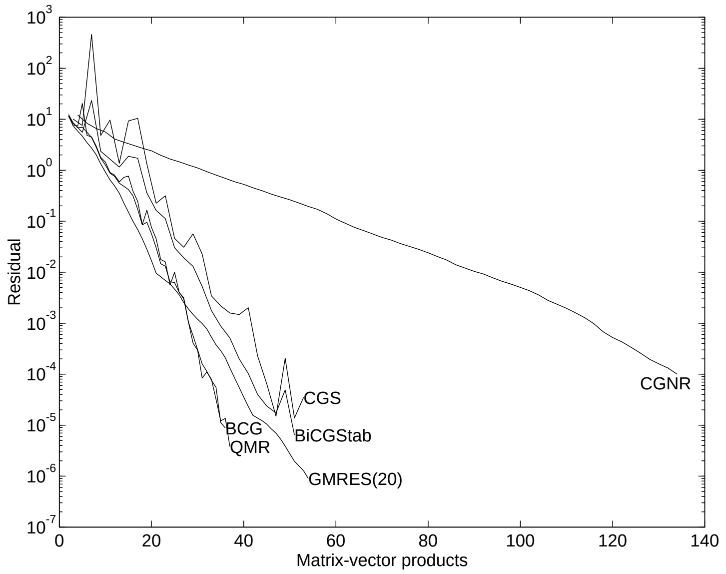

Figure 3.10: Convergence of the tested iterative methods.

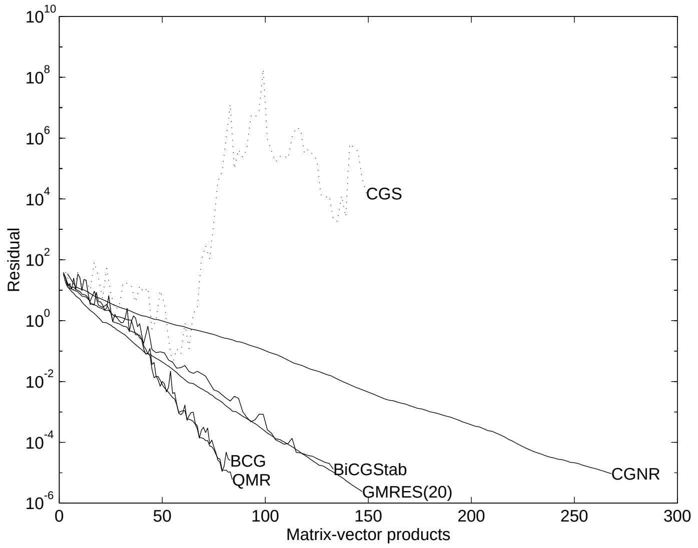

Figure 3.11: Convergence of tested iterative methods. Note that CGS starts to oscillate and BCG is not converging monotonically.

Method	N_{mv}
QMR	1
GMRES	1
BCG	1
CGNR	2
CGS	2
BiCGStab	2

Table 3.11: The number of matrix-vector products per iteration N_{mv} for the tested iterative methods.

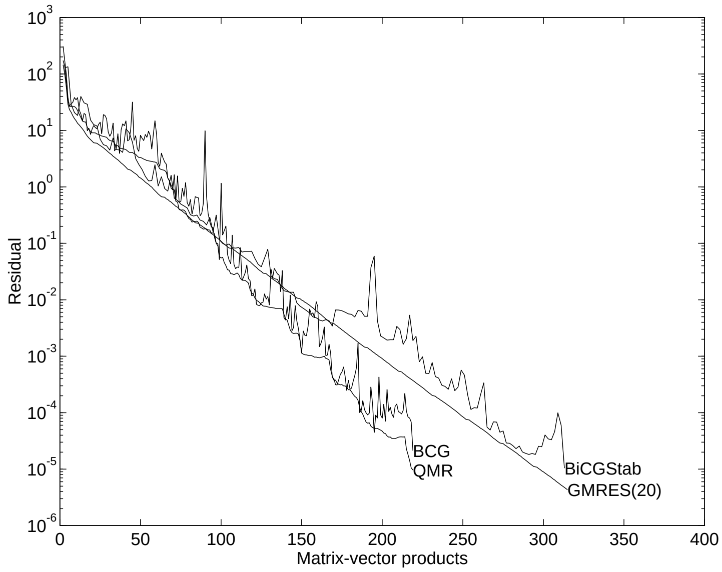

Figure 3.12: Convergence of the four best iterative methods.

m and the size parameter x_0. QMR was generally found to be approximately three times as fast as CGNR.

Figure 3.13 shows a comparison of QMR with GMRES using several memory lengths, and with full GMRES. The different GMRES versions are converging with a linear rate after an initial sublinear phase. The full GMRES and QMR seem find a superlinear rate of convergence. The three iteration phases, sublinear, linear, and superlinear, are typically found when using Krylov-subspace methods [30]. The superlinear phase might not be the true superlinear convergence. Instead, the convergence accelerates as the iterative methods find the eigenvalues that are far away from the line and remove the effect of those eigenvalues. After this the convergence of the iteration is linear and is controlled by the linear part of the spectrum.

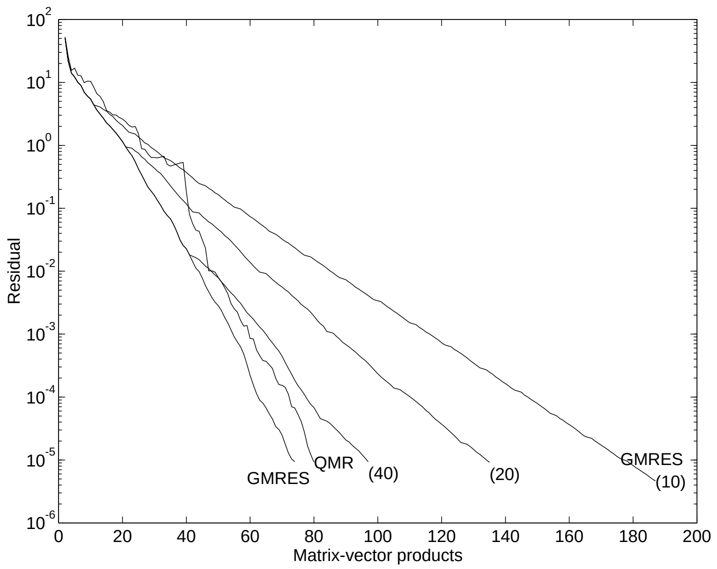

Figure 3.13: A comparison of QMR and GMRES with different memory lengths and with full GMRES.

QMR converges with only a few more iterations that full GMRES. GMRES is optimal in the sense that it computes a minimal residual solution. However, full GMRES needs all the previously computed iterates and can thus be time and memory consuming. QMR is based on short recurrencies and can be implemented efficiently. QMR was chosen to be the iterative method in the production version of this code. Full GMRES can in some cases be afforded because the extra vector operations can be computed quite cheaply in comparison to the calculation of the matrix-vector product. However, if the matrix-vector product is computed using a special fast method, as described in Chapter 4, the vector operations of GMRES may take a substantial amount of time and memory.

The convergence of block QMR (BQMR) is compared to that of QMR. The number of iterations and solution times for a number of geometries are shown in Table 3.12. The simulations were run using index of refraction $m = 3 + 4i$ and size parameter $x_0 = 0.3$. The convergence limit was 10^{-6} and the blocksize for BQMR was 6. The fast Fourier transform was used to compute the matrix-vector product. In Figure 3.14 the block size of block QMR is varied. The index of refraction was $m = 3+4i$, size parameter was $x_0 = 0.3$, and the number of dipoles was $N_{dip} = 1064$. After a certain size, increasing the block size does not results in a smaller CPU time. At some point, the number of iterations decreases quite slowly while the extra matrix arithmetic in the block QMR algorithm takes its time too. Also some problems with the numerical stability of block QMR might arise as the block size is increased. The orthogonality of some vectors in block QMR has to be maintained and this gets

N	I_{QMR}	I_{BQMR}	t_{QMR}	t_{BQMR}
32	18	15	1.15	0.97
136	31	27	5.18	4.5
304	44	36	20.5	16
1064	103	76	125	96
2320	177	140	580	446

Table 3.12: A comparison of QMR and block QMR (BQMR). The size of the linear system is N, the number of iterations are I_{QMR} and I_{BQMR}. The solution times per right hand side are t_{QMR} and t_{BQMR}.

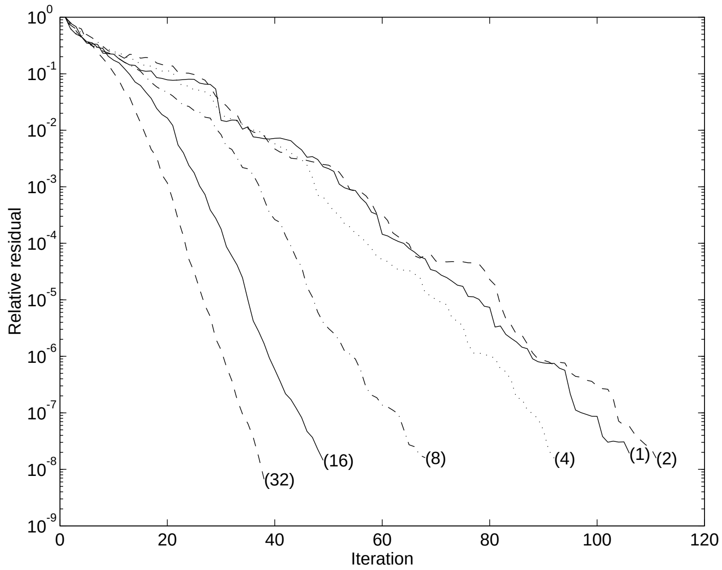

Figure 3.14: The convergence of QMR and block QMR with different block sizes.

harder with large block sizes.

3.3.1 Performance of QMR

Here the number of iterations and the CPU time of QMR are shown for several geometries. Note that in this subsection the index of refraction was $m = 1.6 + 0.05i$, which is smaller than in the earlier figures, resulting in faster convergence.

In the astrophysical calculations, the interest is in the scattering of light by micron-scale particles, leading to dipole configurations of tens or hundreds of thousands of dipoles. In practical calculations the solution is not needed no a very high accuracy because orientational averages of the results are taken.

N_{dip}	x_0	N_{iter}	t
32	0.3	6	0.23
136	0.188	7	0.46
304	0.144	7	1.52
1064	0.0947	7	3.3
2330	0.0731	7	10
5232	0.0555	7	21
10048	0.0448	7	37
20336	0.0355	7	91
137376	0.01875	7	562

Table 3.13: The number of iterations for QMR (N_{iter}) and the elapsed CPU time in seconds (t) needed to reduce the initial residual by 10^{-5} when the *same* particle is discretized with different number of dipoles and with different size parameter associated with each dipole. The number of unknowns is three times the number of dipoles N_{dip}.

When the same particle is discretized with many resolutions, the spectra of the coefficient matrices seem to convergence as shown in Figures 3.1–3.4. Thus the coefficient matrices seem to have asymptotically bounded condition numbers and thus the number of iterations for the iterative methods should be bounded from above. Table 3.13 shows the number of iterations needed to reduce the initial residual by a factor of 10^{-5} and CPU times for QMR when the same particle is discretized using nine different resolutions. The fast Fourier transform was used to calculate the matrix-vector product. As predicted by the spectrum, the number of iterations is remarkably constant. The CPU time grows only slightly faster than the number of dipoles N_{dip}. In fact, with small particles the CPU time grows slower than linearly, which is probably due to vectorization: long data vectors can be computed more efficiently than short vectors on a vector computer.

When larger and larger particles are discretized so that the size of the computational box or the size parameter x_0 stays the same, the number of iterations for QMR of course increases. Table 3.14 shows the number of iterations and the CPU time for pseudospheres of increasing size. The relative convergence criterion was 10^{-5}. The fast Fourier transform was used to compute the matrix-vector product.

N_{dip}	iterations	CPU time (s)
32	7	0.2
136	8	0.5
304	10	2.0
1064	14	6.1
2330	18	23
5232	25	65
10048	32	152
20336	48	551
137376	135	9361

Table 3.14: The number of iterations needed to reduce the initial residual by 10^{-5} and the corresponding CPU time for pseudospheres of different size. The number of equations is three times the number of dipoles N_{dip}. Here the size parameter is $x_0 = 0.3$.

3.4 Preconditioning

In order to reduce the number of iterations in iterative methods, a technique called preconditioning is used. This means that the original linear system $Ax = b$ is multiplied by a matrix M^{-1} so that iterative methods converge faster when solving the resulting new linear system. The preconditioning matrix should somehow approximate the original coefficient matrix and linear systems with the coefficient matrix M should be easy to solve.

For the systems of linear equations arising in the scattering calculations, the iterative methods converge really fast so that in general no preconditioning is necessary.

Possible preconditioning strategies are to use an inner iteration as a preconditioner. We briefly tried to use SSOR or QMR as the inner iteration, but these preconditioners failed to work in a satisfactory way.

Chapter 4

Calculation of the matrix-vector product

The performance of iterative solvers for dense linear equations is determined by the number of iterations and the time it takes to multiply the coefficient matrix by a vector.

There are two evident ways to compute the matrix-vector product. The first one is simply to form the coefficient matrix and to keep it in memory. This is indeed the fastest way to calculate the matrix-vector product for small to medium-size matrices. For example, if half a gigabyte of central storage is available, dense matrices up to 7000×7000 elements can be stored using single precision. The second method, used for larger matrices, is to either store the matrix in the disk ("out of core") or recompute the matrix elements whenever necessary.

In order to be able to solve really large dense linear systems efficiently one has to make use of the "all large dense matrices are structured" hypothesis [10], according to which nature does not just randomly throw n^2 numbers at us. Instead, there is some structure in most dense coefficient matrices and we must find an algorithm that exploits this structure.

In this work, we have applied two fast methods, the fast Fourier transform and the fast multipole method to compute the matrix-vector product.

4.1 The fast Fourier transform

Goodman, Draine, and Flatau [20] showed how to use the fast Fourier transform in conjunction with the DDA. If the dipoles occupy positions in a regular cubical lattice and the lattice dimensions are doubled in each dimension, the matrix-vector multiplication reduces into a 3-D convolution. This convolution can be calculated using a 3-D fast Fourier transform (FFT).

The use of FFT is not limited to cubical arrays of dipoles. The lattice spacings and extents can be different in different dimensions. Not all the lattice positions need be occupied, the artificial dipoles affect only the performance of the Fourier transform. The size of the FFT is determined by first finding a lattice so that all the dipoles sit on that lattice. The smallest rectangular part of the lattice containing all the dipoles is taken as the computational lattice.

The use of the Fourier transform is very efficient if the dipoles are packed together in a dense and regular way. On the other hand, if the dipoles form a sparse dendrite-like particle generated for example by the diffusion-limited aggregation process, many artificial dipoles are needed in order to be able to use the FFT and the method slows down [28].

Let the dipoles be located on a cubic lattice. If the dipoles do not fill all the lattice positions, the matrix-vector product $y = Ax$ can be calculated as follows. For the whole lattice the product would be $\tilde{y} = \tilde{A}\tilde{x}$. In order to calculate Ax, copy x into the corresponding positions of $\tilde{x}$ and set the remaining components (the positions in the lattice where there is no real computational dipole) of $\tilde{x}$ to zero. After calculating $\tilde{y} = \tilde{A}\tilde{x}$ one can gather y from $\tilde{y}$ while the other components of $\tilde{y}$ can be ignored.

Let the dipoles in the lattice be indexed by a vector $\mathbf{i} = (i_x, i_y, i_z)$. Now the dipole positions are given by $\mathbf{r_i} = (i_x d, i_y d, i_z d) + \mathbf{r_0}$, where d is the lattice spacing and $i_x \in \{0, 1, \ldots, n_x - 1\}$, $i_y \in \{0, 1, \ldots, n_y - 1\}$, and $i_z \in \{0, 1, \ldots, n_z - 1\}$. The lattice has $n_L = n_x n_y n_z$ sites. The number of dipoles in the computation is generally less that n_L.

For the rectangular lattice the coefficient matrix, if written as an $n \times n$ matrix of blocksize 3×3, is a symmetric Block-Toeplitz matrix, as shown by Flatau, Stephens, and Draine [14]. A Toeplitz matrix is of the form

$$B = \begin{pmatrix} b_0 & b_1 & \cdots & b_{n-1} \\ b_{-1} & b_0 & & \vdots \\ \vdots & & \ddots & \vdots \\ b_{-n+1} & \cdots & \cdots & b_0 \end{pmatrix}. \tag{4.1}$$

Here we will partition the matrix into a 3×3 block matrix corresponding to the three components of the electric vector:

$$A = \begin{pmatrix} A_{xx} & A_{xy} & A_{xz} \\ A_{yx} & A_{yy} & A_{yz} \\ A_{zx} & A_{zy} & A_{zz} \end{pmatrix}. \tag{4.2}$$

A is symmetric: $A_{kl} = A_{lk}^T$. Each block A_{kl} is a symmetric block-Toeplitz matrix of size $n_L \times n_L$:

$$A_{kl} = \begin{pmatrix} B_0 & B_1 & \cdots & B_{n_x-1} \\ B_{-1} & B_0 & & \vdots \\ \vdots & & \ddots & \vdots \\ B_{-n_x+1} & \cdots & \cdots & B_0 \end{pmatrix}. \tag{4.3}$$

Here each block B_p is in general a nonsymmetric block-Toeplitz matrix of size $n_y n_z \times n_y n_z$:

$$B_p = \begin{pmatrix} C_0 & C_1 & \cdots & C_{n_y-1} \\ C_{-1} & C_0 & & \vdots \\ \vdots & & \ddots & \vdots \\ C_{-n_y+1} & \cdots & \cdots & C_0 \end{pmatrix}. \tag{4.4}$$

Finally, each matrix C_q is a nonsymmetric Toeplitz matrix of size $n_z \times n_z$:

$$C_q = \begin{pmatrix} c_0 & c_1 & \cdots & c_{n_z-1} \\ c_{-1} & c_0 & & \vdots \\ \vdots & & \ddots & \vdots \\ c_{-n_z+1} & \cdots & \cdots & c_0 \end{pmatrix}. \tag{4.5}$$

Note that each block B_p and C_q is either symmetric or antisymmetric depending on which A_{kl} they correspond to. In other words, $B_{-p} = \pm B_p$, $C_{-q} = \pm C_q$, and $c_{-s} = \pm c_s$, which means that $B_p^T = \pm B_p$ and $C_q^T = \pm C_q$.

If the dipoles do not sit on a rectangular lattice, the block-Toeplitz form does not apply. In that case, we can say that the blocks A_{kl} are symmetric but they do not have any block structure.

Now we will indicate how the product Ax can be computed quickly. Equation (2.4) depends on the dipole positions $\mathbf{r_i}$ and $\mathbf{r_j}$ through the difference $\mathbf{r_i} - \mathbf{r_j}$. In the case of a rectangular lattice, this equation depends only on the difference of index vectors $\mathbf{i} - \mathbf{j}$ so that for matrices $D = A_{kl}$ the matrix elements can be given as $D_{\mathbf{ij}} = D'_{\mathbf{i-j}}$. The information for $D'_{\mathbf{i-j}}$ is present in the first row of the matrix D except for the indices where the value of some index i_x, i_y, or i_z is negative. These values can be computed from the corresponding values for positive indices by observing the symmetries of B_p and C_q. For example, the diagonal blocks A_{kk} are unaffected by the change $i_x \to -i_x$, $i_y \to -i_y$, or $i_z \to -i_z$. On the other hand, if $D = A_{xy}$, the transformation $i_x \to -i_x$ or $i_y \to -i_y$ changes the sign to -1 while the change $i_z \to -i_z$ leaves the sign unaffected.

Let D' be doubled in each dimension so that the above mentioned symmetries are taken into account. Now the product $y = Dx$ can be written as

$$y_{\mathbf{i}} = \sum_{\mathbf{j}=(0,0,0)}^{(n_x-1,n_y-1,n_z-1)} D'_{\mathbf{i}-\mathbf{j}} x_{\mathbf{j}}, \tag{4.6}$$

which is a 3D convolution of D' and x, if D' and x are regarded as periodic in each dimension and having dimensions $2n_x$, $2n_y$, and $2n_z$. When x is expanded the new components are zeroed.

Now the 3D-convolutions (4.6) can be computed efficiently using 3D FFT's. The operation count is of the order $\mathcal{O}(n_L \log n_L)$ compared to the operation count of $\mathcal{O}(n^2)$ for the regular matrix-vector product. However, this requires that the lattice dimensions be powers of two (or of the form $2^p 3^q 5^r$ for some FFT implementations). Thus we have to enlarge the grid and pad the new positions with zeroes.

4.2 The fast multipole algorithm

The fast multipole method [35, 21, 6] is an efficient method to compute the electric potential of a system of N charges or multipoles. It can also be used in the solution of integral equations of potential theory [35]. The fast multipole method was initially used in simulations where the Coulomb or gravitational potential is used. In this work we will use the fast multipole method to compute the matrix-vector multiplication in electromagnetic scattering in three dimensions. Here the matrix-vector product should be viewed as follows: given x, the current guess for the electric field caused by the dipoles, Ax gives the combined field of all the other dipoles at all the dipole positions.

The full fast multipole method is a rather complicated algorithm that can achieve $\mathcal{O}(N)$ complexity as opposed to the brute force method of direct pairwise computation with $\mathcal{O}(N^2)$ complexity. For two-dimensional systems, the break-even point between these two algorithms can be as low as 100 particles. In three-dimensional calculations, the break-even point is much larger, typically in the range 1000–10 000. An example of three-dimensional fast multipole computations using the Coulomb potential was given in [38].

For two-dimensional electromagnetic scattering problems, the fast multipole method was given in [11]. The three-dimensional version was given in [7], which formulates the scattering problem using a boundary integral equation. The DDA is related to a volume-integral equation. Thus we use different expansions of the potential than those found in [7].

4.2.1 The multipole and local expansions

Here we shall develop the multipole and local expansions for the dipole potential and show how the electric field and thus the matrix-vector product can be computed from these potentials. The electric field and the dipole moment of a dipole are related by $\mathbf{p}_i = \alpha \mathbf{E}_i$, where α is the polarizability.

The vector potential generated by an oscillating electric dipole at position $\mathbf{r}_i$ with dipole moment $\mathbf{p}_i$ is given by [23]

$$\mathbf{A}(\mathbf{r}) = -ik\mathbf{p}_i \frac{e^{ik|\mathbf{r}-\mathbf{r}_i|}}{|\mathbf{r}-\mathbf{r}_i|}. \tag{4.7}$$

The electric field due to this potential is given by

$$\mathbf{E}(\mathbf{r}) = \frac{i}{k} \nabla \times (\nabla \times \mathbf{A}(\mathbf{r})). \tag{4.8}$$

The expansion of the Green's function occurring in the potential (4.7) in spherical coordinates is [23]

$$\frac{e^{ik|\mathbf{r}-\mathbf{r}_i|}}{|\mathbf{r}-\mathbf{r}_i|} = 4\pi ik \sum_{n=0}^{\infty} \sum_{m=-n}^{n} h_n^{(1)}(kr) j_n(kr_i) Y_n^m(\theta,\phi) \overline{Y_n^m(\theta_i,\phi_i)}, \quad r > r_i, \tag{4.9}$$

$$4\pi ik \sum_{n=0}^{\infty} \sum_{m=-n}^{n} h_n^{(1)}(kr_i) j_n(kr) Y_n^m(\theta,\phi) \overline{Y_n^m(\theta_i,\phi_i)}, \quad r < r_i. \tag{4.10}$$

The function $h_n^{(1)}$ is the spherical Hankel function of the first kind, j_n is the spherical Bessel function, the overline denotes complex conjugation, and Y_n^m are the spherical harmonics defined by

$$Y_n^m(\theta,\phi) = (-1)^m \sqrt{\frac{(2n+1)}{4\pi} \frac{(n-m)!}{(n+m)!}} P_n^m(\cos\theta) e^{im\phi}. \tag{4.11}$$

In the fast multipole algorithm we group dipoles within a certain computational volume and compute a multipole expansion of the combined vector potential of the dipoles. This can be used to evaluate the potential in a point that is further away from the origin than any of the dipoles. We also need a local expansion that is valid when the evaluation point is closer to the origin than any of the dipoles.

The multipole coefficients of a group of N dipoles are given by

$$\mathbf{M}_{nm} = \sum_{i=1}^{N} \mathbf{p}_i j_n(kr_i)\overline{Y_n^m(\theta_i, \phi_i)}. \tag{4.12}$$

The truncated multipole expansion of the potential of the dipoles in terms of the multipole coefficients is

$$\mathbf{A}(\mathbf{r}) = 4\pi k^2 \sum_{n=0}^{n_m} \sum_{m=-n}^{n} \mathbf{M}_{nm} h_n^{(1)}(kr) Y_n^m(\theta, \phi). \tag{4.13}$$

The local coefficients are given by

$$\mathbf{L}_{nm} = \sum_{i=1}^{N} \mathbf{p}_i h_n^{(1)}(kr_i)\overline{Y_n^m(\theta_i, \phi_i)}, \tag{4.14}$$

while the truncated local expansion of the potential is

$$\mathbf{A}(\mathbf{r}) = 4\pi k^2 \sum_{n=0}^{n_m} \sum_{m=-n}^{n} \mathbf{L}_{nm} j_n(kr) Y_n^m(\theta, \phi). \tag{4.15}$$

The summations over n are truncated at a suitable value n_m. Let $\mathbf{A}_e(\mathbf{r})$ be the exact value of the potential. In the multipole expansion, we make a truncation error of

$$\begin{aligned} d &= \left| \mathbf{A}_e(\mathbf{r}) - 4\pi k^2 \sum_{n=0}^{n_m} \sum_{m=-n}^{n} \mathbf{M}_{nm} h_n^{(1)}(kr) Y_n^m(\theta, \phi) \right| \\ &= 4\pi k^2 \left| \sum_{n=n_m+1}^{\infty} \sum_{m=-n}^{n} \mathbf{M}_{nm} h_n^{(1)}(kr) Y_n^m(\theta, \phi) \right|. \end{aligned} \tag{4.16}$$

Now if all the dipoles are inside a sphere of radius R so that $|\mathbf{r}_i| < R$ and we are evaluating the field at a point $\mathbf{r}$ with $r > R_1$, it should be verified that the following holds:

$$d = c\left(\frac{R}{R_1}\right)^{n_m}, \tag{4.17}$$

where c is a constant. A similar error bound holds for the Coulomb potential in the plane [6] and for the acoustic scattering in two dimensions [36]. An equivalent error bound should hold for the local expansions.

When given a potential (4.13) or (4.15) in terms of the multipole or local coefficients we must be able to compute the electric field caused by this potential. For the multipole potential case, let us mark

$$\psi = h_n^{(1)}(kr) Y_n^m(\theta, \phi). \tag{4.18}$$

Now the electric field due to the multipole potential is given by

$$\mathbf{E}(\mathbf{r}) = 4\pi i k \sum_{nm} \left\{ \mathbf{M}_{nm} \cdot \nabla\nabla\psi + k^2 \psi \mathbf{M}_{nm} \right\}, \tag{4.19}$$

where $\nabla\nabla\psi$ is a symmetric second-rank tensor whose representation is given in spherical coordinates by

$$\begin{pmatrix} \frac{\partial^2\psi}{\partial r^2} & \frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial\psi}{\partial\theta}\right) & \frac{1}{\sin\theta}\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial\psi}{\partial\phi}\right) \\ \frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial\psi}{\partial\theta}\right) & \frac{1}{r^2}\frac{\partial^2\psi}{\partial\theta^2}+\frac{1}{r}\frac{\partial\psi}{\partial r} & \frac{1}{r^2}\frac{\partial}{\partial\theta}\left(\frac{1}{\sin\theta}\frac{\partial\psi}{\partial\phi}\right) \\ \frac{1}{\sin\theta}\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial\psi}{\partial\phi}\right) & \frac{1}{r^2}\frac{\partial}{\partial\theta}\left(\frac{1}{\sin\theta}\frac{\partial\psi}{\partial\phi}\right) & \frac{1}{r^2\sin^2\theta}\frac{\partial^2\psi}{\partial\phi^2}+\frac{1}{r}\frac{\partial\psi}{\partial r}+\frac{\cos\theta}{r^2\sin\theta}\frac{\partial\psi}{\partial\theta} \end{pmatrix}. \tag{4.20}$$

The partial derivatives of the component functions of ψ are given by

$$\begin{aligned} {h_n}^{(1)\prime}(kr) &= (nh_{n-1}^{(1)}(kr)-(n+1)h_{n+1}^{(1)}(kr))/(2n+1), \\ {h_n}^{(1)\prime\prime}(kr) &= -(2krh_n'^{(1)}+(k^2r^2-n(n+1))h_n^{(1)})/r^2, \\ \frac{\partial Y_n^m(\theta,\phi)}{\partial\theta} &= m\cot\theta\, Y_n^m(\theta,\phi)+e^{-i\phi}\sqrt{n(n+1)-m(m+1)}Y_n^{m+1}(\theta,\phi), \\ \frac{\partial Y_n^m(\theta,\phi)}{\partial\phi} &= i\,m\,Y_n^m(\theta,\phi), \\ \frac{\partial^2 Y_n^m(\theta,\phi)}{\partial\phi^2} &= -m^2Y_n^m(\theta,\phi), \\ \frac{\partial^2 Y_n^m(\theta,\phi)}{\partial\theta^2} &= Y_n^m(\theta,\phi)(m^2/\sin^2\theta-n(n+1))-\cot\theta\,\frac{\partial Y_n^m(\theta,\phi)}{\partial\theta}. \end{aligned}$$

The electric field due to the local potential is computed similarly.

4.2.2 Addition theorems

In the fast multipole algorithm we need to translate the origin of a multipole or local expansion and also need to switch between multipole and local transformations. The translation formulas are based on the addition theorems for spherical wave functions [18, 40, 8]. The addition theorems are derived here in a form that uses the Clebsch-Gordan or Wigner coefficients $C_{m_1m_2m}^{j_1j_2j}$ [3].

Let us consider spherical waves

$$h_n^{(1)}(kr)P_n^m(\cos\theta)e^{im\phi} \tag{4.21}$$

and

$$j_n(kr)P_n^m(\cos\theta)e^{im\phi}. \tag{4.22}$$

The origin of the coordinate system is O and the waves are evaluated at a point P with spherical coordinates (r,θ,ϕ). We shall give expansions of spherical waves in a new coordinate system with origin O' that has coordinates (r_0,θ_0,ϕ_0) with respect to O. Let (r',θ',ϕ') be the coordinates of P with respect to O'. The coordinate systems are shown in Figure 4.1.

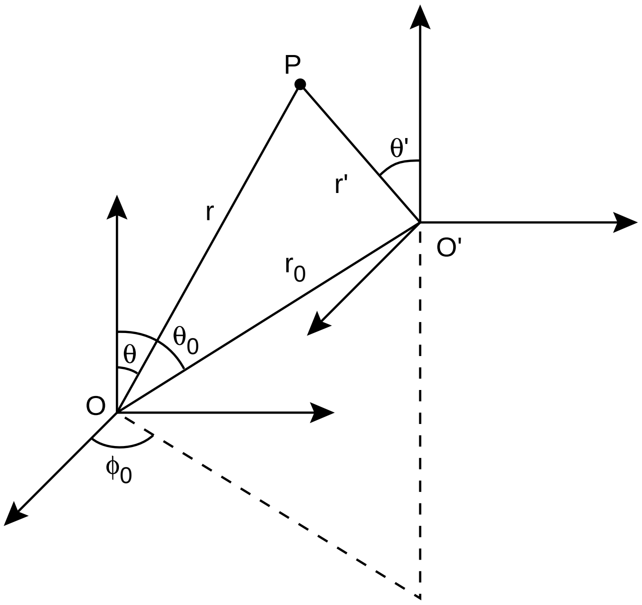

Figure 4.1: The origins of the two coordinate systems [8].

The three expansions that we need are [8]:

$$\begin{aligned} h_n^{(1)}(kr)P_n^m(\cos\theta)e^{im\phi} &= \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p(-1)^\mu i^{\nu+p-n}(2\nu+1)a(m,n,|\mu,\nu|p)\cdot \\ &\quad j_\nu(kr_0)P_\nu^{-\mu}(\cos\theta_0)e^{i\mu\phi_0}h_p^{(1)}(kr')P_p^{m+\mu}(\cos\theta')e^{i(m+\mu)\phi'}, \end{aligned} \tag{4.23}$$

$$\begin{aligned} h_n^{(1)}(kr)P_n^m(\cos\theta)e^{im\phi} &= \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p(-1)^\mu i^{\nu+p-n}(2\nu+1)a(m,n,|\mu,\nu|p)\cdot \\ &\quad j_\nu(kr')P_\nu^{-\mu}(\cos\theta')e^{i\mu\phi'}h_p^{(1)}(kr_0)P_p^{m+\mu}(\cos\theta_0)e^{i(m+\mu)\phi_0}, \end{aligned} \tag{4.24}$$

and

$$\begin{aligned} j_n(kr)P_n^m(\cos\theta)e^{im\phi} &= \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p(-1)^\mu i^{\nu+p-n}(2\nu+1)a(m,n,|\mu,\nu|p)\cdot \\ &\quad j_\nu(kr')P_\nu^{-\mu}(\cos\theta')e^{i\mu\phi'}j_p(kr_0)P_p^{m+\mu}(\cos\theta_0)e^{i(m+\mu)\phi_0}. \end{aligned} \tag{4.25}$$

The first expansion is valid for $r' \geq r_0$, the two following are valid for $r' \leq r_0$. In the expansions, p is summed over $n+\nu$, $n+\nu-2$, ..., $\max(|n-\nu|, |m+\mu|)$.

Here the coefficients $a(m,n,|\mu,\nu|p)$ are given by

$$\begin{aligned} a(m,n,|\mu,\nu|p) &= (-1)^{m+\mu}(2p+1)\left[\frac{(n+m)!(\nu+\mu)!(p-m-\mu)!}{(n-m)!(\nu-\mu)!(p+m+\mu)!}\right]^{1/2}\cdot \\ &\quad \begin{pmatrix} n & \nu & p \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} n & \nu & p \\ m & \mu & -m-\mu \end{pmatrix}, \end{aligned} \tag{4.26}$$

where the Wigner $3-j$ symbols are defined

$$\begin{pmatrix} j_1 & j_2 & j \\ m_1 & m_2 & -m \end{pmatrix} = (-1)^{j_1-j_2+m}(2j+1)^{-1/2}C_{m_1m_2m}^{j_1j_2j}. \tag{4.27}$$

The Clebsch-Gordan or Wigner coefficients are defined by [3]

$$C^{j_1 j_2 j}_{m_1 m_2 m} = \delta(m_1+m_2-m)\left[\frac{(2j+1)(j+j_1-j_2)!(j-j_1+j_2)!(j_1+j_2-j)!}{(j+j_1+j_2+1)!}\right]^{1/2}\cdot$$
$$\left[\frac{(j+m)!(j-m)!}{(j_1+m_1)!(j_1-m_1)!(j_2+m_2)!(j_2-m_2)!}\right]^{1/2}\cdot$$
$$\sum_s \frac{(-1)^{j_2+m_2+s}(j_2+j+m_1-s)!(j_1-m_1+s)!}{s!(j-j_1+j_2-s)!(j+m-s)!(j_1-j_2-m+s)!}, \tag{4.28}$$

where s takes the values: $\max(0, m_1+m_2+j_2-j_1) \le s \le \min(j-j_1+j_2, j+m_1+m_2)$.

When we insert definitions (4.26) and (4.27) into Equation (4.23), and make use of the definition of the spherical harmonics, we get

$$h^{(1)}_n(kr)Y^m_n(\theta,\phi) = \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p (-1)^{\mu} i^{\nu+p-n}\sqrt{\frac{4\pi(2n+1)(2\nu+1)}{2p+1}}\cdot$$
$$C^{n\nu p}_{000}C^{n\nu p}_{m,\mu,m+\mu}j_\nu(kr_0)Y^{-\mu}_\nu(\theta_0,\phi_0)h^{(1)}_p(kr')Y^{m+\mu}_p(\theta',\phi'). \tag{4.29}$$

Now change the summation order of p and ν and make a change of variable $t=\mu+m$. Then relabel $\nu \leftrightarrow p$ and $t \leftrightarrow \mu$:

$$h^{(1)}_n(kr)Y^m_n(\theta,\phi) = \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p (-1)^{m+\mu} i^{\nu+p-n}\sqrt{\frac{4\pi(2n+1)(2p+1)}{2\nu+1}}\cdot$$
$$C^{np\nu}_{000}C^{np\nu}_{m,\mu-m,\mu}h^{(1)}_\nu(kr')Y^{\mu}_\nu(\theta',\phi')j_p(kr_0)Y^{m-\mu}_p(\theta_0,\phi_0). \tag{4.30}$$

With the transformation

$$C^{np\nu}_{m,\mu-m,\mu} = C^{n\nu p}_{m,-\mu,m-\mu}(-1)^{n-m}\sqrt{(2\nu+1)(2p+1)} \tag{4.31}$$

we finally obtain

$$h^{(1)}_n(kr)Y^m_n(\theta,\phi) = \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p (-1)^{\mu} i^{\nu+p-n}\sqrt{\frac{4\pi(2n+1)(2\nu+1)}{2p+1}}\cdot$$
$$C^{n\nu p}_{000}C^{n\nu p}_{m,-\mu,m-\mu}h^{(1)}_\nu(kr')Y^{\mu}_\nu(\theta',\phi')j_p(kr_0)Y^{m-\mu}_p(\theta_0,\phi_0). \tag{4.32}$$

The expansions (4.24) and (4.25) are also simplified using the spherical harmonics and definitions (4.26) and (4.27):

$$h^{(1)}_n(kr)Y^m_n(\theta,\phi) = \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p (-1)^{\mu} i^{\nu+p-n}\sqrt{\frac{4\pi(2n+1)(2\nu+1)}{2p+1}}\cdot$$
$$C^{n\nu p}_{000}C^{n\nu p}_{m,-\mu,m-\mu}j_\nu(kr')Y^{\mu}_\nu(\theta',\phi')h^{(1)}_p(kr_0)Y^{m-\mu}_p(\theta_0,\phi_0), \tag{4.33}$$

$$j_n(kr)Y^m_n(\theta,\phi) = \sum_{\nu=0}^{\infty}\sum_{\mu=-\nu}^{\nu}\sum_p (-1)^{\mu} i^{\nu+p-n}\sqrt{\frac{4\pi(2n+1)(2\nu+1)}{2p+1}}\cdot$$
$$C^{n\nu p}_{000}C^{n\nu p}_{m,-\mu,m-\mu}j_\nu(kr')Y^{\mu}_\nu(\theta',\phi')j_p(kr_0)Y^{m-\mu}_p(\theta_0,\phi_0). \tag{4.34}$$

Now Equations (4.32), (4.33), and (4.34) are the addition theorems that we need in the fast multipole calculations. In (4.32) $r' \ge r_0$, in the other two $r' \le r_0$. In the expansions, p is summed over $n+\nu$, $n+\nu-2$, ..., $\max(|n-\nu|, |\mu-m|)$.

4.2.3 Transformations

The multipole expansion of the vector potential is

$$\mathbf{A}(\mathbf{r}) = 4\pi k^2 \sum_{nm} \mathbf{M}_{nm} h_n^{(1)}(kr) Y_n^m(\theta, \phi) \tag{4.35}$$

and the local expansion is

$$\mathbf{A}(\mathbf{r}) = 4\pi k^2 \sum_{nm} \mathbf{L}_{nm} j_n(kr) Y_n^m(\theta, \phi). \tag{4.36}$$

Now we state the transformation theorems for these expansions based on the addition formulas (4.32), (4.33), and (4.34).

In the transformation formulas, we use coefficients $c_{n\nu}^{m\mu p}$ defined by

$$c_{n\nu}^{m\mu p} = (-1)^m i^{n-\nu+p} \sqrt{\frac{4\pi(2n+1)(2\nu+1)}{2p+1}} C_{000}^{\nu np} C_{\mu,-m,\mu-m}^{\nu np} \tag{4.37}$$

or alternatively by

$$\begin{aligned} c_{n\nu}^{m\mu p} &= (-1)^\mu i^{n-\nu+p} \sqrt{4\pi(2n+1)(2\nu+1)(2p+1)} \cdot \\ &\quad \begin{pmatrix} \nu & n & p \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \nu & n & p \\ \mu & -m & m-\mu \end{pmatrix}. \end{aligned} \tag{4.38}$$

The first transformation is used to translate the origin of a multipole expansion. The new origin is given in spherical coordinates by (r_0, θ_0, ϕ_0).

$$\mathbf{M}'_{nm} = \sum_{\nu=0}^{\infty} \sum_{\mu=-\nu}^{\nu} \mathbf{M}_{\nu\mu} \sum_p c_{n\nu}^{m\mu p} j_p(kr_0) Y_p^{\mu-m}(\theta_0, \phi_0). \tag{4.39}$$

The second transformation is used to switch between a multipole and a local expansion.

$$\mathbf{L}'_{nm} = \sum_{\nu=0}^{\infty} \sum_{\mu=-\nu}^{\nu} \mathbf{M}_{\nu\mu} \sum_p c_{n\nu}^{m\mu p} h_p^{(1)}(kr_0) Y_p^{\mu-m}(\theta_0, \phi_0). \tag{4.40}$$

Finally, this transformation is used to translate the origin of a local expansion.

$$\mathbf{L}'_{nm} = \sum_{\nu=0}^{\infty} \sum_{\mu=-\nu}^{\nu} \mathbf{L}_{\nu\mu} \sum_p c_{n\nu}^{m\mu p} j_p(kr_0) Y_p^{\mu-m}(\theta_0, \phi_0). \tag{4.41}$$

In the above formulas, the summation over p takes the values $n+\nu, n+\nu-2, \ldots, \max(|m-\mu|, |n-\nu|)$. The transformations are exact, but in practice they are truncated in a similar way as the potential expansions.

4.2.4 The fast multipole algorithm

The fast multipole algorithm describes an economical way of computing the electromagnetic interactions between the dipoles. The interactions between near-by dipoles are computed directly, the distant interactions are computed using truncated multipole and local expansions. In the algorithm, the computational domain is adaptively divided into boxes. The interaction between dipoles in two distant boxes is computed by forming the multipole expansions of the dipoles in the boxes, transforming the expansions into local expansions with respect of the origin of the other box and evaluating the local expansions at the dipole positions.

The fast multipole algorithm organizes the evaluation of the dipole interactions so that the computations are made using the coarsest refinement level possible and maintaining a rigorous error bound. The multipole expansions are used only at boxes that are separated from the box whose multipole coefficients are used by a distance of at least the side length of the box. In this case, the ratio R/R_1 in the error bound (4.17) is smaller than $1/\sqrt{3}$. The same argument applies also to the accuracy of local expansions. Now a level of precision ϵ is obtained if $n_m = \mathcal{O}(-\log(\epsilon)/\log(\sqrt{3}))$ terms are used in the multipole calculations. In [38] the ratio R/R_1 is said to be smalled than $1/2$ but clearly this is not the case for a three-dimensional problem.

We will review the fast multipole algorithm, as given in [6]. We shall be working with computational boxes b containing some number of dipoles. Each box is either a *parent box*, if it is further divided in the algorithm, or a *childless box*. *Colleagues* are adjacent boxes at the same refinement level. Each box has at most 26 colleagues.

With each box b we define five lists of other boxes that are used in the fast multipole algorithm.

U_b: If b is childless, U_b consists of b and all childless boxes that are adjacent to b. If b is a parent box, U_b is empty.

V_b: List V_b consists of all the children of colleagues of b's parent that are well separated from b.

W_b: If b is a childless box, W_b consists of all descendants of b's colleagues whose parents are adjacent to b but who are not adjacent to b themselves. If b is a parent box, W_b is empty.

X_b: List X_b consists of all boxes c such that $b \in W_c$.

Y_b: List Y_b consists of all boxes that are well separated from b's parent.

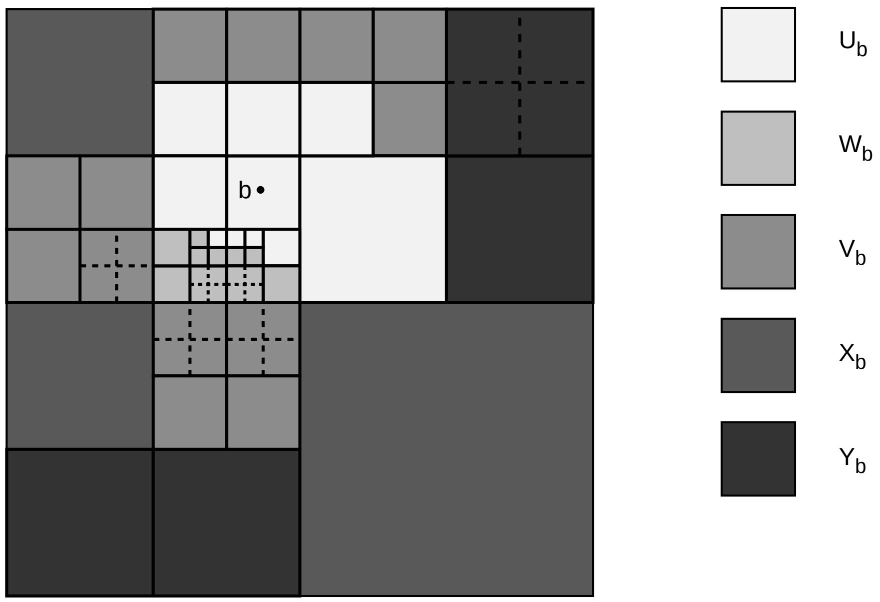

Figure 4.2: The lists of boxes in the fast multipole algorithm used in calculating the effect of all particles on the particles in box b.

The lists are illustrated in Figure 4.2. The dashed line signifies that the multipole or local expansion of the whole box is used rather than that of its children.

In the algorithm, the effect of particles in the different lists on b are computed as follows: The effect of particles in U_b is evaluated directly. For the boxes in V_b, first form the multipole expansion of the particles (either directly or by translating the multipole expansion of children to the centers of their parent boxes) and translate this into a local expansion about the center of b and then evaluate this in the end. For the boxes in W_b, first form the multipole expansions of each box and then evaluate these at the particles in b. For particles in X_b, form the local expansions of the boxes at the center of b and evaluate this in the end.

The boxes Y_b are not used directly in the algorithm, rather their effect is computed by first forming the multipole expansions of each box, then translating this into a local expansion about the center of some of b's ancestors and then transforming this information downwards into a local expansion about the center of b. The local expansions about the center of b are either evaluated directly at each particle in b (if b is childless) or are translated to local expansions about the centers of b's children boxes.

The fast multipole algorithm is as follows:

1. Refine the computational domain adaptively into boxes. If a box contains more than B_{max} dipoles and the current refinement level does not exceed lev_{max}, the

box is subdivided into eight child boxes of equal size.

2. Form the multipole coefficients (4.12) for each childless box. For each parent box, use (4.39) to translate the origin of the multipole expansions of each child to the center of the parent box and add the expansions to get the multipole expansion of the parent box.

3. For each childless box b, compute directly the interactions between the dipoles in b with the dipoles in boxes U_b.

4. For each box b, convert the multipole expansions of each box in V_b into a local expansion about b's center using (4.40), and add them up.

5. For each childless box b, calculate using (4.19) at each dipole position the electric field due to the multipole expansion of each box in W_b.

6. Form the local expansion coefficients (4.14) of each particle in the boxes X_b about the center of b.

7. For each parent box b, shift the center of the local expansion obtained in stages 4 and 6 to the center of b's children using (4.41).

8. For each childless box, calculate at all dipole positions the electric field due to the local expansion from stage 7 using a formula similar to (4.19) and add the contributions from stages 3 and 5 to get the full electric field caused by all the other dipoles.

A major part of the computing time is spent calculating the transformations (4.39)–(4.41). The FMM algorithm can be optimized by precomputing these. There are only a fixed number of possible transformations. The multipole-to-multipole transformations are always from the child boxes to their parent box, the local-to-local transformations are from a parent box to its children. There are 189 possible multipole-to-local transformations in each refinement level, as indicated in [38]. In our code we have precomputed all the possible transformations (4.39)–(4.41).

We could also tabulate the values of the special functions that are needed when forming the multipole coefficients (4.12), but these consume only a marginal part of the CPU time of the algorithm as compared to the transformations (4.39)–(4.41). The evaluation of the electric field by (4.19) is quite an expensive operation that could be optimized in stage 8 of the multipole algorithm. At stage 5 the same evaluation is difficult to optimize because there are so many possibilities of the parameters of the evaluation. For the same reason, the formation of the local coefficients (4.14) in stage 6 was not optimized.

N_{dip}	n_L	t_1	t_2	t_3	M_1	M_2	M_3
136	216	0.07	0.06	0.01	2.2	2.4	6.8
304	512	0.3	0.2	0.07	2.5	3.0	22
1064	1728	3.7	0.4	0.8	3.7	5.8	300
2320	4096	18	1.3	-	5.8	10	-
5232	10648	90	2.4	-	11	27	-

Table 4.1: The matrix-vector computation times and memory usages for several pseudospherical geometries and for three different ways to compute the matrix-vector product. In the table, N_{dip} is the number of dipoles and n_L is the number of lattice points for the FFT algorithm. The matrix-vector computation times t_i are given in CPU seconds and memory usage M_i in megabytes. The matrix-vector products $A\mathbf{x}$ are formed by recomputing the rows of the matrix A when necessary (t_1 and M_1), by the special FFT algorithm (t_2 and M_2), or by keeping the matrix in memory (t_3 and M_3).

4.3 Performance

The performance of the different ways to compute the matrix-vector product is compared. At first, the results for the fast multipole method are not included because the FMM algorithm is quite inefficient for small systems.

For small geometries, the fastest way to compute the matrix-vector product is to form the coefficient matrix and keep it in memory. In double precision, this is practical up to 1300 dipoles, after which the coefficient matrix takes too much central memory. If the geometry is quite dense, such as a pseudosphere, the fast Fourier transform becomes the fastest way of applying the coefficient matrix to a vector in system with more than a few thousand unknowns. The recomputation of the matrix entries is quite slow. The CPU times and the memory usages to compute a single matrix-vector product for several pseudospherical geometries are shown in Table 4.1. The index of refraction was $m = 1.6 + 0.05i$ and the size parameter was $x_0 = 0.3$.

When the geometry is quite sparse, such as sparse cluster or a fractal cluster from the diffusion-limited aggregation process, the FFT algorithm needs many artificial dipoles. In Table 4.2, a comparison of the computation times and memory usages of the matrix-vector products for several sparse cluster geometries is given. Again, the fastest way to compute the matrix-vector product is to compute the coefficient matrix and keep it in the memory. For larger geometries, the memory usage of the full matrix becomes restrictive. In these sparse geometries the FFT algorithm and recomputing the matrix entries are many times of comparable speed.

N_{dip}	n_L	t_1	t_2	t_3	M_1	M_2	M_3
289	4096	0.3	1.2	0.07	2.5	6.6	21
1024	4096	3.3	1.2	0.7	3.7	7.8	298
782	13824	2.0	3.2	0.4	3.3	20	153
1462	13824	6.8	3.2	-	4.4	21	-
676	46656	1.6	11	0.3	3.1	69	83
2756	46656	26	11	-	6.6	72	-

Table 4.2: The matrix-vector computation times and memory usages for several sparse cluster geometries and for three different ways to compute the matrix-vector product. The meaning of the column labels is the same as in Table 4.1.

4.3.1 FMM

The speed and accuracy of the fast multipole method (FMM) are evaluated in this section. The accuracy of FMM depends on the number of terms n_m that are used in the multipole expansions. The performance of FMM depends on n_m, on the number of dipoles, and on the parameters B_{max} and lev_{max}. Table 4.3 shows the relative error that FMM makes in a single matrix-vector multiplication, and the computation time for two different geometries and for various values of n_m. The error falls off quite rapidly with n_m. The relative error for the solution of the linear system is of the same order of magnitude than the relative error for the matrix-vector multiplication. For these geometries, the FFT takes 1.2 and 2.4 seconds, respectively, to compute the matrix-vector product.

Next we will examine what is the effect of the parameter B_{max}, the maximum number of particles in a box, on the performance. Tables 4.4–4.6 give the number of boxes after the division, the total number of boxes in all lists U_b, V_b, W_b, and X_b, and the CPU time. Table 4.4 gives these number for a pseudosphere of 2320 dipoles, table 4.5 for a pseudosphere of 5232 dipoles, and table 4.6 for a DLA cluster of 3000 dipoles. In all the tables lev_{max}=5 and $n_m = 4$.

For example in Table 4.4 the execution time and the division into boxes stays constant when B_{max} is between 20 and 50. After two recursive divisions of the initial box, the number of particles in each childless box is approximately the same. The third recursive division into eight subboxes occurs when B_{max} crosses this average number (about 60 in the example). In table 4.6 the execution time does not vary very much with B_{max}, although the division changes. This is probably due to the fractal nature of the DLA clusters.

n_m	$N_{dip} = 2320$	$N_{dip} = 5232$
2	$\epsilon = 5.8 \cdot 10^{-2}$ $t = 10$	$\epsilon = 0.1$ $t = 35$
3	$\epsilon = 1.5 \cdot 10^{-2}$ $t = 14$	$\epsilon = 4.9 \cdot 10^{-2}$ $t = 59$
4	$\epsilon = 2.6 \cdot 10^{-3}$ $t = 17$	$\epsilon = 1.4 \cdot 10^{-2}$ $t = 102$
5	$\epsilon = 2.4 \cdot 10^{-4}$ $t = 23$	$\epsilon = 2.6 \cdot 10^{-3}$ $t = 163$
6	$\epsilon = 7.4 \cdot 10^{-5}$ $t = 30$	$\epsilon = 4.5 \cdot 10^{-4}$ $t = 239$

Table 4.3: Relative errors (ϵ) and computation times (t) (in CPU seconds) of the FMM algorithm for a single matrix-vector product. The geometries are two pseudospheres. The factor n_m gives the maximum number of terms in the multipole expansion. In this simulation, B_{max} was 10 and lev_{max} was 3 and 4.

B_{max}	N_{box}	$\lvert U_b \rvert$	$\lvert V_b \rvert$	$\lvert W_b \rvert$	$\lvert X_b \rvert$	t
10	425	9448	36744	272	272	71
20	329	7168	26928	1160	1160	107
30	329	7168	26928	1160	1160	109
50	329	7168	26928	1160	1160	108
60	137	2656	6192	920	920	128
70	73	1088	3096	0	0	18
90	73	1088	3096	0	0	18

Table 4.4: The effect of B_{max} on the performance of FMM in the case of a pseudosphere of 2320 dipoles. In the table, N_{box} means the number of boxes produced by the division algorithm and $|U_b|$ means the number of boxes in all lists U_b, and t is the CPU time in seconds.

B_{max}	N_{box}	$\lvert U_b \rvert$	$\lvert V_b \rvert$	$\lvert W_b \rvert$	$\lvert X_b \rvert$	t
40	473	10600	43032	272	272	106
120	329	7168	26928	1160	1160	215
170	73	1088	3096	0	0	49

Table 4.5: Same as Table 4.4 but for a pseudosphere of 5232 dipoles.

B_{max}	N_{box}	$\lvert U_b \rvert$	$\lvert V_b \rvert$	$\lvert W_b \rvert$	$\lvert X_b \rvert$	t
80	148	2754	6148	747	747	113
90	143	2660	5820	735	735	118
100	133	2457	4894	749	749	126
110	122	2269	4238	720	720	131
120	106	1919	3146	630	630	134
140	91	1586	2236	537	537	131
160	83	1413	1790	476	476	125

Table 4.6: Same as Table 4.4 but for a DLA particle of 3000 dipoles.

It seems that the performance of QMR slows down when there are many particles in the lists W_b and X_b. This is easily understood if we look at the operations that are needed when processing each list V_b, W_b, and X_b. For V_b the operations are multipole to local transformations which have been optimized. For W_b we have to evaluate the multipole expansions of each box in W_b at each particle in b. These operations have not been optimized. For X_b we have to form local expansions of each particle in X_b about the center of b and this operation has not been optimized either.

A large number of particles in the lists W_b and X_b is caused by an uneven division of the computational domain into boxes so that different childless boxes are on different refinement levels. This can happen if the value B_{max} is near the critical value where many computational boxes are divided into subboxes. At that point the boxes near the center of the pseudosphere have been divided further but the boxes near the surface of the sphere have not.

A better way to divide the computational domain into subdomains is to set B_{max} to a small number, say 10, and to specify the maximum refinement level lev_{max} that is used in the division. In this algorithm, the number of particles in each box varies but most of the childless boxes are on the same refinement level.

Tables 4.7–4.9 show the performance of FMM when lev_{max} is given suitable values. Table 4.7 gives the performance for a pseudosphere of 2320 dipoles, table 4.8 for a pseudosphere of 5232 dipoles, and table 4.9 for a DLA cluster of 3000 dipoles. In all the tables B_{max}=5 and $n_m = 4$.

In these tables the number of boxes in lists W_b and X_b is clearly less than in the algorithm were B_{max} was varied. The performance is at least at good as for the B_{max} algorithm, in the DLA case it is much better.

lev_{max}	N_{box}	$\|U_b\|$	$\|V_b\|$	$\|W_b\|$	$\|X_b\|$	t
3	73	1088	3096	0	0	18
4	425	9448	36744	272	272	70

Table 4.7: The effect of lev_{max} on the performance of FMM in the case of a pseudosphere of 2320 dipoles. The meaning of the other columns is the same as in Table 4.4.

lev_{max}	N_{box}	$\|U_b\|$	$\|V_b\|$	$\|W_b\|$	$\|X_b\|$	t
3	73	1088	3096	0	0	48
4	473	10600	43032	272	272	102

Table 4.8: Same as Table 4.7, but for the case of a pseudosphere of 5232 dipoles.

lev_{max}	N_{box}	$\|U_b\|$	$\|V_b\|$	$\|W_b\|$	$\|X_b\|$	t
3	52	730	1214	0	0	26
4	241	4461	13658	166	166	40

Table 4.9: Same as Table 4.7, but for the case of a DLA cluster of 3000 dipoles.

Next we will show some performance figures when FMM is compared with FFT. In all these comparisons $B_{max} = 10$. Also the memory usage of the algorithms is shown. Table 4.10 shows the computation time for pseudospheres. Table 4.11 shows the performance for three sparse cluster geometries. The clusters consist of 100, 200, and 300 pseudospheres of 32 dipoles.

In the pseudosphere geometries, the fast Fourier transform is an order of magnitude faster than the FMM algorithm. On the other hand, the FFT algorithm consumes about twice the amount of central memory as the FMM algorithm. The FMM becomes faster than direct calculation at approximately 2000 dipoles, as can be seen from Tables 4.1 and 4.10.

For the sparse clusters, FMM and FFT are of comparable speed for the smallest cluster of 3200 dipoles. In larger clusters the computation time for FFT remains virtually constant while that of FMM grows. The FFT algorithm needs a regular lattice that contains all the dipoles. In the three cluster geometries the FFT lattice is virtually of the same size as can be seen by looking at the parameter n_L. For the cluster geometries, the FMM algorithm needs considerably less memory than

N	lev_{max}	t_{FMM}	M_{FMM}	t_{FFT}	M_{FFT}	n_L	ϵ_{rel}
2320	3	17.5	10	1.2	10	4096	$2.6 \cdot 10^{-3}$
5232	3	48.2	15	2.5	27	10648	$1.3 \cdot 10^{-2}$
10048	3	130	23	4.5	35	17576	$2.3 \cdot 10^{-2}$
10048	4	121	31	4.5	35	17576	$2.4 \cdot 10^{-2}$
20336	3	549	40	11	101	39304	$4.7 \cdot 10^{-2}$
20336	4	213	48	11	101	39304	$4.7 \cdot 10^{-2}$
137376	4	5106	241	72	491	262144	0.24
137376	5	1885	279	72	491	262144	0.24

Table 4.10: The performance and accuracy of FMM as compared to FFT in pseudospherical geometries. The execution times for a single matrix-vector multiplication t_{FMM} and t_{FFT} are in CPU seconds. The memory usages M_{FMM} and M_{FFT} are in megabytes, and n_L is the number of lattice points for the FFT algorithm. The relative error of the FMM algorithm is given by ϵ_{rel}.

N	lev_{max}	t_{FMM}	M_{FMM}	t_{FFT}	M_{FFT}	n_L	ϵ_{rel}
3200	3	23.7	12	20.7	73	73728	$2.3 \cdot 10^{-2}$
6400	3	65.2	17	22.6	144	92160	$3.6 \cdot 10^{-2}$
6400	4	67.5	25	22.6	144	92160	$3.8 \cdot 10^{-2}$
9600	3	123.5	22	22.1	149	92160	$4.3 \cdot 10^{-2}$
9600	4	102.1	30	22.1	149	92160	$4.4 \cdot 10^{-2}$

Table 4.11: Same as Table 4.10, but for three sparse cluster geometries.

FFT. Also in the sparse DLA geometry described by Table 4.9 the FMM and FFT algorithms are of comparable speed.

As an extremely sparse geometry, consider a cluster of 66 pseudospheres each containing 66 dipoles, a total of 4356 dipoles, where the distances between the dipoles have been doubled from their normal value. For this geometry, the FMM algorithm needs 32 seconds and it needs 13 megabytes of memory. The estimated computing time for the FFT is about 2 minutes with a memory usage of over 500 megabytes. The lattice size would be over 370 000 particles for the FFT algorithm.

In practice, we observed that the computing time for the FMM algorithm grows like $N \log N$, where N is the number of dipoles. The FMM algorithm is reported to have $\mathcal{O}(N)$ time dependence in [6], but there the authors assume that the number of divisions lev_{max} is constant and is given by the machine precision.

In conclusion, the fastest way to compute the matrix-vector product for systems of up to 1500 dipoles is to form the coefficient matrix and keep it in memory. If the dipoles sit on a regular lattice and the dipoles fill most of the lattice positions, the fast Fourier transform is the fastest method and can be used to solve the DDA equations for up to 130 000 dipoles.

For very sparse geometries or for irregular geometries where the FFT algorithm cannot be used, the fast multipole method is the fastest one. On the other hand, the speed of the FMM algorithm depends heavily on the accuracy needed in the calculations.

In this chapter, most of the results were obtained by setting n_m to 4 in the FMM algorithm. This means that the computation error is of the order of one per cent, that is unacceptable in many computational problems. In the scattering calculations where orientational averages are taken this kind of error might be acceptable in some cases.

Chapter 5

Conclusion

In this thesis we have described how the system of dense linear equations from scattering computations can be efficiently solved. First the integral equation describing the scattering of electromagnetic radiation from a particle is presented. It is shown that the discretized integral equation is closely related to a technique called the discrete-dipole approximation, where the particle is replaced by interacting electric dipoles.

An area of future research would be to investigate the possibility of using more accurate ways to discretize the scattering integral equation. Specifically, higher order basis functions and more accurate integration methods could be used in the Galerkin method.

In Chapter 3 the solution of dense linear equations is discussed. The iterative methods conjugate gradient, GMRES, QMR, block QMR, BCG, CGS, and BiCGStab were implemented and tested. The complex symmetric version of QMR was found to be the best method in this setting. Also, the block version of QMR can accelerate convergence by a factor of three.

The spectra of several coefficient matrices were calculated. By potential-theoretic arguments described in [30] we were able to estimate the convergence of iterative methods. The estimates proved to be quite accurate. It would be interesting to calculate the spectrum of the operators in the integral equation and to compare that spectrum to the discrete one. This would also serve as a benchmark for the quality of the discretization of the integral equation.

In Chapter 4 we discussed the efficient computation of the matrix-vector product. There are several straightforward methods that require $\mathcal{O}(n^2)$ operations. In addition to these we implemented a fast algorithm based on the fast Fourier transform

(FFT) that can be used to compute the matrix-vector product if the dipoles are on a regular lattice. The FFT algorithm requires $\mathcal{O}(n_L \log n_L)$ operations, where n_L is the number of points in the lattice. The FFT algorithm is very efficient if the number of dipoles is of the same order of magnitude as the number of lattice points. We were able to solve systems of linear equations with over 400 000 unknowns with the FFT algorithm using a couple of CPU hours on a Convex C3840.

We also implemented another fast algorithm for the matrix-vector product, the fast multipole method (FMM). The FMM is an $\mathcal{O}(N \log N)$ algorithm, where N is the number of dipoles. We derived the potential expansions and the transformation formulas that are needed in the FMM algorithm and optimized the computer implementation of FMM. The rigorous error bounds of the FMM should be computed.

The performance of various methods of calculating the matrix-vector product was evaluated. For large problems where the geometry is quite regular, the FFT algorithm in unbeatable. For very sparse geometries the FMM algorithm becomes competitive. However, the FMM is only an approximate method and it becomes quite expensive if the solution of the linear system is needed to a high accuracy.

In the course of this research a computer code was implemented that allows one to compute the scattering of light from a single particle. This code is among the fastest programs of its kind. The program has been applied to astrophysical scattering calculations to obtain new scientific results [28].

Although the emphasis of this thesis has been on the solution of linear equations arising from the DDA formalism, the methods described here are readily applied to the integral equation formalism of scattering. Similar techniques can be applied to other problems in scientific computing where the physics is described by an integral equation. Such problems arise for example in the calculation of electromagnetic fields.

Bibliography

[1] R. Barrett et al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, 1993.

[2] L. P. Bayvel and A. R. Jones. *Electromagnetic scattering and its applications*. Applied Science Publishers, London, 1981.

[3] L. C. Biedenharn and J. D. Louck. *Angular Momentum in Quantum Physics*. Addison-Wesley, Reading, Massachusetts, 1981.

[4] C. F. Bohren and D. R. Huffman. *Absorption and Scattering of Light by Small Particles*. John Wiley & Sons, New York, 1983.

[5] W. E. Boyse and A. A. Seidl. A sparse matrix iterative method for efficiently computing multiple simultaneous solutions. *SIAM J. Sci. Comput.*, submitted.

[6] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM J. Sci. Stat. Comput.* **9**, No. 4, 669–686, 1988.

[7] R. Coifman, V. Rokhlin, and S. Wandzura. The fast multipole method for the wave equation: a pedestrian approach. *IEEE Antennas and Propagation Magazine* **35**, No. 3, 7–12, June 1993.

[8] O. Cruzan. Translational addition theorems for spherical vector wave functions. *Quart. Appl. Math.* **20**, 33–40, 1962.

[9] B. T. Draine. The discrete-dipole approximation and its application to interstellar graphite grains. *Astrophys. J.* **333**, 848–872, 1988.

[10] A. Edelman. Large dense numerical linear algebra in 1993: the parallel computing influence. *Int. J. of Supercomputer Applications* **7**, No. 2, 113–128, 1993.

[11] N. Engheta, W. D. Murphy, V. Rokhlin, and M. S. Vassiliou. The fast multipole method (FMM) for electromagnetic scattering problems. *IEEE Trans. on Antennas and Propagation* **40**, No. 6, 634–641, 1992.

[12] V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.* **21**, No. 2, 352–362, 1984.

[13] V. Faber and T. Manteuffel. Orthogonal error methods. *SIAM J. Numer. Anal.* **24**, No. 1, 170–187, 1987.

[14] P. J. Flatau, G. L. Stephens, and B. T. Draine. Light scattering by rectangular solids in the discrete-dipole approximation: a new algorithm exploiting the block-Toeplitz structure. *J. Opt. Soc. Am. A* **7**, No. 4, 593–600, 1990.

[15] R. W. Freund. Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices. *SIAM J. Sci. Stat. Comput.* **13**, No. 1, 425–448, 1992.

[16] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative solution of linear systems. In *Acta Numerica 1992*, pages 57–100. Cambridge University Press, 1992.

[17] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik* **60**, 315–339, 1991.

[18] B. Friedman and J. Russek. Addition theorems for spherical waves. *Quart. Appl. Math.* **12**, 13–23, 1954.

[19] G. H. Goedecke and S. G. O'Brien. Scattering by irregular inhomogeneous particles via the digitized green's function algorithm. *Applied Optics* **27**, No. 12, 2431–2438, 1988.

[20] J. J. Goodman, B. T. Draine, and P. J. Flatau. Application of fast-Fourier-transform techniques to the discrete-dipole approximation. *Optics Letters* **16**, No. 15, 1198–1200, 1991.

[21] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comp. Phys.* **73**, 325–348, 1987.

[22] J. Hage. *The optics of porous particles and the nature of comets.* PhD thesis, University of Leiden, 1990.

[23] J. D. Jackson. *Classical Electrodynamics.* Wiley, New York, 1983.

[24] A. Lakhtakia and G. W. Mulholland. On two numerical techniques for light scattering by dielectric agglomerated structures. *J. Res. Natl. Inst. Stand. Technol.* **98**, No. 6, 699–716, 1993.

[25] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Nat. Bur. Standards* **45**, 255–282, 1950.

[26] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Nat. Bur. Standards* **49**, 33–53, 1952.

[27] D. E. Livesay and K.-M. Chen. Electromagnetic fields induced inside arbitrarily shaped biological bodies. *IEEE Trans. on Microwave Theory Tech.* **22**, No. 12, 1273–1280, 1974.

[28] K. Lumme and J. Rahola. Light scattering by porous dust particles in the discrete-dipole approximation. *Astrophys. J.* **425**, 653–667, April 1994.

[29] P. Meakin. Diffusion-controlled cluster formation in two, three, and four dimensions. *Phys. Rev. A* **27**, 604–607, 1983.

[30] O. Nevanlinna. *Convergence of Iterations for Linear Equations.* Birkhäuser, Basel, 1993.

[31] D. P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra Appl.* **29**, 293–322, 1980.

[32] D. P. O'Leary. Parallel implementations of the block conjugate gradient algorithm. *Parallel Computing* **5**, 127–140, 1987.

[33] J. Peltoniemi. Electromagnetic scattering by cosmic grains using discretized volume integral equation method. Unpublished manuscript, 1994.

[34] J. Rahola. Solution of dense systems of linear equations in the discrete-dipole approximation. CSC Preprint Series 1/1994, Center for Scientific Computing, 1994. Submitted to SIAM J. Sci. Comput.

[35] V. Rokhlin. Rapid solution of integral equations of classical potential theory. *J. Comp. Phys.* **60**, 187–207, 1985.

[36] V. Rokhlin. Rapid solution of integral equations of scattering theory in two dimensions. *J. Comp. Phys.* **86**, 414–439, 1990.

[37] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* **7**, No. 3, 856–869, 1986.

[38] K. E. Schmidt and M. A. Lee. Implementing the fast multipole method in three dimensions. *J. of Statistical Physics* **63**, No. 56, 1223–1235, 1991.

[39] P. Sonneveld. CGS, A fast lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* **10**, No. 1, 36–52, 1989.

[40] S. Stein. Addition theorems for spherical wave functions. *Quart. Appl. Math.* **19**, 15–24, 1961.

[41] H. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.* **13**, 631–644, 1992.

[42] H. Voss. *Iterative methods for linear systems of equations.* Luentomoniste 27. Jyväskylän yliopisto, matematiikan laitos, 1993.

[43] A. D. Yaghijan. Electric dyadic Green's functions in the source region. *Proceedings of the IEEE* **68**, No. 2, 248–263, 1980.