 0 & 0 & 0 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & 0 & \mathbf{A}_{24} & 0 & 0 & 0 & \mathbf{A}_{28} & 0 \\ 0 & 0 & \mathbf{A}_{33} & 0 & 0 & \mathbf{A}_{36} & 0 & 0 & \mathbf{A}_{39} \\ 0 & \mathbf{A}_{42} & 0 & \mathbf{A}_{44} & 0 & 0 & \mathbf{A}_{47} & 0 & 0 \\ \mathbf{A}_{51} & 0 & 0 & 0 & \mathbf{A}_{55} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{A}_{63} & 0 & 0 & \mathbf{A}_{66} & 0 & \mathbf{A}_{68} & 0 \\ 0 & 0 & 0 & \mathbf{A}_{74} & 0 & 0 & \mathbf{A}_{77} & 0 & 0 \\ 0 & \mathbf{A}_{82} & 0 & 0 & 0 & \mathbf{A}_{86} & 0 & \mathbf{A}_{88} & \mathbf{A}_{89} \\ 0 & 0 & \mathbf{A}_{93} & 0 & 0 & 0 & 0 & \mathbf{A}_{98} & \mathbf{A}_{99} \end{bmatrix}, \tag{11.89}$$

which is a simple but realistic representation of the FMM sparse near matrix. As each block column of $\mathbf{M}$ is computed independently of the others, let us compute block column $\mathbf{M}_{*4}$. In this case, the nonzero block rows of $\mathbf{M}_{*4}$ comprise rows 2, 4, and 7, which multiply block columns 2, 4, and 7 of $\mathbf{A}$ in the matrix-vector product in (11.88). Retaining only these block rows of $\mathbf{M}_{*4}$ and columns of $\mathbf{A}$ allows us to write the matrix-vector product as

$$\begin{bmatrix} \mathbf{A}_{12} & 0 & 0 \\ \mathbf{A}_{22} & \mathbf{A}_{24} & 0 \\ 0 & 0 & 0 \\ \mathbf{A}_{42} & \mathbf{A}_{44} & \mathbf{A}_{47} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \mathbf{A}_{74} & \mathbf{A}_{77} \\ \mathbf{A}_{82} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{M}_{24} \\ \mathbf{M}_{44} \\ \mathbf{M}_{74} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \mathbf{I} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{11.90}$$

We note that many of the block rows in (11.90) are zero. Removing these rows from both sides, we are left with

$$\tilde{\mathbf{A}}\tilde{\mathbf{M}} = \tilde{\mathbf{E}} = \begin{bmatrix} \mathbf{A}_{12} & 0 & 0 \\ \mathbf{A}_{22} & \mathbf{A}_{24} & 0 \\ \mathbf{A}_{42} & \mathbf{A}_{44} & \mathbf{A}_{47} \\ 0 & \mathbf{A}_{74} & \mathbf{A}_{77} \\ \mathbf{A}_{82} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{M}_{24} \\ \mathbf{M}_{44} \\ \mathbf{M}_{74} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{I} \\ 0 \\ 0 \end{bmatrix}, \tag{11.91}$$

which comprises a dense, overdetermined system. This can be solved in the least-squares sense using a QR factorization, such as the LAPACK function `cgels`. As each block column $\mathbf{M}_{*k}$ is independent, this is easily parallelized using threads.

When computing the block column $\mathbf{M}_{*k}$, it is straightforward to construct $\tilde{\mathbf{A}}$ by copying full matrix blocks or expanding compressed blocks from $\mathbf{A}$ into $\tilde{\mathbf{A}}$. After we have solved for $\tilde{\mathbf{M}}$, the corresponding blocks in $\mathbf{M}$ can be populated. Diagonal blocks are stored in full form, and an attempt is made to compress off diagonal blocks. As with the near matrix, if the compression succeeds, we retain the compressed version, otherwise the full version is stored.

11.6 Software Implementation Notes

In this section, we discuss some elements of how the MLFMA/SVD was implemented in our *Serenity* MoM solver. As with the ACA/MLACA, the techniques presented here are suggestions based on the author's experience.

11.6.1 Software Class Support

In this section we will discuss some of the software classes implemented in support of the MLFMA. Fortunately, some of the classes written previously for the ACA can be reused as-is, or with only minor modifications.

11.6.1.1 Element Engine Class

The `ElementEngine` class from Section 9.6.1.1 needs no modifications for filling near-matrix blocks, as groups constructed using the octree (FMM) or K-means (ACA) are functionally identical if using the same data structures.

11.6.1.2 Sparse Block Matrix Class

The `BlockMatrix` class described in Section 9.6.1.2 is not optimal for storing the FMM near matrix. In the ACA, the groups are very large and may number several hundred to a thousand, whereas in the FMM the groups are much smaller and there may be tens (or hundreds) of thousands of groups. Thus, for M groups, a row-major array of size $M \times M$ is not a viable approach for storing pointers to the near matrix blocks.

To solve this problem, we derive a new class from `BlockMatrix`. The new `SparseBlockMatrix` class stores block pointers in two different ways. During matrix filling, pointers are stored in a `std::map` C++ STL container using the block row sr and column sc as the key. When filling is complete, a new structure is built that facilitates fast access to blocks via `getBlock(sr, sc)`, because `std::map` is less efficient in that role. This structure is the Compressed Sparse Row (CSR) format, described in the following section.

11.6.1.2.1 Compressed Sparse Row Format

The Compressed Sparse Row (CSR) format [21] is a popular and efficient way to store any sort of sparse matrix data. To illustrate, consider the sparse integer matrix $\mathbf{A}$ defined as

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 9 & 10 & 0 \\ 0 & 0 & 0 & 0 & 11 \end{bmatrix}, \tag{11.92}$$

which is a 5×5 matrix with $N_z = 11$ nonzero entries. The CSR representation for **A** comprises three vectors **a**, **j**, and **i**, which in this case are

$$\mathbf{a} = \big[1\,2\,3\,4\,5\,6\,7\,8\,9\,10\,11\big], \tag{11.93}$$

$$\mathbf{j} = \big[1\,2\,1\,2\,3\,2\,3\,4\,3\,4\,5\big], \tag{11.94}$$

and

$$\mathbf{i} = \big[1\,3\,6\,9\,11\,12\big]. \tag{11.95}$$

Vector **a** is of length N_z and contains the nonzero elements of **A**, stored row by row. Vector **j** is also of length N_z and contains the column indices of the matrix elements in **a**. Vector **i** has length $N + 1$ and stores the offsets to each row in **a** and **j**, allowing fast access to the start and end of each row. The last element of **i** contains an offset to the fictitious row $m = N + 1$, allowing a fast lookup of the number of nonzero columns for each row. Note that as the rows and columns are sorted, we can quickly find matrix element $\mathbf{A}_{mn}$ by performing a binary search for column n within positions $\mathbf{i}(m)$ to $\mathbf{i}(m+1) - 1$ in vector **j**.

11.6.1.3 FMM Region Class

The `FMMRegion` class is responsible for computing and storing all radiation, receive and transfer functions in a dielectric region. It also implements routines that perform the upward and downward passes in the MLFMA and maps between global and local basis-function vectors.

11.6.1.4 FMM Octree Class

The `FMMOctree` class groups basis functions recursively using the octree, and provides this data to the `ElementEngine` class for computing the near matrix blocks and to the `FMMRegion` class for computing radiation and receive functions for each group/cube. It also finds all near and far groups and unique transfer vectors on each octree level, information used by the `FMMRegion` class in performing the upward and downward passes.

11.6.2 Shared Memory Processing

Our implementation in *Serenity* is for shared memory systems, performing all operations in parallel on the CPU using POSIX Threads. For all level 2 BLAS functions and the SVD, the CPU-optimized Intel MKL BLAS and LAPACK libraries are used.

11.6.2.1 FMM CPU Thread Class

The `FMMCPUThread` class performs a variety of setup tasks as well as the matrix solution in parallel on the CPU. These operations are described in more detail in this section.

11.6.2.1.1 Near Matrix Filling

When filling the near matrix, each thread computes matrix blocks asynchronously from other threads, choosing from near matrix blocks that are not yet filled.

11.6.2.1.2 Radiation, Receive and Transfer Functions

Computation of the transfer, radiation, and receive functions in each region is parallelized by assigning unique transfer vectors or cubes to threads.

11.6.2.1.3 Preconditioner Matrix

Construction of the ILU(0) or ILUT preconditioner follows Section 11.5.3, and repurposes the code written for the ACA LU factorization described in Section 9.6.2.1. Key differences include the additional code needed to handle the fill-in blocks, which are not present in the ACA block matrix. Construction of the SAI preconditioner follows the dense QR factorization outlined in Section 11.5.4.1, where each thread computes block columns of $\mathbf{M}$ asynchronously from other threads, choosing from columns not yet processed.

11.6.2.1.4 Parallelization of Matrix-Vector Product

We have identified two strategies for parallelizing the matrix-vector product, as discussed in this section.

In the first method, each CPU thread is assigned a small, contiguous portion of all right-hand sides, which they solve asynchronously from other threads. In radar cross section problems with many incident angles, this approach is more efficient as the angles in each portion are often closely spaced, and the reuse of solution vectors from previous angles is more effective in reducing iteration count. However, this approach requires a separate MLFMA workspace for each CPU thread, increasing the total memory requirements. Additionally, iteration count is not always the same between angles, so some threads often finish their portion before the rest and go idle.

In the second method, all threads participate in the solution of each RHS, with each thread performing some portion of the calculation. Thus, thread synchronization is kept to a minimum. The near product is easily done in parallel by assignment of the block rows of $\mathbf{Z}^{near}$ to individual threads. We next consider the upward and downward passes. We start both passes on the coarsest octree level ($l = 3$), and note that the computations for each cube, and their children, can be done independently of other cubes if the upward pass is completed prior to starting the downward pass. Therefore, both passes can be performed in parallel asynchronously, with each thread starting on the coarsest level and selecting from cubes not already processed for that pass. A barrier is needed at the end of the upward pass to ensure that the aggregated fields across all levels have been computed. We note that on level $l = 3$ there are at most 64 cubes, and therefore this second method does not scale past 64 threads.

11.6.2.1.5 MLFMA Workspace Management

For the upward pass (aggregation), two complex vectors of length $N_{k,l}$ are required for each group on levels $l \geq 3$ to store the θ- and ϕ-polarized aggregated fields. For the downward pass (disaggregation), two vectors of length $N_{k,l}$ are required for each level $l \geq 3$ for storing the θ- and ϕ-polarized sum of anterpolated/received fields along the recursive path.

11.7 Numerical Examples

In this section we will compute the radar cross section of several test articles using our *Serenity* MoM solver, which implements the MLFMA using the techniques outlined in this chapter. We again consider conducting, dielectric, and coated spheres, EMCC benchmark targets, and the monoconic RV with dielectric nose, again with larger electrical sizes than in prior chapters. We will also analyze the performance of the preconditioners presented in Section 11.5.

11.7.1 Compute Platform

Computations in this section were carried out on the Dell workstation previously described in Section 8.8.2.

11.7.2 Run Parameters

All triangle meshes were constructed with at least 10 edges per wavelength in each dielectric region. The CFIE is applied on closed conductors with $\alpha = 0.5$, and PMCHWT applied to dielectric interfaces. For the examples in Sections 11.7.3–11.7.6, the finest-level cube size in the octree is $\lambda_0/2$. For SVD compression of radiation and receive functions, $\tau_{SVD} = 10^{-2}$, and for the near matrix, $\tau_{SVD} = 10^{-4}$. For each case an ILUT preconditioner is built, with $\tau_{SVD} = 10^{-3}$ for existing off-diagonal blocks, $\tau_{SVD} = 10^{-1}$ for fill-in blocks, and a block drop tolerance $\tau_{ILUT} = 10^{-4}$. For the iterative solution, GMRES iteration is used, with a residual tolerance $\epsilon = 10^{-3}$.

11.7.3 Spheres

In this section we will consider the bistatic RCS of conducting, fully dielectric, and coated spheres, having the same dimensions as the spheres in Section 8.8.3.

11.7.3.1 Conducting Sphere

We first consider a conducting sphere with a radius of 0.5 meters at frequencies of 0.75, 1.5, 3.0, 6.0, 12.0 and 24.0 GHz. For the MoM simulation, we use facet models comprising 5120, 20480, 81920, 327680, 1310720, and 5242880 triangles, resulting in 7680, 30720, 122880, 491520, 1966080, and 7864320 unknowns, respectively. The bistatic RCS from the MLFMA is compared to the Mie series in Figures 11.14a–11.14f and 11.15a–11.15d. The comparison is very good at all frequencies.

11.7.3.2 Dielectric Sphere

We next consider a lossless dielectric sphere ($\epsilon_r = 2.56$) with a radius of 0.5 meters at frequencies of 3.0, 6.0, and 12.0 GHz. For the MoM simulation, we re-use the facet models from Section 11.7.3.1, however there are now an equal number of magnetic basis functions, resulting in 245760, 983040 and 3932160 basis functions at 3.0, 6.0 and 12.0 GHz, respectively. The bistatic RCS from the MLFMA is compared to the Mie series in Figures 11.16a–11.16d, where the comparison is very good. At 12.0 GHz, there is a slight disagreement between 30 and 60 degrees in the VV-pol data, which is not seen in the HH-pol results. The cause of the discrepancy is not known.

11.7.3.3 Coated Sphere

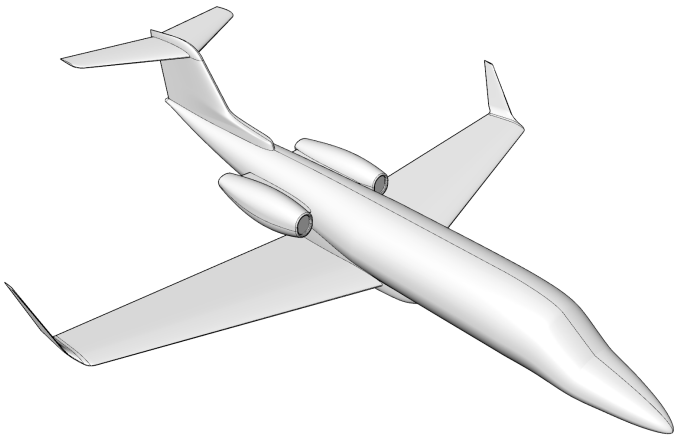
We next consider the lossless coated sphere ($\epsilon_r = 2.56$) at 3.0, 6.0 and 12.0 GHz. At 3.0 GHz, the surface meshes for the conducting core and the coating comprise 20480 and 81920 triangles, resulting in 153600 electric and 122880 magnetic basis functions, respectively, for a total of 276480 unknowns. At 6.0, they comprise 81920 and 327680 triangles, yielding 614400 electric and 491520 magnetic basis functions, for a total of 1105920 unknowns. Finally, at 12.0 GHz, they comprise 327680 and 1310720 triangles, yielding 2457600 electric and 1966080 magnetic basis functions, for a total of 4423680 unknowns. The bistatic RCS from the FMM is compared to the Mie series in Figures 11.17a–11.17d, where the comparison is again very good at all frequencies.

11.7.4 Thin Square Plate

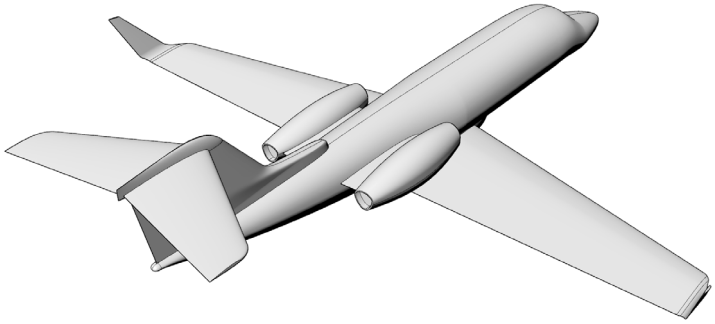
We next consider a thin, square conducting plate having a length and width of 1 meter at 1.0, 2.0, 4.0, 8.0, and 16.0 GHz. For the MoM simulation, facet models with 12288, 49152, 196608, 786432, and 3145728 facets are used, resulting in 18560, 73984, 294400, 1178624, and 4716544 unknowns, respectively. The results from the MLFMA are compared to those from the single-level ACA for horizontal polarization in Figures 11.18a–11.18e, where the results are nearly indistinguishable.

11.7.5 Monoconic Reentry Vehicle

Next, we will examine the monoconic RV with lossless dielectric nose, previously considered in Section 10.4.4, and we will compare the results from the MLFMA with those from the MLACA ($L = 3$). We compute the monostatic RCS at 14.0, 16.0, 20.0, and 24.0 GHz, using facet models having 1065932, 1167268, 2206972, and 3252068 triangles, respectively. These comprise the total number of facets used for the nose, base and septum interfaces, resulting in a total of 1645768, 1799000, 3403364, and 5022318 unknowns, respectively. The results are compared for vertical and horizontal polarizations in Figures 11.19a–11.19h, where the comparison is very good at all frequencies.



(a) Front View



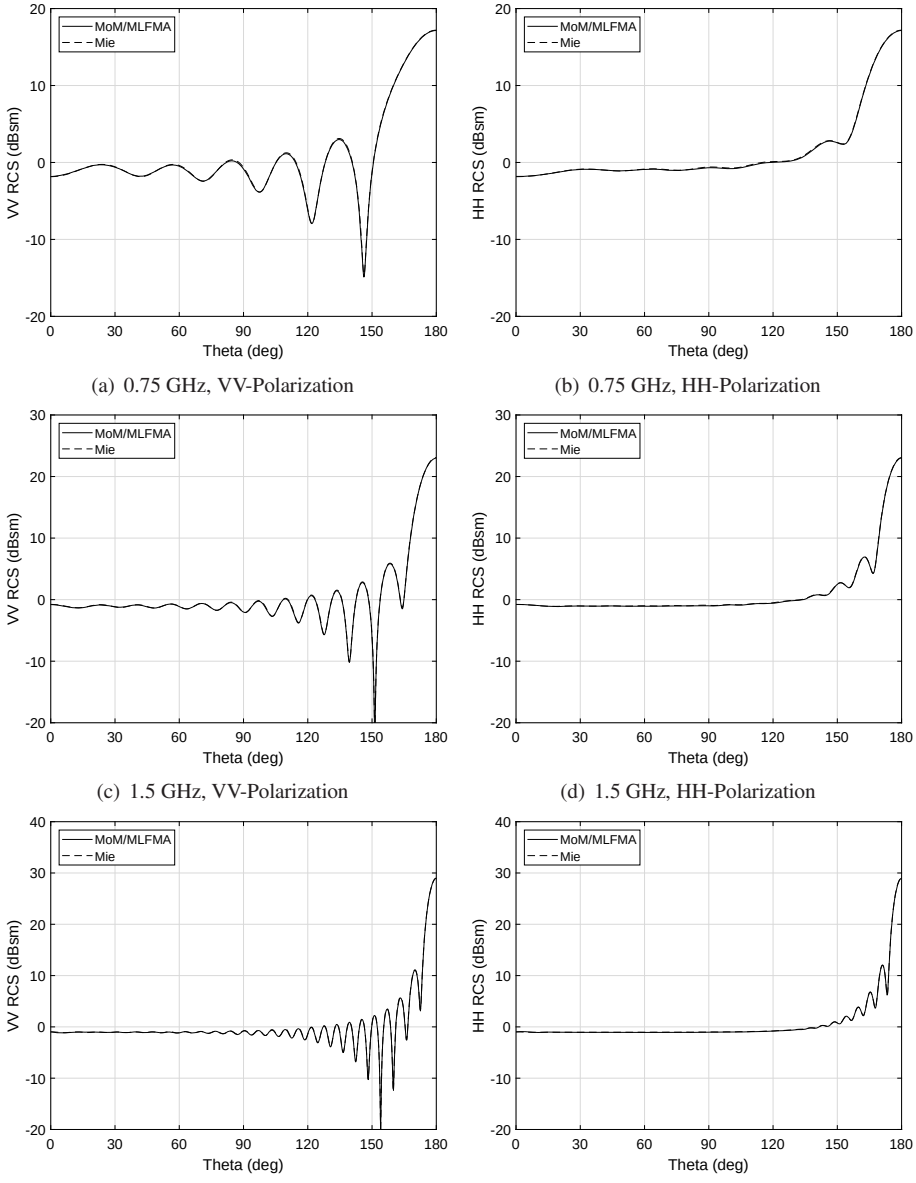
(b) Rear View

Figure 11.13: Business Jet

11.7.6 Business Jet

Finally, we consider a business jet, depicted in Figures 11.13a and 11.13b from front and rear viewing angles, respectively. The jet has a length of 17.6 meters (60 feet), a wingspan of 16.2 meters (53 feet), and a height of 3.8 meters (12 feet). The original model was obtained from an Internet database of freely available 3D models. To simplify the meshing process, the model was converted to a closed, conducting surface by removing features such as the windows and landing gear, and by closing off the engine nacelles. The model was then oriented in the xy plane, with the nose aligned along the $+x$ axis.

We compute the monostatic RCS in the xy plane for azimuthal angles ϕ from 0 to 180 degrees, at frequencies of 0.6, 1.2, 1.8 and 2.4 GHz. In support, facet models were constructed with 430392, 1606124, 3555630, and 6039472 triangles, resulting in 645588, 2409186, 5333445 and 9059208 unknowns, respectively. The results from the MLFMA are compared to those from the MLACA ($L = 3$) for vertical and horizontal polarizations in Figures 11.20a–11.20h, where the results are almost indistinguishable.



(a) 0.75 GHz, VV-Polarization

(b) 0.75 GHz, HH-Polarization

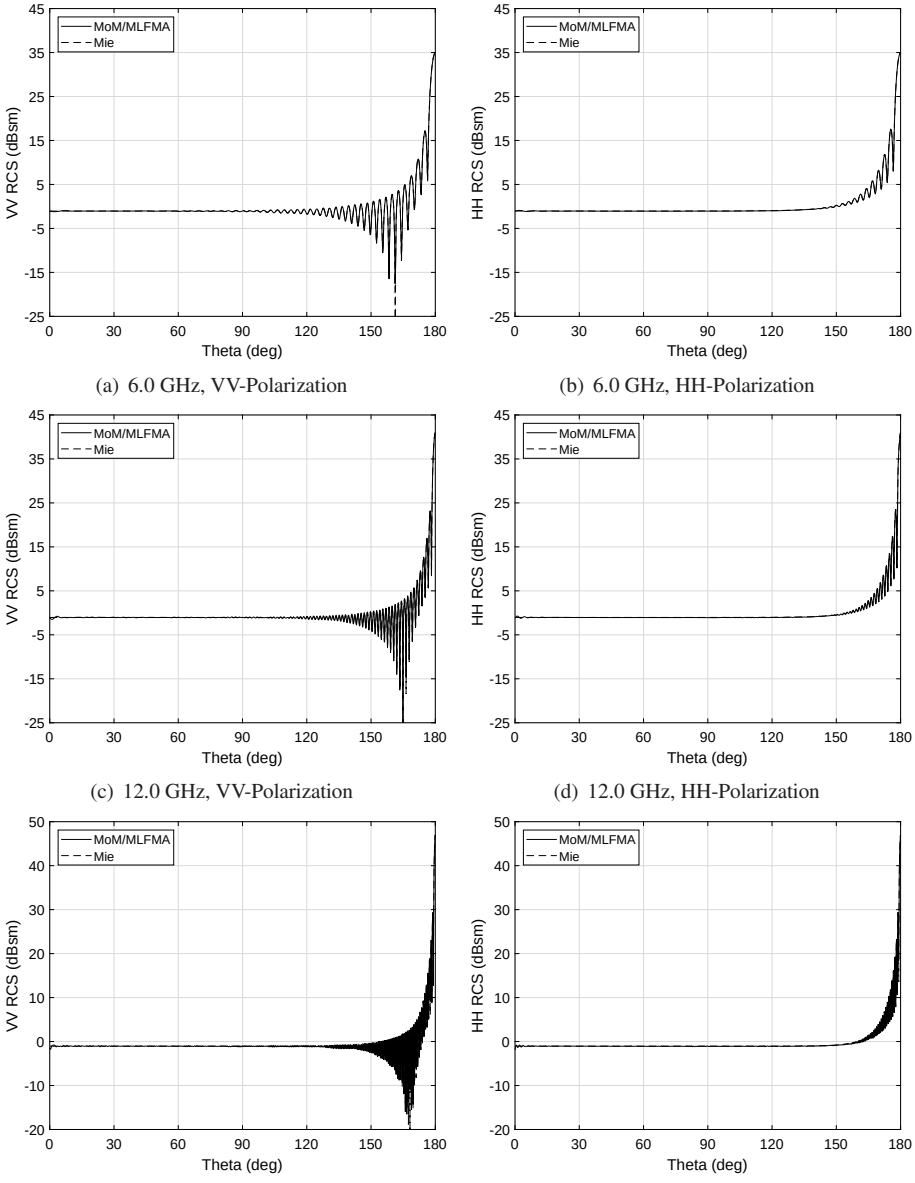
(c) 1.5 GHz, VV-Polarization

(d) 1.5 GHz, HH-Polarization

(e) 3.0 GHz, VV-Polarization

(f) 3.0 GHz, HH-Polarization

Figure 11.14: Conducting Sphere: Bistatic RCS (0.75–3.0 GHz)



(a) 6.0 GHz, VV-Polarization

(b) 6.0 GHz, HH-Polarization

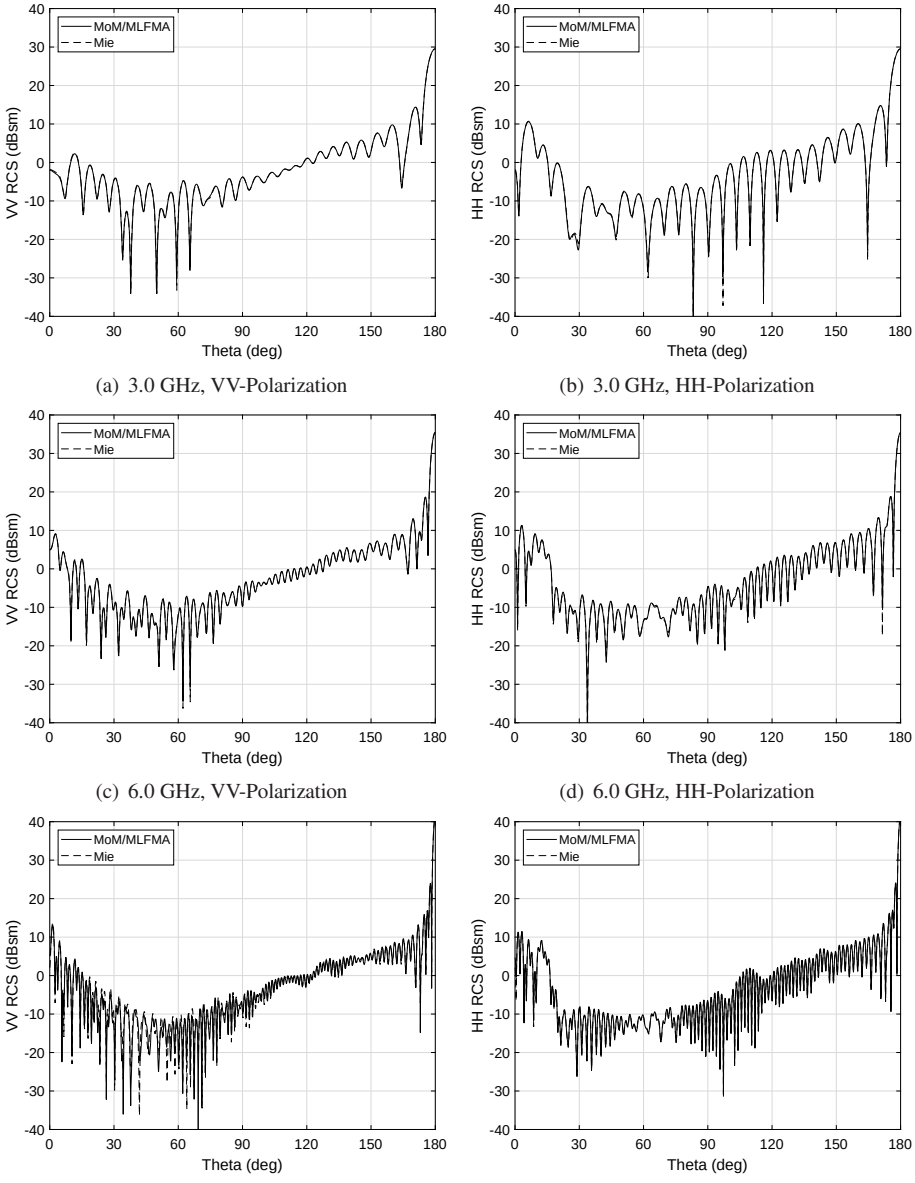
(c) 12.0 GHz, VV-Polarization

(d) 12.0 GHz, HH-Polarization

(e) 24.0 GHz, VV-Polarization

(f) 24.0 GHz, HH-Polarization

Figure 11.15: Conducting Sphere: Bistatic RCS (6.0–24.0 GHz)



(a) 3.0 GHz, VV-Polarization

(b) 3.0 GHz, HH-Polarization

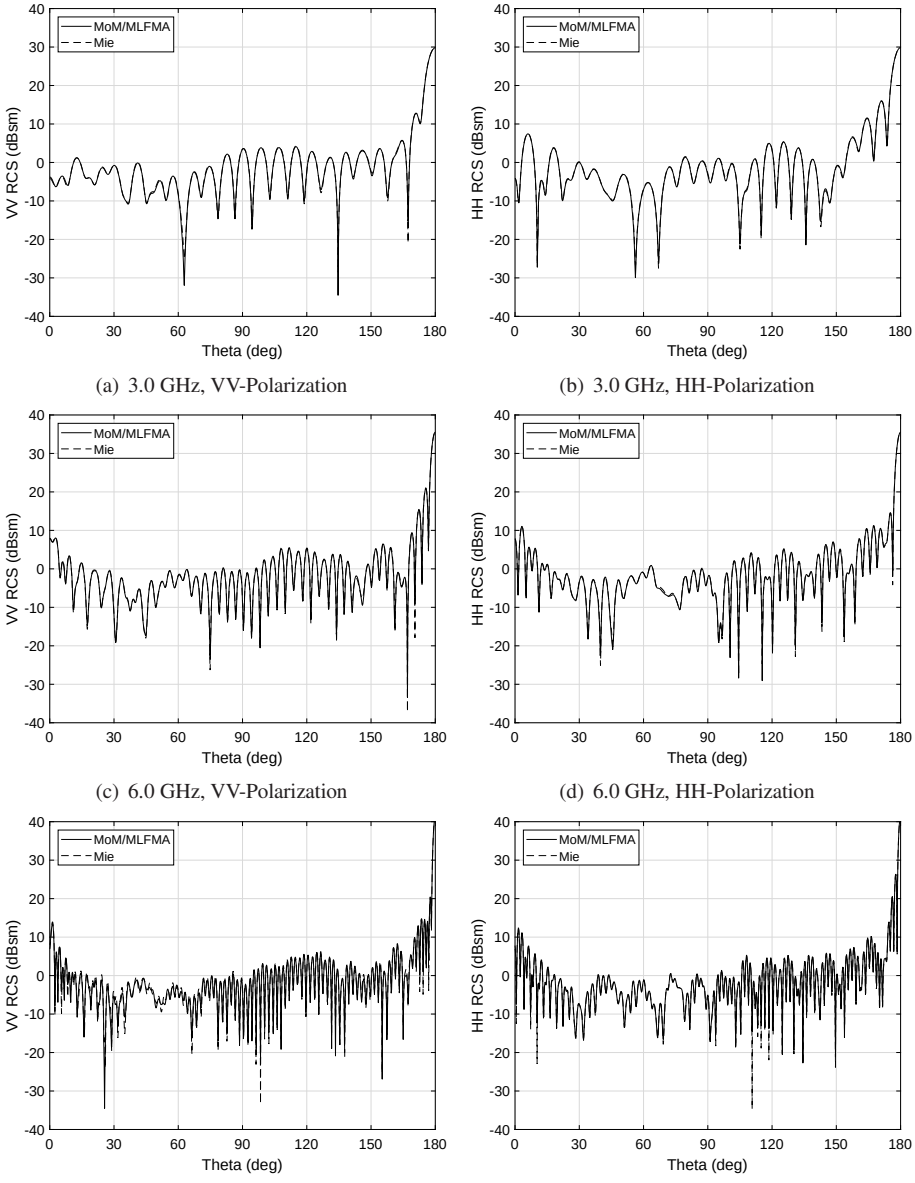
(c) 6.0 GHz, VV-Polarization

(d) 6.0 GHz, HH-Polarization

(e) 12.0 GHz, VV-Polarization

(f) 12.0 GHz, HH-Polarization

Figure 11.16: Dielectric Sphere: Bistatic RCS



(a) 3.0 GHz, VV-Polarization

(b) 3.0 GHz, HH-Polarization

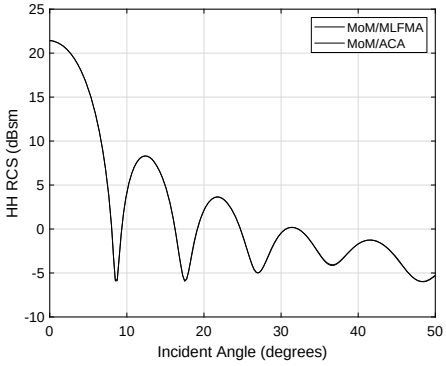
(c) 6.0 GHz, VV-Polarization

(d) 6.0 GHz, HH-Polarization

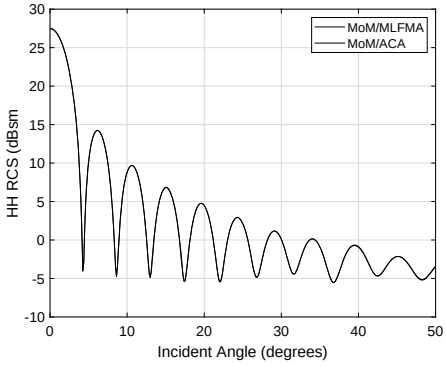
(e) 12.0 GHz, VV-Polarization

(f) 12.0 GHz, HH-Polarization

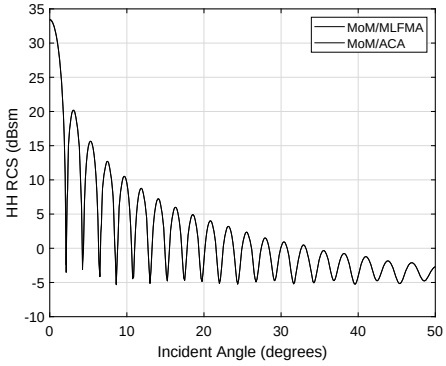
Figure 11.17: Coated Sphere: Bistatic RCS



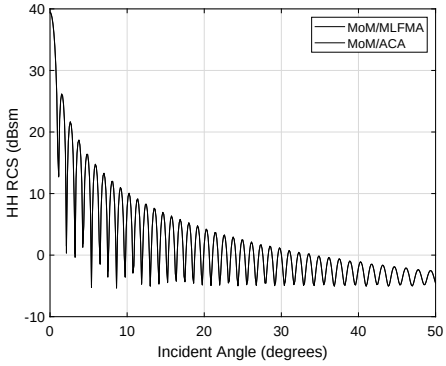
(a) 1.0 GHz, HH-Polarization



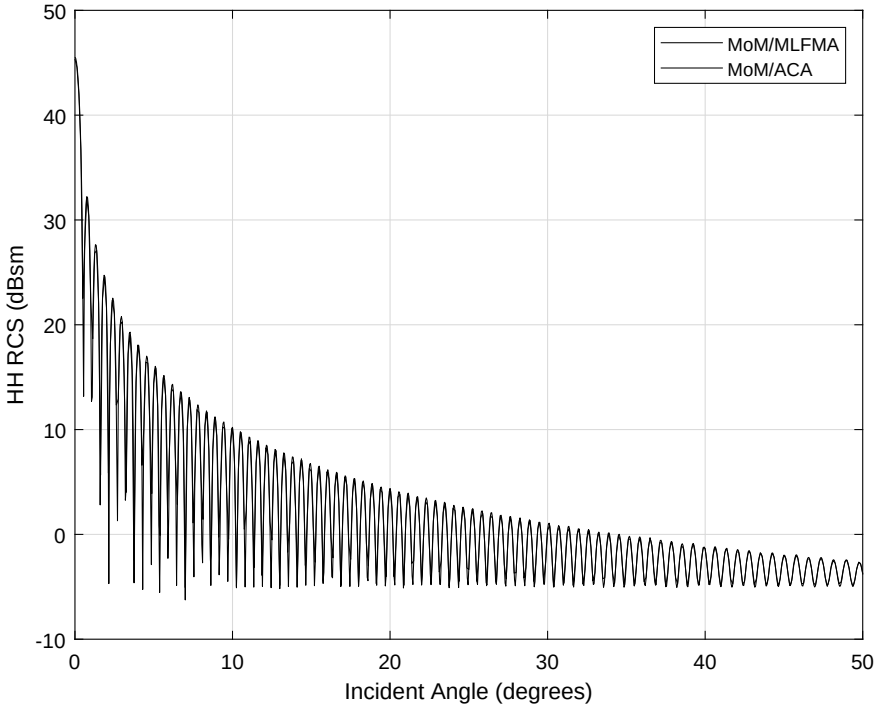
(b) 2.0 GHz, HH-Polarization



(c) 4.0 GHz, HH-Polarization

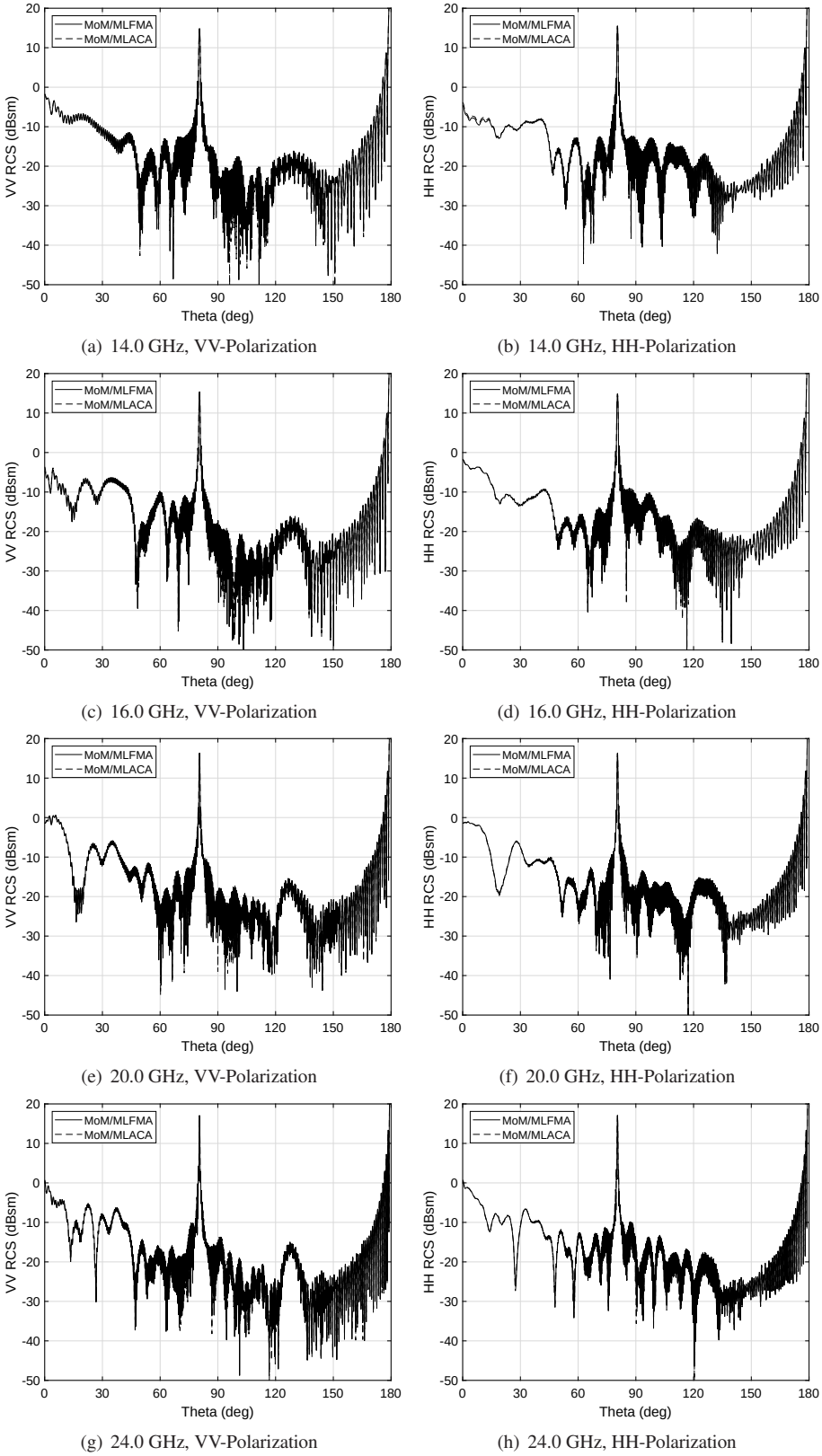


(d) 8.0 GHz, HH-Polarization



(e) 16.0 GHz, HH-Polarization

Figure 11.18: Thin Plate: HH RCS at 1.0, 2.0, 4.0, 8.0, and 16.0 GHz



(a) 14.0 GHz, VV-Polarization

(b) 14.0 GHz, HH-Polarization

(c) 16.0 GHz, VV-Polarization

(d) 16.0 GHz, HH-Polarization

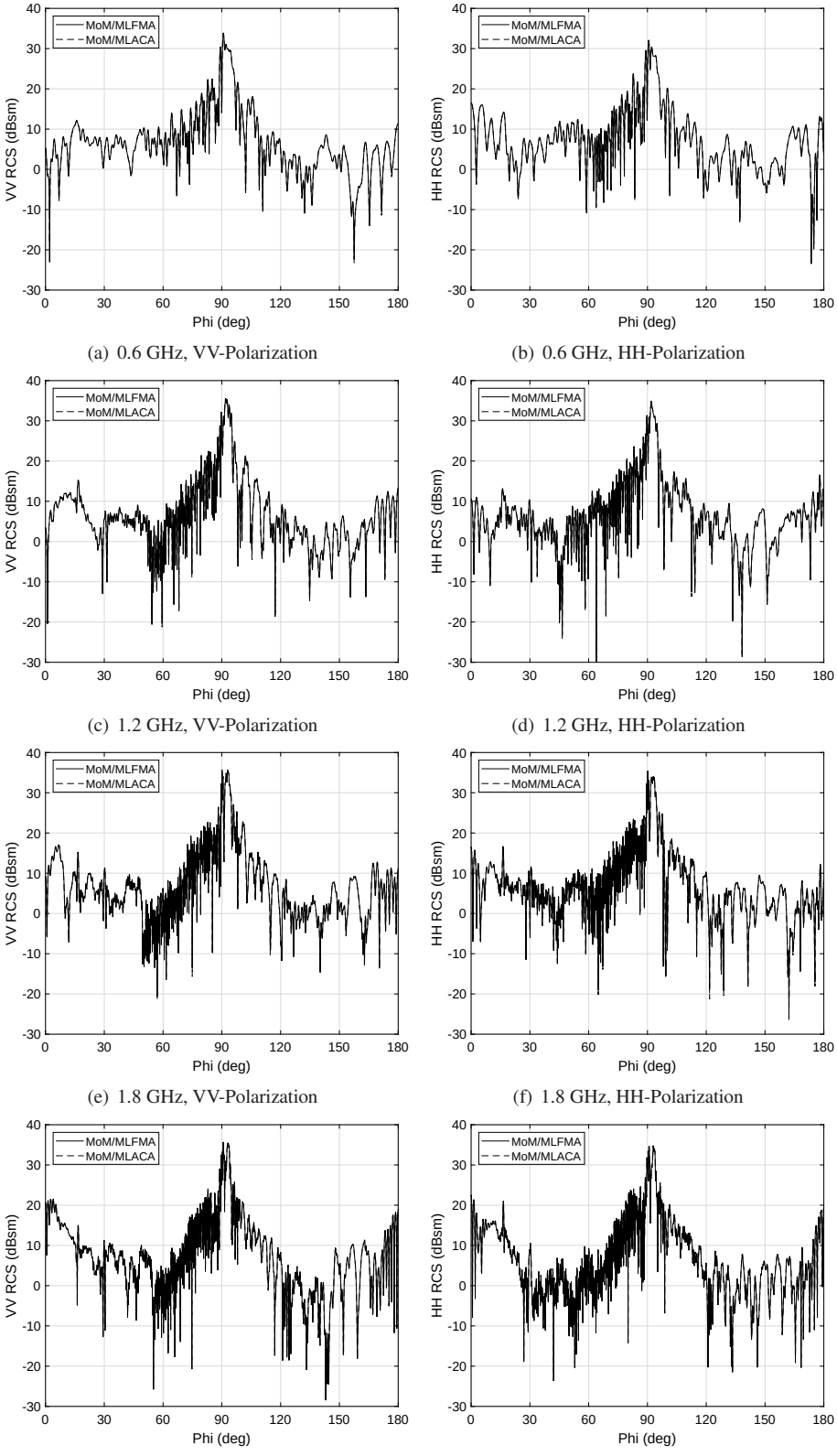
(e) 20.0 GHz, VV-Polarization

(f) 20.0 GHz, HH-Polarization

(g) 24.0 GHz, VV-Polarization

(h) 24.0 GHz, HH-Polarization

Figure 11.19: RV with Lossless Nose: RCS at 14.0, 16.0, 20.0, and 24.0 GHz



(a) 0.6 GHz, VV-Polarization

(b) 0.6 GHz, HH-Polarization

(c) 1.2 GHz, VV-Polarization

(d) 1.2 GHz, HH-Polarization

(e) 1.8 GHz, VV-Polarization

(f) 1.8 GHz, HH-Polarization

(g) 2.4 GHz, VV-Polarization

(h) 2.4 GHz, HH-Polarization

Figure 11.20: Business Jet: RCS at 0.6, 1.2, 1.8, and 2.4 GHz

11.7.7 Summary of Examples

Run metrics for the examples in this section are summarized in Table 11.4. Listed are the objects, operating frequency (f_0, in GHz), number of electric and magnetic basis functions (N_J and N_M, respectively), storage (in GB) for the near matrix (M_{near}), preconditioner (M_{pre}), radiation, receive, and transfer functions (M_{rad}, M_{rec}, and M_T, respectively), and the number of octree levels (N_O).

In Figure 11.21 is plotted the total memory requirement ($M_{near} + M_{pre} + M_{rad} + M_{rec} + M_T$) versus the number of unknowns N for all test cases in Table 11.4. We note that the storage requirement of our MLFMA/SVD algorithm scales approximately as $O(N \log N)$, which is consistent with the standard MLFMA [2].

Table 11.4: Run Metrics

Object	f_0	N_J	N_M	M_{near}	M_{pre}	M_{rad}	M_{rec}	M_T	N_O
PEC Sphere	0.75	7680	0	.039	.037	.005	.004	.005	4
PEC Sphere	1.5	30720	0	.16	.15	.019	.018	.013	5
PEC Sphere	3.0	122880	0	.63	.60	.075	.072	.035	6
PEC Sphere	6.0	491520	0	2.6	2.3	.31	.29	.10	7
PEC Sphere	12.0	1966080	0	9.8	8.7	1.2	1.1	.311	8
PEC Sphere	24.0	7864320	0	39	35	4.8	4.6	1.0	9
Dielectric Sphere	3.0	122880	122880	2.6	2.4	.24	.78	.10	6
Dielectric Sphere	6.0	491520	491520	10	9.5	.96	3.2	.32	7
Dielectric Sphere	12.0	1966080	1966080	42	37	3.8	13	1.0	8
Coated Sphere	3.0	153600	122880	2.8	2.6	.28	.86	.10	6
Coated Sphere	6.0	614400	491520	11	10	1.1	3.5	.32	7
Coated Sphere	12.0	2457600	1966080	44	39	4.5	14	1.0	8
Thin Plate	1.0	18304	0	.14	.11	.003	.003	.001	4
Thin Plate	2.0	73472	0	.60	.48	.011	.011	.003	5
Thin Plate	4.0	294400	0	2.5	1.9	.049	.049	.008	6
Thin Plate	8.0	1178624	0	10	7.8	.19	.19	.023	7
Thin Plate	16.0	4716544	0	41	32	.77	.77	.073	8
Dielectric Nose RV	14.0	1598780	46988	12	10	.56	1.2	.78	8
Dielectric Nose RV	16.0	1750780	48220	12	10	.72	1.7	.81	8
Dielectric Nose RV	20.0	3310288	93076	25	21	1.1	2.6	1.8	9
Dielectric Nose RV	24.0	4877898	144420	37	31	1.8	4.0	2.5	9
Business Jet	0.6	645588	0	6.2	5.7	.22	.21	.12	8
Business Jet	1.2	2409186	0	20	19	.83	.80	.40	9
Business Jet	1.8	5333445	0	41	37	1.8	1.7	.59	9
Business Jet	2.4	9059208	0	66	59	3.2	3.0	1.4	10

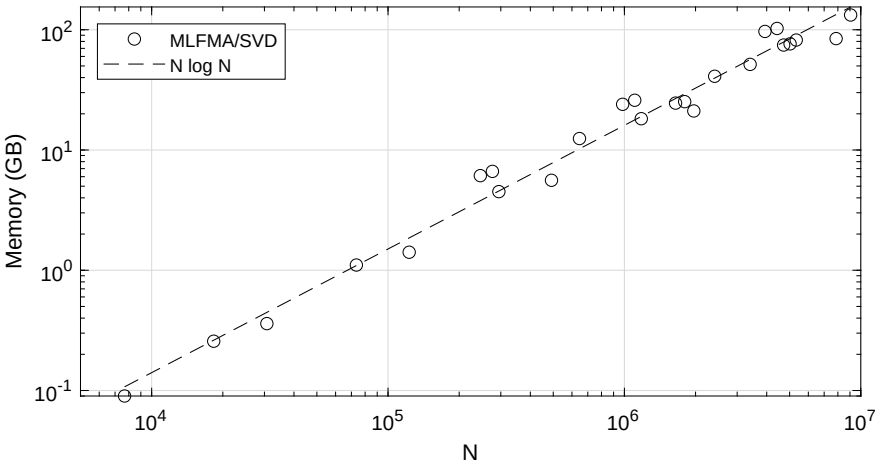


Figure 11.21: MLFMA/SVD Storage versus Number of Unknowns N

11.7.8 Preconditioner Performance

We will next compare the memory requirements and overall effectiveness of the preconditioners discussed in Section 11.5. We consider again the thin plate from Section 11.7.4 at 1.0, 2.0, 4.0 and 8.0 GHz, and as it is an open surface, only the EFIE is applied. As the EFIE is known to produce a poorly conditioned system matrix, this should be a good test article. Unless specified otherwise, all run settings are identical to those used in Sections 11.7.3-11.7.6.

11.7.8.1 Compressed versus Uncompressed Preconditioners

In practice we will compress the off-diagonal preconditioner blocks, so we must show that preconditioning effectiveness is not adversely impacted by the compression. To establish a baseline, we will first examine the performance of each preconditioner with no compression applied. For the ILUT preconditioner, fill-in blocks are dropped with a tolerance $\tau_{ILUT} = 10^{-4}$ but are not compressed, resulting in a much higher memory usage.

Table 11.5 summarizes the iteration counts for the preconditioners at each frequency for vertically and horizontally-polarized incident plane waves at an incident angle of 45 degrees off of broadside. GMRES iteration is used to a residual tolerance of 10^{-3}, with 100 iterations allowed before restarting and no limit on the number of iterations. Compared to the non-preconditioned iterations, we note that the diagonal preconditioner reduces the iteration count a fair amount, up to 24 percent at 8.0 GHz. The block diagonal preconditioner is more effective at 0.5 and 1.0 GHz, however at 2.0 GHz and above, convergence degrades significantly. At 8.0 GHz, the iteration did not converge after 10^4 iterations and was halted manually. It is with the ILU and SAI preconditioners where convergence improves dramatically. Comparing ILU(0), ILUT and SAI, we note that ILUT is the clear winner among the three, with SAI only slightly behind ILUT, and ILU(0) falling far behind.

Table 11.5: Thin Plate Iteration Counts (Uncompressed Preconditioner)

f_0 (GHz)	0.5		1.0		2.0		4.0		8.0	
Polarization	V	H	V	H	V	H	V	H	V	H
None	384	346	524	502	718	544	884	709	1178	933
Diagonal	347	300	489	466	683	523	905	700	1198	997
Block Diagonal	121	93	355	255	1114	639	3223	1318	fail	fail
ILU(0)	9	9	17	18	29	30	50	54	66	68
ILUT	5	6	8	9	12	12	17	17	27	26
SAI	7	7	12	11	16	13	23	18	32	23

Table 11.6: Thin Plate Preconditioner Storage (Uncompressed, in GB)

f_0 (GHz)	0.5	1.0	2.0	4.0	8.0
Block Diagonal	.0011	.051	.22	.9	3.6
ILU(0)	.075	.40	1.8	7.7	32
ILUT	.097	.81	4.9	18.2	73
SAI	.075	.40	1.8	7.7	32

Table 11.7: Thin Plate Iteration Counts (Compressed Preconditioner)

f_0 (GHz)	0.5		1.0		2.0		4.0		8.0	
Polarization	V	H	V	H	V	H	V	H	V	H
ILU(0)	9	9	17	18	29	30	49	53	65	68
ILUT	5	6	8	9	12	12	17	17	29	28
SAI	7	7	12	11	16	13	23	18	31	23

Table 11.6 shows the storage requirements (in GB) for the preconditioners in uncompressed form. The diagonal preconditioner is not listed as its storage requirements are negligible. Of particular note is the large requirement of the ILUT preconditioner compared to ILU(0) and SAI, due to added fill-in blocks. SAI and ILU(0) have the same sparsity pattern, thus their storage requirements are identical.

We next consider the same example, but with the ILU(0), ILUT, and SAI preconditioners in compressed form. For regular off-diagonal blocks, $\tau_{SVD} = 10^{-3}$. For ILUT, $\tau_{SVD} = 10^{-1}$ for fill-in blocks, and the drop tolerance is $\tau_{ILUT} = 10^{-4}$. In Table 11.7 are summarized the iteration counts. We note that these are essentially unchanged from those in Table 11.5, confirming that the preconditioners continue to perform well after compression. Table 11.8 summarizes the corresponding storage requirements (M_{pre}, in GB) for the compressed preconditioners, and the amount of compression (C, in percent). The compression level for each is significant, particularly for ILUT with its many fill-in blocks.

Table 11.8: Thin Plate Preconditioner Storage (Compressed, in GB)

f_0 (GHz)	0.5		1.0		2.0		4.0		8.0	
	M_{pre}	C(%)	M_{pre}	C(%)	M_{pre}	C(%)	M_{pre}	C(%)	M_{pre}	C(%)
ILU(0)	.021	71	.11	73	.45	76	1.9	76	7.6	76
ILUT	.022	78	.11	86	.48	91	1.9	89	7.8	89
SAI	.024	68	.12	69	.53	71	2.2	72	8.9	72

Table 11.9: Thin Plate Iteration Counts versus Cube Size

λ	0.25		0.375		0.5		0.75	
Polarization	V	H	V	H	V	H	V	H
ILU(0)	1*	1*	178	164	68	70	1*	1*
ILUT	47	38	65	59	29	28	20	22
SAI	48	36	35	27	31	23	20	17

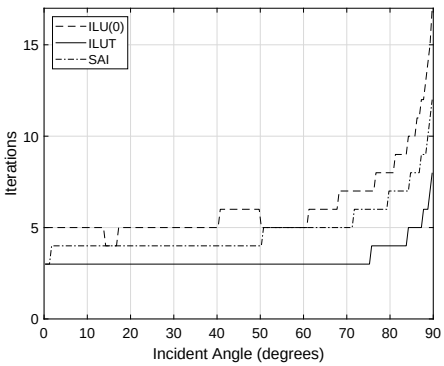
* Iteration failed.

11.7.8.2 Performance versus Incident Angle

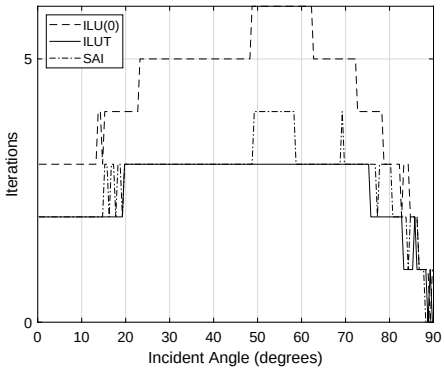
We next compute the RCS of the thin plate for 361 incident angles from 0 to 90 degrees, where 0 degrees is at broadside. The finest level cube size is 0.5λ, and the previous solution vector is used as the initial guess for subsequent iterations. In Figures 11.22a–11.22h are compared the iteration counts for the ILU(0), ILUT and SAI preconditioners at 1.0, 2.0, 4.0 and 8.0 GHz, for vertical and horizontal incidence. Again, ILU(0) performs the worst of the three. ILUT performs as well as or slightly better than SAI at 4.0 GHz or below and is slightly worse at 8.0 GHz in horizontal polarization and at higher angles in vertical polarization.

11.7.8.3 Performance versus Cube Size

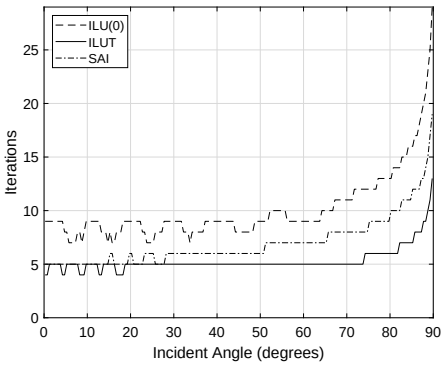
Finally, we consider the performance of the ILU(0), ILUT, and SAI preconditioners at 8.0 GHz, for finest level cube sizes of 0.25λ, 0.375λ, 0.5λ, and 0.75λ. Summarized in Table 11.9 are the iteration counts for vertical and horizontal incidence. We observe that as the size of the cubes grows, the iteration count decreases. This result is expected, as the added information content in the near matrix results in a more effective preconditioner. Unexpectedly, ILU(0) failed at 0.25λ and 0.75λ, where the convergence test passed after a single iteration, but the result was incorrect. This suggests that ILU(0) is not reliable and should not be used in practice. Table 11.10 summarizes the corresponding storage requirements (M_{pre}, in GB) for the preconditioners, and the amount of compression (C, in percent). The level of compression increases with cube size, as noted previously in Section 11.3.2.1.



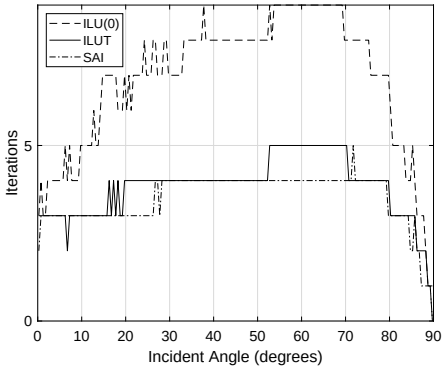
(a) 1.0 GHz, VV-Polarization



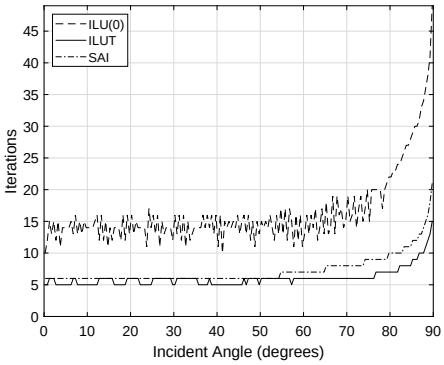
(b) 1.0 GHz, HH-Polarization



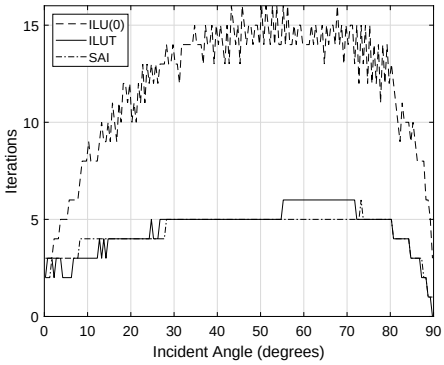
(c) 2.0 GHz, VV-Polarization



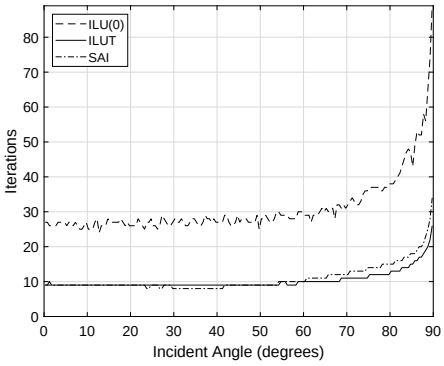
(d) 2.0 GHz, HH-Polarization



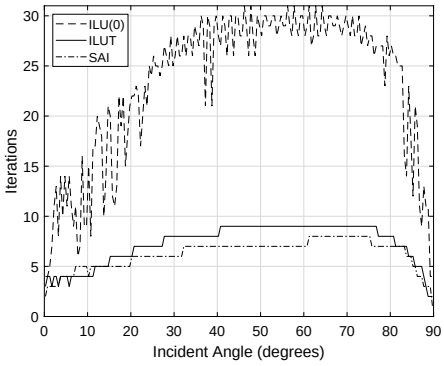
(e) 4.0 GHz, VV-Polarization



(f) 4.0 GHz, HH-Polarization



(g) 8.0 GHz, VV-Polarization



(h) 8.0 GHz, HH-Polarization

Figure 11.22: Thin Plate Iteration Counts versus Incident Angle

Table 11.10: Thin Plate Preconditioner Storage (in GB) versus Cube Size

λ	0.25		0.375		0.5		0.75	
	M_{pre}	C(%)	M_{pre}	C(%)	M_{pre}	C(%)	M_{pre}	C(%)
ILU(0)	3.4	58	5.3	71	7.6	76	13.3	81
ILUT	3.7	83	5.6	88	7.8	89	13.7	92
SAI	4.3	47	6.4	64	8.9	72	15.0	79

11.7.9 Initial Guess in Iterative Solution

When using an iterative solver, there is often no *a priori* knowledge regarding the solution vector. Therefore, for an arbitrary right-hand side vector, the usual approach is to simply initialize the solution vector with zeros. In radar cross section problems, where incident angles are closely spaced, the right-hand side and solution vectors will be similar between angles. Therefore, the previous solution vector can be reused, reducing the number of iterations. In [2], it was suggested that this can be improved further by adjusting the phase of the solution vector between incident angles. To illustrate, consider the plane-wave incident vector $\hat{\mathbf{r}}_1$, where the phase term at a point $\mathbf{r}$ in the free space region R_0 is

$$\psi_1(\mathbf{r}) = e^{jk_0\hat{\mathbf{r}}_1\cdot\mathbf{r}}\,. \tag{11.96}$$

Between angles, we can apply the following correction to the solution vector:

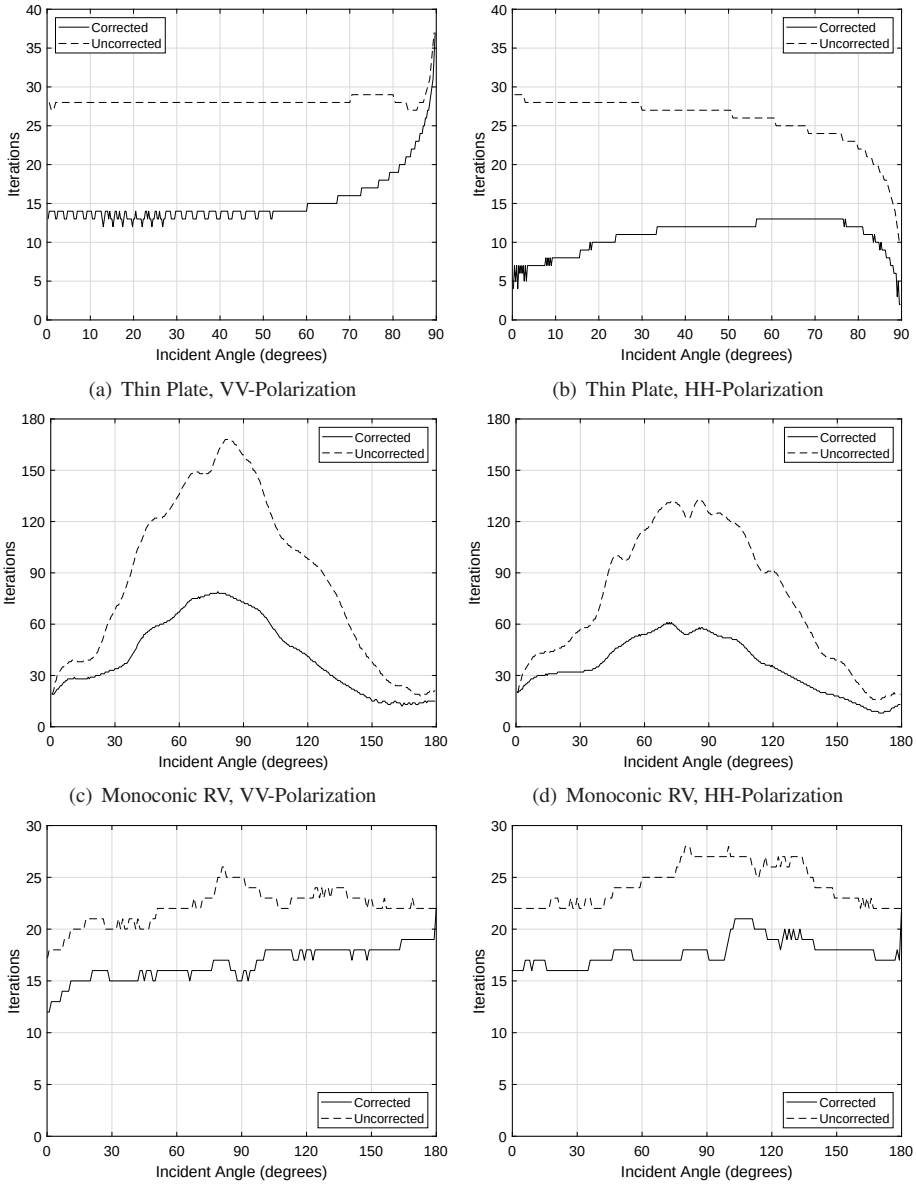
$$\Delta\psi(\mathbf{r}) = e^{jk_0(\hat{\mathbf{r}}_2-\hat{\mathbf{r}}_1)\cdot\mathbf{r}}. \tag{11.97}$$

If using RWG basis functions, this can be implemented by adjusting the phase of the solution vector elements corresponding to the electric and magnetic basis functions assigned to each edge in R_0. For a given edge, the correction is

$$\Delta\psi(\mathbf{r}_c) = e^{jk_0(\hat{\mathbf{r}}_2-\hat{\mathbf{r}}_1)\cdot\mathbf{r}_c}, \tag{11.98}$$

where $\mathbf{r}_c$ is the center point of the edge.

We now consider several examples that demonstrate the effectiveness of this phase correction. We first consider the thin plate from Section 11.7.4 at 8.0 GHz, for 361 incident angles from 0 to 90 degrees (a 0.25-degree step size). The number of iterations per right-hand side is compared for vertical and horizontal incidence in Figures 11.23a and 11.23b, respectively. We see that the number of iterations has been decreased significantly by adding the correction. Figures 11.23c and 11.23d present a similar comparison for the monoconic reentry vehicle from Section 11.7.5 at 16.0 GHz, for 361 incident angles from 0 to 180 degrees (a 0.5-degree step size). The improvement is again significant. Finally, Figures 11.23e and 11.23f show a comparison for the business jet from Section 11.7.6 at 1.2 GHz, for 181 incident angles from 0 to 180 degrees (a 1-degree step size). The improvement is again substantial.



(a) Thin Plate, VV-Polarization

(b) Thin Plate, HH-Polarization

(c) Monoconic RV, VV-Polarization

(d) Monoconic RV, HH-Polarization

(e) Business Jet, VV-Polarization

(f) Business Jet, HH-Polarization

Figure 11.23: Phase Correction Impact on Iterative Solver Performance

References

[1] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propagat. Magazine*, vol. 35, pp. 7–12, June 1993.

[2] W. C. Chew, J. M. Jin, E. Michielssen, and J. Song, *Fast and Efficient Algorithms in Computational Electromagnetics*. Artech House, 2001.

[3] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, pp. 325–348, 1987.

[4] J. Stratton, *Electromagnetic Theory*. McGraw-Hill, 1941.

[5] D. E. Amos, "A subroutine package for Bessel functions of a complex argument and nonnegative order," Tech. Rep. SAND85-1018, Sandia National Laboratory, May 1985.

[6] P. Havé, "A parallel implementation of the fast multipole method for Maxwell's equations," *Int. J. Numer. Meth. Fluids*, vol. 43, no. 8, pp. 839–864, 2003.

[7] J. M. Song and W. C. Chew, "Error analysis for the truncation of multipole expansion of vector Green's functions," *IEEE Microwave Wireless Components Lett.*, vol. 11, no. 7, pp. 311–313, 2001.

[8] O. M. Bucci and G. Franceschetti, "On the spatial bandwidth of scattered fields," *IEEE Trans. Antennas Propagat.*, vol. 35, pp. 1445–1455, December 1987.

[9] N. Geng, A. Sullivan, and L. Carin, "Fast multipole method for scattering from an arbitrary PEC target above or buried in a lossy half space," *IEEE Trans. Antennas Propagat.*, vol. 49, pp. 740–748, May 2001.

[10] O. M. Bucci, C. Gennarelli, and C. Savarese, "Optimal interpolation of radiated fields over a sphere," *IEEE Trans. Antennas Propagat.*, vol. 39, pp. 1633–1643, November 1991.

[11] M. Böhme and D. Potts, "A fast algorithm for filtering and wavelet decomposition on the sphere," *Trans. Numer. Anal*, vol. 16, pp. 70–93, 2002.

[12] J. M. Song, C. C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *IEEE Trans. Antennas Propagat.*, vol. 45, pp. 1488–1493, October 1997.

[13] S. Velamparambil and W. C. Chew, "Analysis and performance of a distributed memory multilevel fast multipole algorithm," *IEEE Trans. Antennas Propagat.*, vol. 53, pp. 2719–2727, August 2005.

[14] J. Foley, A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*. Addison-Wesley, 1996.

[15] B. Decker, "World geodetic system 1984," Tech. Rep. AD-A 167 570, Defense Mapping Agency, April 1996.

[16] F. Lemoine, S. Kenyon, J. Factor, R. Trimmer, N. Pavlis, D. Chinn, C. Cox, S. Klosko, S. Luthcke, M. Torrence, Y. Wang, R. Williamson, E. Pavlis, R. Rapp, and T. R. Olson, "The development of the joint NASA GSFC and the national imagery and mapping agency (NIMA) geopotential model EGM96," Tech. Rep. NASA/TP—1998–206861, National Aeronautics and Space Administration, July 1998.

[17] S. Koc, J. M. Song, and W. C. Chew, "Error analysis for the numerical evaluation of the diagonal forms of the spherical addition theorem," *SIAM J. Numer. Anal.*, vol. 36, no. 3, pp. 906–921, 1999.

[18] J. Song and W. Chew, "Interpolation of translation matrix in MLFMA," *Microw. Opt. Technol. Lett.*, vol. 30, pp. 109–113, July 2001.

[19] A. Elafrou, G. Goumas, and N. Koziris, "Performance analysis and optimization of sparse matrix-vector multiplication on modern multi- and many-core processors," in *2017 46th International Conference on Parallel Processing (ICPP)*, pp. 292–301, 2017.

[20] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," *Supercomputing (SC07)*, 2007.

[21] Y. Saad, *Iterative Methods for Sparse Linear Systems*. PWS, 1st ed., 1996.

[22] T. Malas and L. Gurel, "Accelerating the multilevel fast multipole algorithm with the sparse-approximate-inverse (SAI) preconditioning," *SIAM J. Sci. Comput.*, vol. 31, no. 3, pp. 1968–1984, 2009.

[23] B. Carpentieri, I. Duff, L. Giraud, and G. Sylvand, "Combining fast multipole techniques and an approximate inverse preconditioner for large electromagnetism calculations," *SIAM J. Sci. Comput.*, vol. 27, no. 3, pp. 774–792, 2005.

[24] J. Lee, J. Zhang, and C. Lu, "Sparse inverse preconditioning of multilevel fast multipole algorithm for hybrid integral equations in electromagnetics," *IEEE Trans. Antennas Propagat.*, vol. 52, no. 9, pp. 2277–2287, 2004.