Chapter 14

Near-to-Far-Field Transformation

14.1 Introduction

As we have seen, the FDTD method provides the fields throughout some finite region of space, i.e., the fields throughout the computational domain. However, in practice, we are often interested in the fields far away from the region we have modeled. For example, an FDTD implementation may have modeled an antenna or some scatterer. But, the fields in the immediate vicinity of that antenna or scatterer may not be the primary concern. Rather, the distant or “far” fields may be the primary concern. In this chapter we show how the near fields, which are essentially the fields within the FDTD grid, can be used to obtain the far fields. We start with a brief review of the underlying theory that pertains in the continuous world and then discuss the implementation details for the FDTD method.

14.2 The Equivalence Principle

Recall the boundary conditions that pertain to the electric and magnetic fields tangential to an interface:

\[ \hat{n}' \times (E_1 - E_2) = -M_s, \]  
\[ \hat{n}' \times (H_1 - H_2) = J_s, \]  

(14.1) (14.2)

where \( \hat{n}' \) is normal to the interface, pointing toward region 1. The subscript 1 indicates the fields immediately adjacent to one side of the interface and the subscript 2 indicates the fields just on the other side of the interface. The “interface” can either be a physical boundary between two media or a fictitious boundary with the same medium to either side. The current \( M_s \) is a magnetic surface current, i.e., a current that only flows tangential to the interface. In practice there is no magnetic charge and thus no magnetic current. Therefore (14.1) states that the tangential components of \( E \) must be continuous across the boundary. However, in theory, we can imagine a scenario where the tangential fields are discontinuous. If this were the case, the magnetic current \( M_s \) must be non-zero to account for this discontinuity. In a little while we will see why it is convenient to envision such a scenario. The current \( J_s \) in (14.2) is the usual electric surface current.
As depicted in Fig. 14.1(a), consider a space in which there is a source or scatterer that radiates (or scatters) some fields. We can define a fictitious boundary that surrounds this source or scatterer. Let us then imagine that the fields exterior to this boundary are unchanged but the fields interior to the boundary are set to zero as depicted in Fig. 14.1(b). By setting the fields interior to the boundary to zero, we will create discontinuities in the tangential components on either side of the fictitious boundary. These discontinuities are perfectly fine provided we account for them by having the appropriate surface currents flow over the boundary. These currents are given by (14.1) and (14.2) where the fields in region 2 are now zero. Thus,

\[
M_s = -\mathbf{n}' \times \mathbf{E}_1, \quad (14.3)
\]

\[
J_s = \mathbf{n}' \times \mathbf{H}_1. \quad (14.4)
\]

As you may recall and as will be discussed in further detail below, it is fairly simple to find the fields radiated by a current (whether electric or magnetic) when that current is radiating in a homogeneous medium. Unfortunately, as shown in Fig. 14.1(b), the surface currents are not radiating in a homogeneous medium. But, in fact, since the fields within the fictitious boundary are zero, we can place anything, or nothing, within the boundary and that will have no effect on the fields exterior to the boundary. So, let us maintain the same surface currents but discard any inhomogeneity that were within the boundary. That leaves a homogeneous region as depicted in Fig. 14.1(c) and it is fairly straightforward to find these radiated fields.

### 14.3 Vector Potentials

Before proceeding further, let us briefly review vector potentials. First, consider the case (which corresponds to the physical world) where there is no magnetic charge—electric currents can flow but magnetic currents cannot. Thus,

\[
\nabla \cdot \mathbf{B}_A = \nabla \cdot \mu \mathbf{H}_A = 0 \quad (14.5)
\]

where the subscript \( A \) indicates we are considering the case of no magnetic charge. There is a vector identity that the divergence of the curl of any vector field is identically zero. Therefore (14.5) will automatically be satisfied if we write

\[
\mathbf{H}_A = \frac{1}{\mu} \nabla \times \mathbf{A} \quad (14.6)
\]

where \( \mathbf{A} \) is a yet-to-be-determined field known as the magnetic vector potential. Now, using Faraday’s law we obtain

\[
\nabla \times \mathbf{E}_A = -j\omega \mu \mathbf{H}_A = -j\omega \mathbf{H}_A = -j\omega \nabla \times \mathbf{A}. \quad (14.7)
\]

Using the terms on the left and the right and regrouping yields

\[
\nabla \times (\mathbf{E}_A + j\omega \mathbf{A}) = 0 \quad (14.8)
\]

The curl of the gradient of any function is identically zero. Thus we can set the term in parentheses equal to the (negative of the) gradient of some unknown scalar electric potential function \( \Phi_e \) and in this way (14.8) will automatically be satisfied. Therefore we have

\[
\mathbf{E}_A + j\omega \mathbf{A} = -\nabla \Phi_e \quad (14.9)
\]
14.3. VECTOR POTENTIALS

Figure 14.1: (a) A space containing a source or scatterer that is surrounded by a fictitious boundary which is indicated by the dashed line. The fields are continuous across this boundary. (b) The fields are set to zero within the boundary. Surface currents must be used to account for the discontinuity across the boundary. (c) Since the fields are zero within the boundary, any inhomogeneities within the boundary can be discarded.
or, after rearranging,

\[ E_A = -j\omega A - \nabla \Phi_e. \] (14.10)

Using the remaining curl equation, Ampere’s law, we can write

\[ \nabla \times H_A = J + j\omega \epsilon E_A \] (14.11)
\[ \nabla \times \frac{1}{\mu} \nabla \times A = J + j\omega \epsilon (-j\omega A - \nabla \Phi_e) \] (14.12)

Multiplying through by \( \mu \) and expanding the curl operations yields

\[ \nabla (\nabla \cdot A) - \nabla^2 A = \mu J + \omega^2 \mu \epsilon A - j\omega \mu \epsilon \nabla \Phi_e. \] (14.13)

Regrouping terms yields

\[ \nabla^2 A + \omega^2 \mu \epsilon A = -\mu J + \nabla (\nabla \cdot A + j\omega \mu \epsilon \Phi_e). \] (14.14)

So far we have said what the curl of \( A \) must be, but that does not fully describe the field. To fully describe a vector field one must specify the curl, the divergence, and the value at a point (which we will ultimately assume is zero at an infinite distance from the origin). We are free to make the divergence of \( A \) any convenient value. Let us use the “Lorentz gauge” of

\[ \nabla \cdot A = -j\omega \mu \epsilon \Phi_e. \] (14.15)

By doing this, (14.14) reduces to

\[ \nabla^2 A + k^2 A = -\mu J. \] (14.16)

where \( k = \omega \sqrt{\mu \epsilon} \).

Distinct from the scenario described above, let us imagine a situation where there is no free electric charge. Magnetic currents can flow, but electric currents cannot. Thus the divergence of the electric flux density is

\[ \nabla \cdot D_F = \nabla \cdot \epsilon E_F = 0 \] (14.17)

where here the subscript \( F \) is used to indicate the case of no electric charge. Again, this equation will be satisfied automatically if we represent the electric field as the curl of some potential function \( F \). To this end we write

\[ E_F = -\frac{1}{\epsilon} \nabla \times F \] (14.18)

where \( F \) is known as the electric vector potential.

Following steps similar to the ones we used to obtain (14.16), one can obtain the differential equation that governs \( F \), namely,

\[ \nabla^2 F + k^2 F = -\epsilon M. \] (14.19)

Thus both \( A \) and \( F \) are governed by the wave equation. We see that the source of \( A \), i.e., the forcing function that creates \( A \) is the electric current \( J \). Similarly, the source of \( F \) is the magnetic current \( M \). (We have not yet restricted these currents to be surface currents. At this point they can be any current distribution, whether distributed throughout a volume, over a surface, or along a line.)

Note that both the Laplacian (\( \nabla^2 \)) and the constant \( k^2 \) that appear in (14.16) and (14.19) are scalar operators. They do not change the orientation of a vector. Thus, the \( x \) component of \( J \) gives
rise to the \( x \) component of \( A \), the \( y \) component of \( M \) gives rise to the \( y \) component of \( F \), and so on. In this way, (14.16) and (14.19) could each be broken into their three Cartesian components and we would be left with six scalar equations.

These equations have relatively straightforward solutions. Let us consider a slightly simplified problem, the solution of which can easily be extended to the full general problem. Consider the case of an incremental current of length \( d\ell \) that is located at the origin and oriented in the \( z \) direction. In this case (14.16) reduces to

\[
\nabla^2 A_z(r) + k^2 A_z(r) = -\mu I d\ell \delta(r) \tag{14.20}
\]

where \( I \) is the amount of current and \( \delta(r) \) is the 3D Dirac delta function. The Dirac delta function is zero except when its argument is zero. For an argument of zero, \( \delta(r) \) is singular, i.e., infinite. However, this singularity is integrable. A volume integral of any region of space that includes the Dirac delta function at the origin (i.e., \( r = 0 \)) will yield unit volume. For any observation point other than the origin, (14.20) can be written

\[
\nabla^2 A_z(r) + k^2 A_z(r) = 0 \quad r \neq 0. \tag{14.21}
\]

It is rather easy to show that a general solution to this is

\[
A_z(r) = C_1 \frac{e^{-jkr}}{r} + C_2 \frac{e^{jkr}}{r}. \tag{14.22}
\]

We discard the second term on the right-hand side since that represents a spherical wave propagating in toward the origin. Thus we are left with

\[
A_z(r) = C_1 \frac{e^{-jkr}}{r} \tag{14.23}
\]

where we must now determine the constant \( C_1 \) based on the “driving function” on the right side of (14.20).

To obtain \( C_1 \), we integrate both sides of (14.20) over a small spherical volume of radius \( r_0 \) and take the limit as \( r_0 \) approaches zero:

\[
\lim_{r_0 \to 0} \int_V \left[ \nabla^2 A_z + k^2 A_z \right] dv = \lim_{r_0 \to 0} \int_V -\mu I d\ell \delta(r) dv. \tag{14.24}
\]

Using the sifting property of the delta function, the right-hand side of (14.24) is simply \(-\mu I d\ell \). For the left-hand side, we first expand the integral associated with the second term in the square brackets

\[
\lim_{r_0 \to 0} \int_{r=0}^{r_0} \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} k^2 C_1 \frac{e^{-jkr}}{r} r^2 \sin \theta \, d\phi \, d\theta \, dr. \tag{14.25}
\]

Including the term \( r^2 \sin \theta \), which is contributed by the volume element \( dv \), the entire integrand is proportional to \( r \). Therefore as \( r_0 \) (the upper limit of integration in \( r \)) goes to zero, this integral goes to zero.
The remaining integral on the left side of (14.24) is

$$\lim_{r_0 \to 0} \int_V \nabla \cdot \nabla A_z \, dv = \lim_{r_0 \to 0} \oint_S \nabla A_z \cdot \mathbf{ds}$$

(14.26)

where we have used the divergence theorem to convert the volume integral to a surface integral (and used the fact that $\nabla^2 = \nabla \cdot \nabla$). The integrand of the surface integral is given by

$$\nabla A_z \bigg|_{r=r_0} = \hat{r} \frac{\partial A_z}{\partial r} \bigg|_{r=r_0} = \hat{r} \left( -\frac{e^{-jkr_0}}{r_0^2} - jk \frac{e^{-jkr_0}}{r_0} \right) C_1.$$  

(14.27)

The surface element $\mathbf{ds}$ is given by $\hat{r} r_0^2 \sin \theta d\phi d\theta$ so that the entire surface integral is given by

$$\lim_{r_0 \to 0} \int_{\theta=0}^{\pi} \int_{\phi=0}^{2\pi} C_1 e^{-jkr_0} (-1 - jkr_0) \sin \theta d\phi d\theta = \lim_{r_0 \to 0} -C_1 e^{-jkr_0} (1 + jkr_0) 4\pi = -C_1 4\pi.$$  

(14.28)

Equating this with the right-hand side of (14.20), we can solve for $C_1$. The final result is

$$C_1 = \frac{\mu I d\ell}{4\pi}.$$  

(14.29)

It should be noted that there is actually nothing special about having $r_0$ approach zero. The same coefficient is obtained for any value of $r_0$. Letting $r_0$ approach zero merely simplifies the problem a bit.

We now have that a filamentary current $I$ of length $d\ell$ located at the origin and oriented in the $z$ direction produces the vector potential

$$A_z(r) = \frac{\mu I d\ell e^{-jkr}}{4\pi r}.$$  

(14.30)

This is simply a spherical wave radiating symmetrically away from the origin. If the source is located at the point $r'$ instead of the origin, one merely needs to account for this displacement. The vector potential in that case is

$$A_z(r) = \frac{\mu I d\ell e^{-jk|r-r'|}}{4\pi |r - r'|}.$$  

(14.31)

If the current was oriented in the $x$ or $y$ direction, that would produce a vector potential that only had a $x$ or $y$ component, respectively.

For a current source, the “strength” of the source is, in one way of thinking, determined by the amount of current that is flowing times the length over which that current flows. For a filament we have that the “source strength” is given by $I d\ell$. For a surface current the equivalent concept is $J_s \, ds$ where $J_s$ is the surface current density in Amperes/meter and $ds$ is an incremental surface area. Similarly, for a volumetric current, the equivalent term is $J \, dv$ where $J$ is the current density in Amperes/meter$^2$ and $dv$ is an incremental volume.

Instead of having just a point source, currents can be distributed throughout space. To get the corresponding vector potentials, we merely have to sum the contributions of the current wherever it
exists, accounting for the location (displacement from the origin), the orientation, and the amount of current.

For surface currents, the vector potentials are given by

$$A(r) = \mu \oint_S J_s(r') \frac{e^{-jk|r-r'|}}{4\pi|r - r'|} ds', \quad (14.32)$$

$$F(r) = \epsilon \oint_S M_s(r') \frac{e^{-jk|r-r'|}}{4\pi|r - r'|} ds', \quad (14.33)$$

where, as before, $r$ is the observation point, $r'$ is the location of the source point (i.e., the location of the currents), and $S$ is the surface over which the current flows.

Equations (14.32) and (14.33) give the vector potentials at an arbitrary observation point in three dimensions. The surface $S$ is a surface that exists in the 3D space (such as the surface of a sphere or a cube).

Let us consider the two-dimensional case. We can consider the 2D case as a special case in 3D in which there is no variation in the $z$ direction. The observation point is a point in the $xy$ plane specified by the vector $\rho$. Thus, the magnetic vector potential could be written as

$$A(\rho) = \mu \oint_L J_s(\rho') \frac{e^{-jk|\rho - \rho'|^2 + z'^2}}{4\pi \sqrt{|\rho - \rho'|^2 + z'^2}} d\rho' \, dz', \quad (14.34)$$

$$F(\rho) = \epsilon \oint_L M_s(\rho') \frac{e^{-jk|\rho - \rho'|^2 + z'^2}}{4\pi \sqrt{|\rho - \rho'|^2 + z'^2}} d\rho' \, dz'. \quad (14.35)$$

The term in parentheses can be integrated to obtain

$$\int_{z'=-\infty}^{\infty} \frac{e^{-jk|\rho - \rho'|^2 + z'^2}}{4\pi \sqrt{|\rho - \rho'|^2 + z'^2}} dz' = -\frac{j}{4} H_0^{(2)}(k|\rho - \rho'|) \quad (14.36)$$

where $H_0^{(2)}$ is the zeroth-order Hankel function of the second kind. This represents a cylindrical wave radiating from the point $\rho'$. Similar steps can be done for $F$ and in this way $z$ is eliminated from the expressions for the vector potentials. We are left with a 2D representation of the fields, namely,

$$A(\rho) = -\frac{j}{4} \mu \oint_L J(\rho') H_0^{(2)}(k|\rho - \rho'|) d\rho', \quad (14.37)$$

$$F(\rho) = -\frac{j}{4} \epsilon \oint_L M(\rho') H_0^{(2)}(k|\rho - \rho'|) d\rho'. \quad (14.38)$$

where the explicit $s$ subscript has been dropped from the currents.

The approximation of the zeroth-order Hankel function of the second kind as the argument $\xi$ gets large is

$$H_0^{(2)}(k\xi) \approx \sqrt{\frac{j^2}{\pi k\xi}} e^{-jk\xi}. \quad (14.39)$$
Now let $\xi = |\rho - \rho'|$ where $\rho$ is large enough that the following approximations are valid:

$$\xi \approx \begin{cases} \rho - \rho' \cos \psi & \text{for the phase}, \\ \rho & \text{for the magnitude.} \end{cases} \quad (14.40)$$

Thus the Hankel function can be written

$$H_0^{(2)}(k|\rho - \rho'|) \approx \sqrt{\frac{j2}{\pi k \rho}} e^{-jk\rho e^{j k \rho' \cos \psi}}. \quad (14.41)$$

The $A$ vector potential for a 2D problem can thus be written

$$A(\rho) = -j\mu \frac{1}{4} \oint_L S(\rho') H_0^{(2)}(k|\rho - \rho'|) d\ell', \quad (14.42)$$

$$\approx -j\mu \frac{1}{4} \sqrt{\frac{j2}{\pi k \rho}} e^{-jk\rho} \oint_L S(\rho') e^{jk\rho' \cos \psi} d\ell', \quad (14.43)$$

$$= -j\mu \frac{1}{4} \sqrt{\frac{j2}{\pi k \rho}} e^{-jk\rho} N_{2D}, \quad (14.44)$$

where

$$N_{2D} = \oint_L S(\rho') e^{jk\rho' \cos \psi} d\ell'. \quad (14.45)$$

Correspondingly, the $F$ vector potential can be written

$$F(\rho) = -j\epsilon \frac{1}{4} \oint_L S(\rho') H_0^{(2)}(k|\rho - \rho'|) d\ell', \quad (14.46)$$

$$\approx -j\epsilon \frac{1}{4} \sqrt{\frac{j2}{\pi k \rho}} e^{-jk\rho} \oint_L S(\rho') e^{jk\rho' \cos \psi} d\ell', \quad (14.47)$$

$$= -j\epsilon \frac{1}{4} \sqrt{\frac{j2}{\pi k \rho}} e^{-jk\rho} L_{2D}, \quad (14.48)$$

where

$$L_{2D} = \oint_L S(\rho') e^{jk\rho' \cos \psi} d\ell'. \quad (14.49)$$

Nominally $N_{2D}$ and $L_{2D}$ are functions of $\rho$. However, within these functions the only thing that depends on $\rho$ is $\psi$. The angle $\psi$ only changes for large changes in $\rho$—incremental changes of $\rho$ will not affect $\psi$. Thus, derivatives of $N_{2D}$ or $L_{2D}$ with respect to $\rho, \phi$, or $z$ (i.e., derivatives with respect to the unprimed coordinates) are zero. The geometry is depicted in Fig. 14.2.

It is convenient to think of the currents, and subsequently $N_{2D}$ and $L_{2D}$ (and ultimately the potentials), in terms of cylindrical coordinates, i.e.,

$$J(\rho) = J_\rho \hat{\rho} + J_\phi \hat{\phi} + J_z \hat{z}, \quad (14.50)$$

$$M(\rho) = M_\rho \hat{\rho} + M_\phi \hat{\phi} + M_z \hat{z}. \quad (14.51)$$
Combining the contributions from both \( A \) and \( F \), the electric and magnetic fields are given by

\[
E(\rho) = -j\omega \left[ A + \frac{1}{k^2} \nabla (\nabla \cdot A) \right] - \frac{1}{\varepsilon} \nabla \times F, \quad (14.52)
\]

\[
H(\rho) = -j\omega \left[ F + \frac{1}{k^2} \nabla (\nabla \cdot F) \right] + \frac{1}{\mu} \nabla \times A. \quad (14.53)
\]

By plugging (14.44) and (14.48) into (14.52) and (14.53) and performing the various operations in cylindrical coordinates and discarding any terms that fall off faster than \( 1/\sqrt{\rho} \), one can obtain expressions for the electric and magnetic fields in the far field. (Note that the \( \nabla \) operator acts on the unprimed coordinates and, as mentioned above, \( N_{2D} \) and \( L_{2D} \) are not considered functions of the unprimed coordinates.)

### 14.4 Electric Field in the Far-Field

Following the steps outlined in the previous section, the scattered electric field \( E_s(\rho) \) at the far-field point \( \rho \) can be obtained from the scattered “near” fields using

\[
E_s(\rho) = \sqrt{\frac{j}{8\pi k}} \frac{e^{-jk\rho}}{\sqrt{\rho}} \left\{ \hat{a}_\phi (\omega \mu_0 N_{2D} \cdot \hat{a}_\phi + kL_{2D} \cdot \hat{a}_z) - \hat{a}_z (\omega \mu_0 N_{2D} \cdot \hat{a}_z - kL_{2D} \cdot \hat{a}_\phi) \right\},
\]

where, as stated previously,

\[
N_{2D} = \oint_L J(\rho')e^{jk\rho' \cos \psi} \, dl', \quad (14.55)
\]

\[
L_{2D} = \oint_L M(\rho')e^{jk\rho' \cos \psi} \, dl', \quad (14.56)
\]
CHAPTER 14. NEAR-TO-FAR-FIELD TRANSFORMATION

$L$ is the closed path of integration, $\psi$, given by $\phi - \phi'$, is the angle between the source point and observation point, $M = -\hat{n}' \times E$, $J = \hat{n}' \times H$, and $\hat{n}'$ is a unit vector normal to the integration contour on the same side of the contour as the observation point (i.e., the outward normal). Unprimed coordinates correspond to the observation point while primed coordinates indicate the “source” location (i.e., points along the contour).

Let us now restrict consideration to TM$^z$ polarization where the non-zero field are $H_x$, $H_y$, and $E_z$. Since the outward normal is restricted to exist in the $xy$ plane, $\hat{n}' \times H$ only has a non-zero component in the $z$ direction while $-\hat{n}' \times E$ only has non-zero components in the $xy$ plane. Thus, for this polarization only the $z$ component of the electric field is non-zero—the $\phi$ component of (14.54) is zero. The electric field can be written

$$E_z^s(\rho) = -\sqrt{\frac{j}{8\pi k}} \frac{e^{-jk\rho}}{\sqrt{\rho}} \oint_L (\omega\mu_0 J(\rho') \cdot \hat{a}_z - kM(\rho') \cdot \hat{a}_\phi) e^{ik\rho' \cos\psi} d\rho'. \quad (14.57)$$

Usually the scattering width is of more interest than the field itself. For TM$^z$ polarization the two-dimensional scattering width is defined to be

$$\sigma_{2D}(\phi) = \lim_{|\rho - \rho'| \to \infty} \frac{2\pi \rho |E_z^s(\rho)|^2}{|E_z|^2} \quad (14.58)$$

Noting that $\omega\mu_0 = k\eta_0$ and plugging (14.57) into (14.58) and normalizing by the wavelength yields

$$\frac{\sigma_{2D}(\phi)}{\lambda} = \frac{1}{8\pi|E_z|^2} \left| \oint_L \{\eta_0 J(\rho') \cdot \hat{a}_z - M(\rho') \cdot \hat{a}_\phi\} e^{ik\rho' \cos(\phi - \phi')} kd\rho' \right|^2. \quad (14.59)$$

The term $r' \cos(\phi - \phi')$ which appears in the exponent can be written as $\hat{a}_\phi \cdot \rho = \hat{a}_\phi \cdot (x'\hat{a}_x + y'\hat{a}_y) = x' \cos \phi + y' \sin \phi$. This last form is especially useful since $\phi$ is fixed by the observation direction and therefore the sine and cosine functions can be determined outside of any loop (rather than over and over again as we move along the integration contour).

For TM$^z$ polarization the unit vector normal to the integration path is restricted to lie in the $xy$ plane, i.e., $\hat{n}' = n'_x \hat{a}_x + n'_y \hat{a}_y$ where $(n'_x^2 + n'_y^2)^{1/2} = 1$. Thus, the electric current $J$ is given by

$$J = \hat{n}' \times H = \begin{vmatrix} \hat{a}_x & \hat{a}_y & \hat{a}_z \\ n'_x & n'_y & 0 \\ H_x & H_y & 0 \end{vmatrix} = \hat{a}_z(n_x' H_y - n_y' H_x). \quad (14.60)$$

The dot product of $\hat{a}_z$ and $J$ yields

$$J \cdot \hat{a}_z = n'_x H_y - n'_y H_x. \quad (14.61)$$

The magnetic current is given by

$$M = -\hat{n}' \times E = \begin{vmatrix} \hat{a}_x & \hat{a}_y & \hat{a}_z \\ n'_x & n'_y & 0 \\ 0 & 0 & E_z \end{vmatrix} = -(\hat{a}_x n'_y E_z - \hat{a}_y n'_x E_z). \quad (14.62)$$
14.4. ELECTRIC FIELD IN THE FAR-FIELD

\[ y = n' \Delta y \]

\[ (0, L_y \Delta y) \]

\[ \hat{n}' = \hat{a}_y \]

\[ (L_x \Delta x, L_y \Delta y) \]

\[ \hat{n}' = -\hat{a}_x \]

\[ \text{Side } L_1 \]

\[ (0, 0) \]

\[ \hat{n}' = -\hat{a}_y \]

\[ x = m' \Delta x \]

\[ \hat{n}' = \hat{a}_x \]

\[ \text{Side } L_3 \]

\[ (L_x \Delta x, 0) \]

\[ \hat{n}' = \hat{a}_y \]

\[ \text{Side } L_4 \]

\[ \hat{n}' = -\hat{a}_x \]

\[ \text{Side } L_2 \]

Figure 14.3: Depiction of integration boundary for near-to-far-field transformation in the FDTD grid.

The dot product of \( \mathbf{\hat{a}} \phi \) and \( \mathbf{M} \) yields

\[ \mathbf{M} \cdot \mathbf{\hat{a}} \phi = -\hat{a}_x \cdot \mathbf{\hat{a}} \phi n'_y E_z + \hat{a}_y \cdot \mathbf{\hat{a}} \phi n'_x E_z = (n'_y \sin \phi + n'_x \cos \phi) E_z \]  

(14.63)

Incorporating (14.61) and (14.63) into (14.59) yields a general expression for the scattering width:

\[ \frac{\sigma_{2D}(\phi)}{\lambda} = \frac{1}{8\pi |E_z|^2} \left| \oint_{L} \{ \eta_0(n'_x H_y - n'_y H_x) - (n'_y \sin \phi + n'_x \cos \phi) E_z \} e^{jkr' \cos(\phi - \phi')} kd\ell' \right|^2 \]  

(14.64)

We now want to specialize this equation to a rectangular box which is typical of the integration boundary which would be employed in an FDTD simulation.

Assume the integration boundary corresponds to the dashed box shown in Fig. 14.3. The width of this rectangle is \( L_x \Delta x \) and the height is \( L_y \Delta y \). In an FDTD grid there are \( L_x + 1 \) samples of the fields along the top and bottom (i.e., spanning the width) and \( L_y + 1 \) total samples along the left and right (i.e., spanning the height). The integration over the closed path \( L \) consists of the integration over the four sides of this box, i.e., \( L = L_1 + L_2 + L_3 + L_4 \). Using this geometry, the quantities needed to perform each integral are presented in the following two tables.

<table>
<thead>
<tr>
<th>Side</th>
<th>( n'_x )</th>
<th>( n'_y )</th>
<th>( J \cdot \mathbf{\hat{a}}_z )</th>
<th>( \mathbf{M} \cdot \mathbf{\hat{a}}_\phi )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L_1 )</td>
<td>(-1)</td>
<td>(0)</td>
<td>(-H_y)</td>
<td>(-\cos \phi E_z)</td>
</tr>
<tr>
<td>( L_2 )</td>
<td>(0)</td>
<td>(1)</td>
<td>(-H_x)</td>
<td>(\sin \phi E_z)</td>
</tr>
<tr>
<td>( L_3 )</td>
<td>(1)</td>
<td>(0)</td>
<td>(H_y)</td>
<td>(\cos \phi E_z)</td>
</tr>
<tr>
<td>( L_4 )</td>
<td>(0)</td>
<td>(-1)</td>
<td>(H_x)</td>
<td>(-\sin \phi E_z)</td>
</tr>
</tbody>
</table>
Assume we wish to integrate the function \( f(x) \) over the interval \( 0 \leq x \leq L \) where \( L \) is an even integer. The integral can be obtained using Simpson’s composite integration as follows

\[
\int_{0}^{L} f(x)dx \approx \frac{1}{3} \left[ f(0) + 2 \sum_{m=1}^{L/2-1} f(2m) + 4 \sum_{m=1}^{L/2} f(2m-1) + f(L) \right]
\] (14.66)
14.6 Collocating the Electric and Magnetic Fields: The Geometric Mean

Figure 14.4: Depiction of a TM grid showing the integration boundary. The boundary is assumed to be aligned with electric-field nodes. The expanded views show the offset of the magnetic-field nodes from the boundary.

Note that this approximation requires a total of \( L + 1 \) samples of the function (so we need an odd number of samples). Using Simpson’s approximation yields quite a bit of additional accuracy over what would be obtained using a straight Riemann sum and it costs essentially nothing (it just requires slightly more bookkeeping).

14.6 Collocating the Electric and Magnetic Fields: The Geometric Mean

Figure 14.4 depicts an integration boundary in a TM grid. The boundary is assumed to be aligned with the electric-field nodes, i.e., the \( E_z \) nodes. The expanded views show a portion of the boundary along the right side and the bottom. The field notation employs superscripts to indicate time steps while spatial indices are given as arguments within parentheses. Half-step spatial offsets are implicitly understood. Thus, the nodes in space-time and the corresponding notation are

\[
H_x^{(q-1/2)\Delta t} (m \Delta_x, [n + 1/2] \Delta_y) = H_x^{q-1/2} (m, n),
\]

\[
H_y^{(q-1/2)\Delta t} ([m + 1/2] \Delta_x, n \Delta_y) = H_y^{q-1/2} (m, n),
\]

\[
E_z^{q\Delta t} (m \Delta_x, n \Delta_y) = E_z^q (m, n),
\]

where \( \Delta_x \) and \( \Delta_y \) are the spatial steps in the \( x \) and \( y \) directions, respectively, and \( \Delta_t \) is the temporal step. The index \( q \) indicates the temporal step and we assume it varies between 1 and \( N_T \) which is the total number of time-steps.

Near-to-far-field (NTFF) transforms require that the fields be defined over a single surface and use the same phase reference. For harmonic fields, the temporal offset can be easily accounted for with a phase factor. Assume the magnetic fields have been recorded at times of
The frequency domain. For example, the harmonic electric field on the boundary is given by

\[ \hat{E}_z^k(m, n) = \mathcal{F}[E_z^q(m, n)] , \]

\[ = \frac{1}{N_T} \sum_{q=\langle N_T \rangle} E_z^q(m, n) e^{-jk\frac{2\pi q}{N_T}} , \]  

(14.71)

where \( \mathcal{F} \) is the discrete Fourier transform. For situations where the entire spectrum is not of interest, typically a running discrete Fourier transform will be used only at the particular frequencies of interest. A frequency is specified by the index \( k \) which varies between zero (dc) and \( N_T - 1 \). Regardless of the implementation used, the resulting spectral terms \( \hat{E}_z^k \) will be the same.

The time-domain series \( E_z^q(m, n) \) can be obtained from \( \hat{E}_z^k(m, n) \) via

\[ E_z^q(m, n) = \sum_{k=\langle N_T \rangle} \hat{E}_z^k(m, n) e^{jk\frac{2\pi q}{N_T}} . \]  

(14.72)

Because of the temporal offset between the electric and magnetic fields, the desired field is actually \( E_z^{q-1/2}(m, n) \), thus, plugging \( q - 1/2 \) into (14.72) yields

\[ E_z^{q-1/2}(m, n) = \sum_{k=\langle N_T \rangle} \left( \hat{E}_z^k(m, n) e^{-jk\frac{\pi q}{N_T}} \right) e^{jk\frac{2\pi q}{N_T}} . \]  

(14.73)

In practice one calculates \( \hat{E}_z^k(m, n) \) and the spectral representation of the magnetic fields in the same way, i.e., as in (14.71). Then one multiplies \( \hat{E}_z^k(m, n) \) by \( \exp(-jk\pi/N_T) \) to account for the temporal offset.

The spatial offset is slightly more problematic than the temporal offset. As shown in Fig. 14.4, the integration boundary can be aligned with only one of the fields. The magnetic field tangential to the integration boundary is found from the nodes that are a spatial half-step to either side of the boundary.

To obtain the magnetic field on the boundary, the traditional approach has been to use a spatial average of the nodes to either side of the boundary. For example, along the right side of the boundary, the harmonic magnetic field would be given by

\[ \hat{H}_y^k(m, n) = \frac{1}{2} \mathcal{F} \left[ H_y^{q-1/2}(m - 1, n) + H_y^{q-1/2}(m, n) \right] . \]  

(14.74)

Because of this spatial average, \( \hat{H}_y(m, n) \) and \( \hat{E}_z(m, n) \) are assumed to be collocated and, with the temporal phase correction, can be used to determine the equivalent currents over the integration boundary (which are then used in the NTFF transform itself).

Unfortunately the arithmetic mean used in (14.74) introduces errors. To illustrate this, assume a harmonic plane wave is propagating in the grid. The temporal frequency \( \omega \) is \( 2\pi k'/N_T \) where \( k' \) is an integer constant and, as before, \( N_T \) is the total number of time-steps in a simulation. The \( y \) component of the magnetic field is given by

\[ H_y^{q-1/2}(m, n) = \cos \left( \omega \left[ q - \frac{1}{2} \right] \Delta_t - \xi \right) \]

\[ = \frac{e^{j(k'\frac{2\pi}{N_T}[q-\frac{1}{2}])\Delta_t + \xi}}{2} + e^{-j(k'\frac{2\pi}{N_T}[q-\frac{1}{2}])\Delta_t + \xi} . \]  

(14.76)
where $\xi = \beta_x(m + 1/2)\Delta_x + \beta_y n \Delta_y$, and $\beta_x$ and $\beta_y$ are the $x$ and $y$ components of the wave vector, respectively. Taking the discrete Fourier transform of (14.76), i.e.,

$$
\hat{H}_y^k(m, n) = \frac{1}{N_T} \sum_{q=(N_T)} H_y^{q-1/2}(m, n) e^{-jk \frac{2\pi}{N_T}(q-\frac{1}{2})},
$$

(14.77)

one notes that the sum yields zero when $k$ is anything other than $k'$ or $N_T - k'$. The values of $k$ that yield non-zero correspond to the positive and negative frequency of the continuous world and, like the continuous world, the corresponding spectral values are complex conjugates. Without loss of generality, we will continue the discussion in terms of the spectral component corresponding to the positive frequency, i.e.,

$$
\hat{H}_y^k(m, n) = \frac{1}{2} \exp(-j[\beta_x(m + 1/2)\Delta_x + \beta_y n \Delta_y]).
$$

(14.78)

(Note that since the time-domain functions are real-valued, in practice one does not need to calculate the transform at any of the negative frequencies. They are merely the complex conjugates of the values at the positive frequencies.)

Because the Fourier transform is a linear operator, using (14.76) in (14.74) yields

$$
\hat{H}_y^k(m, n) = e^{-j\beta_y n \Delta_y} e^{-j\beta_x (m-\frac{1}{2}) \Delta_x} + e^{-j\beta_x (m+\frac{1}{2}) \Delta_x}.
$$

(14.79)

$$
= \frac{1}{2} e^{-j(\beta_x m \Delta_x + \beta_y n \Delta_y)} \cos \left( \frac{\beta_x \Delta_x}{2} \right).
$$

(14.80)

The exact expression for the magnetic field on the integration boundary is $\exp(-j[\beta_x m \Delta_x + \beta_y n \Delta_y])/2$. Thus, the cosine term represents an error—one which vanishes only in the limit as the spatial-step size goes to zero.

Instead of taking the Fourier transform of the average of the time-domain fields, let us take the Fourier transform of the fields to either side of the boundary. We define the transforms as

$$
\hat{H}_y^+(m, n) = \mathcal{F}[H_y^{q-1/2}(m, n)],
$$

(14.81)

$$
\hat{H}_y^-(m, n) = \mathcal{F}[H_y^{q-1/2}(m - 1, n)].
$$

(14.82)

Still assuming a single harmonic plane wave, for the “positive frequency” corresponding to $k = k'$, $\hat{H}_y^+(m, n)$ and $\hat{H}_y^-(m, n)$ are given by

$$
\hat{H}_y^+(m, n) = \frac{1}{2} e^{-j(\beta_x (m+1/2) \Delta_x + \beta_y n \Delta_y)},
$$

(14.83)

$$
\hat{H}_y^-(m, n) = \frac{1}{2} e^{-j(\beta_x (m-1/2) \Delta_x + \beta_y n \Delta_y)}.
$$

(14.84)

Were one to calculate the arithmetic mean of $\hat{H}_y^+(m, n)$ and $\hat{H}_y^-(m, n)$, the result would be the same as given in (14.80). However, consider the geometric mean (where the geometric mean of $a$ and $b$ is $\sqrt{ab}$) of $\hat{H}_y^+(m, n)$ and $\hat{H}_y^-(m, n)$:

$$
\hat{H}_y^k(m, n) = \left( \hat{H}_y^+(m, n) \hat{H}_y^-(m, n) \right)^{1/2},
$$

(14.85)

$$
= \frac{1}{2} e^{-j(\beta_x m \Delta_x + \beta_y n \Delta_y)}.
$$

(14.86)
This is precisely the correct answer. There is no error introduced by the geometric mean. A note of caution: when calculating the square root of these complex quantities, one must ensure that the proper branch cut is selected. Thus, when $\hat{H}_y^+(m, n)$ and $\hat{H}_y^-(m, n)$ have phases near $\pm \pi$ the geometric mean should also have a phase near $\pm \pi$ rather than near zero.

In practice, at any given frequency there will be an angular spectrum of wave vectors present and hence any averaging, whether geometric or arithmetic, will introduce some errors. However, for a single wave vector the geometric mean is exact and it has been our experience that the geometric mean provides superior results for nearly all discretizations and scattering angles. The following section demonstration the use of the geometric mean in several scenarios.

### 14.7 NTFF Transformations Using the Geometric Mean

#### 14.7.1 Double-Slit Radiation

To demonstrate the difference between the arithmetic and geometric mean, we begin by considering the radiation from a double-slit aperture in a perfect electrical-conductor (PEC) screen which is illuminated by a normally incident pulsed plane wave. TM polarization is assumed. As shown in Fig. 14.5(a), in this case the boundary over which the fields are measured is three-sided and exists on only one side of the screen.

Given the fields over the three-sided boundary, one then assumes the fields “interior” to this boundary (i.e., the region which includes the slits) are zero while the fields exterior to the boundary are unchanged. To account for the discontinuity in the fields across the integration boundary, surface currents must be present. Since the fields are zero within the boundary, one can replace the actual interior with anything without affecting the exterior fields. One thus assumes that the slits are not present—that the PEC plane is unbroken. The surface currents over the three-sided boundary are now radiating in the presence of an infinite plane. The far-field radiation can be calculated with a three-sided integral where one uses the Green’s function for a source above an infinite plane. This, equivalently, from image theory, is simply the radiation from the original (measured) current and the image of the current. Both the measured current and the image current are radiating in free space. In this way the three-sided boundary can be replaced with a closed four-sided boundary as shown in Fig. 14.5(b). The corresponding currents over this surface, i.e., the measured currents over half the boundary and the image currents over the other half, are transformed to the far field.

The incident field is introduced over a total-field/scattered-field (TFSF) boundary which only exists to the left side of the screen. The grid is terminated with an eight-cell perfectly matched layer (PML).

The right side of the integration surface is $D$ cells away from the PEC screen. The length of the right side of the integration boundary is held fixed at 75 cells. In principle, the location of the integration boundary should make no difference to the far-fields. However, when using the arithmetic mean, the far-fields are sensitive to the boundary location, i.e., sensitive to the value of $D$. Note that were a single component of the field integrated over the aperture, as advocated by [1], averaging is not an issue. However, that approach is restricted to screens which are planar and there are no inhomogeneities present other than the screen. The approach we advocate can accommodate any screen or scatterer geometry provided it can be contained within the integration boundary. Nevertheless, we will employ the aperture-based approach as a reference solution.
Figure 14.5: (a) Geometry of the double-slit experiment. A pulsed plane wave is introduced via a TFSF boundary on the left side of the screen. The field are recorded over the three-sided integration boundary to the left of the screen. (b) Image theory is used to create a four-sided closed surface over which the currents are transformed to the far-field. The dashed line corresponds to the location where the PEC plane had been.
The simulation uses “slits” which are 15 cells wide. The PEC between the slits is 30 cells wide. The excitation is a Ricker wavelet discretized such that there are 30 cells per wavelength at the most energetic frequency. The simulation is run at the 2D Courant limit \((1/\sqrt{2})\) for 1024 time steps.

Figure 14.6(a) shows the far-field radiation pattern which is obtained using the arithmetic mean. The pattern is symmetric about zero degrees which corresponds to the direction normal to the screen. The radiation pattern is calculated using

\[
\frac{1}{\lambda} \lim_{\rho \to \infty} \left[ 2\pi \rho \frac{\left| \hat{E}_z(\phi) \right|^2}{\left| \hat{E}_i^z \right|^2} \right]
\]  

(14.87)

where \(\phi\) is the scattering angle, \(\rho\) is the distance from the slits, \(\hat{E}_z(\phi)\) is the field radiated in the \(\phi\) direction, and \(\hat{E}_i^z\) is the complex amplitude of the incident plane wave at the frequency of interest. Results are shown for a frequency corresponding to 10.0566 cells per wavelength. Figure 14.6(a) shows the pattern when \(D\) is either 5, 6, or 7 cells.

Note that there are significant differences in the central peak depending on the displacement \(D\) between the right-side integration boundary and the PEC screen. Figure 14.6(b) shows an expanded view of the pattern in the neighborhood of the peak. As can be seen, displacing the integration boundary by two cells causes a change in the peak of approximately 9 percent. The results as a function of displacement are nearly periodic, e.g., displacements of 5 and 15 cells (not shown) yield nearly the same results as do displacements of 6 and 16 cells, and so on. (The period of 10 cells is a consequence of examining a frequency corresponding to approximately 10 cells per wavelength.) Also shown as plus signs in Fig. 14.6(b) are the results obtained when the transform uses the electric field (i.e., the equivalent magnetic current) over the aperture. No averaging is involved in this case. The aperture-based results are seen to agree well with the arithmetic-mean results when the displacement \(D\) is 5 cells. Unfortunately one does not know \textit{a priori} that this agreement will exist nor does this displacement provide similar agreement for other frequencies.

On the other hand, when using the geometric mean, there is almost no variation in the radiation pattern as the integration boundary is displaced. Figure 14.6(c) shows the same results as presented in Fig. 14.6(b) except now the geometric mean of the harmonic fields is used to obtain the magnetic fields on the integration boundary. The variation between these peaks is less than 0.022, i.e., a reduction in variation by a factor of approximately 270. This demonstrates that, unlike with the arithmetic mean, the location of the integration boundary is effectively irrelevant when using the GM-NTFF transform.

Naturally, at finer discretizations, the difference between the geometric mean and the arithmetic mean are less dramatic, but the geometric mean consistently performs better than the arithmetic mean.

(Simpson’s rule was used for all integrations except for the integration of the aperture fields where a Riemann sum was used.)

### 14.7.2 Scattering from a Circular Cylinder

Consider scattering from a PEC circular cylinder under TM\(^z\) polarization as shown in Fig. 14.7. The Dey-Mittra scheme is employed to help reduce the effects of staircasing [2]. The cylinder has eight cells along its radius. A pulsed plane wave which travels in the \(x\) direction is introduced via
Figure 14.6: (a) Radiation from the double slit for angles between $-5$ and 90 degrees (the pattern is symmetric about zero degrees). Results are shown for boundary displacements $D$ of 5, 6, and 7 cells. The arithmetic mean is used. (b) Expanded view of the central peak using the arithmetic mean. Also shown are the fields obtained when a single field component is recorded over the aperture and transformed to the far field. (c) Same as (b) except the geometric mean is used. The variation of the fields caused by the displacement of the boundary is now essentially negligible.
a TFSF boundary. The simulation is run 512 time steps. Because the Dey-Mittra scheme is used, the Courant number was reduced to approximately 35 percent of the 2D limit in order to ensure stability [3].

Figure 14.8 shows the wavelength-normalized scattering width of the cylinder as a function of scattering angle obtained using the series solution for a circular cylinder [4], the arithmetic-mean NTFF transform, and the GM-NTFF transform. (The normalized scattering width is the same as the radiation pattern given in (14.87) where \( \rho \) is now taken to be the distance from the center of the cylinder.) The discretization is such that there are 9.92526 cells per wavelength at the frequency being considered here. One should keep in mind that the discretization of the cylinder introduces some errors and hence the “exact” solution for a circular cylinder is not truly exact for the scatterer present in the simulation. Therefore, the reference solution does not provide a perfect way with which to judge the solutions. Nevertheless, one hopes that the FDTD scatterer, when employing the Dey-Mittra scheme, is a close approximation to a true circular scatterer and thus the exact solution from the continuous world provides a reasonable basis for comparison.

The difference between the solutions in Fig. 14.8 are seen to be relatively small. Figure 14.9 shows a plot of the magnitude of the difference between the exact solution and the FDTD-based solutions. Although there are angles where the arithmetic mean performs better than the geometric mean, in general the geometric mean is better. The integrated error for angles between 0 and \( \pi \) is 0.6936 for the arithmetic mean and 0.3221, i.e., the error is reduced by more than a factor of two by using the geometric mean.

### 14.7.3 Scattering from a Strongly Forward-Scattering Sphere

Finally, consider scattering from a dielectric sphere, depicted in Fig. 14.10, which has a relative permittivity \( \epsilon_r \) of 1.21. Such a sphere was considered in [5] and can also be found in Sec. 8.7 of
Figure 14.8: Scattering width of a circular cylinder. The radius is eight cells and the frequency corresponds to 9.92526 cells per wavelength.

Figure 14.9: Magnitude of the difference between the FDTD-based solutions and the nominally exact solution.
Figure 14.10: Geometry of the dielectric sphere. The relative permittivity $\varepsilon_r$ is 1.21. This incident field is polarized in the $z$ direction and travels in the $x$ direction. The equatorial angle $\phi$ is in the $xy$-plane with $\phi = 0$ corresponding to the $+x$ direction.

In this case the transformation traditionally entails finding the tangential fields over the six sides of a cuboid which bounds the sphere. The sphere is discretized such that there are 60 cells along the radius. A staircase representation is used (where a node is simply either inside or outside the sphere). The simulation is run at 95 percent of the 3D Courant limit ($0.95/\sqrt{3}$) for 2048 time steps. The grid is terminated with an eight-cell perfectly-matched layer.

Figure 14.11 shows the normalized scattering cross section as a function of the equatorial angle $\phi$. The frequency corresponds to $20.06$ cells per wavelength. The exact solution was obtained via the Mie series (see, e.g., [7]). As was the case in 2D, both the arithmetic and geometric mean perform reasonably well, but the geometric mean is generally more accurate than the arithmetic mean. In Fig. 14.11 visible errors are only present near the back-scattering direction of $\phi = 180$ degrees. Note that there is a large difference, approximately five orders of magnitude, between the scattering in the forward and backward directions.

In order to improve the results in the back-scattering direction Li et al. [5] advocated calculating the transformation using the five faces other than the forward-scattering face. Figure 14.12 shows the normalized backscattering cross section versus wavelength (expressed in terms of number of cells per wavelength) calculated using the arithmetic mean. The normalized cross section is given by

$$\frac{1}{\lambda^2} \lim_{r \to \infty} \left[ \frac{4\pi r^2 |\tilde{E}_z(\theta, \phi)|^2}{|\tilde{E}_z|^2} \right]$$

(14.88)

where $r$ is the distance from the center of the sphere, $\theta$ is the azimuthal angle and $\phi$ is the equatorial angle. For backscatter, $\theta = \pi/2$ and $\phi = \pi$.

The NTFF transform results shown in Fig. 14.12 were calculated using either the fields over all six faces of the integration boundary or the fields over the five faces advocated by Li et al. The results in this figure correspond to those shown in Fig. 1(b) of [5] for the sphere with a radius of $3 \mu m$. (However, for the sake of generality, here the results are plotted in terms of unitless quantities.) Note that the six-sided arithmetic-mean results presented here are better than those
Figure 14.11: Scattering cross section of a dielectric sphere versus the equatorial angle $\phi$. The sphere is discretized with 60 cells along the radius and the frequency used here corresponds to 20.06 cells per wavelength.
CHAPTER 14. NEAR-TO-FAR-FIELD TRANSFORMATION

Figure 14.12: Backscatter from a sphere with $\varepsilon_r = 1.21$ versus the wavelength (expressed in terms of number of cells). The FDTD transformations are calculated using the arithmetic mean and either a five- or six-sided transformation boundary.

Figure 14.13 is the same as Fig. 14.12 except the transformation is done using the GM-NTFF transform. In this case discarding data from the forward-scattering face actually slightly degrades the quality of the transform. Thus, when using the geometric mean there is no need to discard data. One can use it confidently for all scattering angles and all sizes.

To summarize, unlike the traditional arithmetic mean, for a single harmonic plane wave the geometric mean accounts for the spatial offset of the fields in a way that is exact. In practice, where a spectrum of wave vectors are present, the geometric mean typically performs significantly better than the arithmetic mean. The geometric mean is much less sensitive to the integration-boundary location than is the arithmetic mean. For strongly forward-scattering objects, the use of the geometric mean obviates the need to discard the fields over the forward face (as has been advocated previously) when calculating the backscatter. The geometric mean does entail a slight increase in computational cost because for each node along the integration boundary a DFT must be calculated for three fields instead of two. However, this cost is typically minor compared to the

presented in [5]. We were able to duplicate the results presented in [5] by not applying a temporal phase-correction factor (or, similarly, by applying a correction factor which is twice the factor given here). Nevertheless, the recommendation of Li et al. is true that the five-sided computation is better than the six-sided one for calculating the backscattering when using the arithmetic mean. However, for directions other than backscatter or for other sizes, one does not know \textit{a priori} if a face should or should not be discarded.
14.7. NTFF TRANSFORMATIONS USING THE GEOMETRIC MEAN

Figure 14.13: Backscatter from a sphere with $\epsilon_r = 1.21$ versus the wavelength (expressed in terms of number of cells). The transformations use the geometric mean and the fields over either five or six faces of the integration boundary.
overall simulation cost.
Appendix A

Construction of Fourth-Order Central Differences

Assuming a uniform spacing of $\delta$ between sample points, we seek an approximation of the derivative of a function at $x_0$ which falls midway between two sample point. Taking the Taylor series expansion of the function at the four sample points nearest to $x_0$ yields

\[ f \left( x_0 + \frac{3\delta}{2} \right) = f(x_0) + \frac{3\delta}{2} f'(x_0) + \frac{1}{2!} \left( \frac{3\delta}{2} \right)^2 f''(x_0) + \frac{1}{3!} \left( \frac{3\delta}{2} \right)^3 f'''(x_0) + \ldots, \quad (A.1) \]

\[ f \left( x_0 + \frac{\delta}{2} \right) = f(x_0) + \frac{\delta}{2} f'(x_0) + \frac{1}{2!} \left( \frac{\delta}{2} \right)^2 f''(x_0) + \frac{1}{3!} \left( \frac{\delta}{2} \right)^3 f'''(x_0) + \ldots, \quad (A.2) \]

\[ f \left( x_0 - \frac{\delta}{2} \right) = f(x_0) - \frac{\delta}{2} f'(x_0) + \frac{1}{2!} \left( \frac{\delta}{2} \right)^2 f''(x_0) - \frac{1}{3!} \left( \frac{\delta}{2} \right)^3 f'''(x_0) + \ldots, \quad (A.3) \]

\[ f \left( x_0 - \frac{3\delta}{2} \right) = f(x_0) - \frac{3\delta}{2} f'(x_0) + \frac{1}{2!} \left( \frac{3\delta}{2} \right)^2 f''(x_0) - \frac{1}{3!} \left( \frac{3\delta}{2} \right)^3 f'''(x_0) + \ldots. \quad (A.4) \]

Subtracting (A.3) from (A.2) and (A.4) from (A.1) yields

\[ f \left( x_0 + \frac{\delta}{2} \right) - f \left( x_0 - \frac{\delta}{2} \right) = \delta f'(x_0) + \frac{2}{3!} \left( \frac{\delta}{2} \right)^3 f'''(x_0) + \ldots \quad (A.5) \]

\[ f \left( x_0 + \frac{3\delta}{2} \right) - f \left( x_0 - \frac{3\delta}{2} \right) = 3\delta f'(x_0) + \frac{2}{3!} \left( \frac{3\delta}{2} \right)^3 f'''(x_0) + \ldots \quad (A.6) \]

The goal now is to eliminate the term containing $f'''(x_0)$. This can be accomplished by multiplying (A.5) by 27 and then subtracting (A.6). The result is

\[ 27 f \left( x_0 + \frac{\delta}{2} \right) - 27 f \left( x_0 - \frac{\delta}{2} \right) - f \left( x_0 + \frac{3\delta}{2} \right) + f \left( x_0 - \frac{3\delta}{2} \right) = 24\delta f'(x_0) + O(\delta^5). \quad (A.7) \]

Solving for $f'(x_0)$ yields

\[ \left. \frac{df(x)}{dx} \right|_{x=x_0} = \frac{9}{8} f \left( x_0 + \frac{\delta}{2} \right) - f \left( x_0 - \frac{\delta}{2} \right) - 24 \frac{f \left( x_0 + \frac{3\delta}{2} \right) - f \left( x_0 - \frac{3\delta}{2} \right)}{\delta} + O(\delta^4). \quad (A.8) \]
The first term on the right-hand side is the contribution from the sample points nearest \( x_0 \) and the second term is the contribution from the next nearest points. The highest-order term not shown is fourth-order in terms of \( \delta \).
Appendix B

Generating a Waterfall Plot and Animation

Assume we are interested in plotting multiple snapshots of one-dimensional data. A waterfall plot displays all the snapshots in a single image where each snapshot is offset slightly from the next. On the other hand, animations display one image at a time, but cycle through the images quickly enough so that one can clearly visualize the temporal behavior of the field. Animations are a wonderful way to ascertain what is happening in an FDTD simulation but, since there is no way to put an animation on a piece of paper, waterfall plots also have a great deal of utility.

We begin by discussing waterfall plots. First, the data from the individual frames must be loaded into Matlab. The m-file for a function to accomplish this is shown in Program B.1. This function, `readOneD()`, reads each frame and stores the data into a matrix—each row of which corresponds to one frame. `readOneD()` takes a single argument corresponding to the base name of the frames. For example, issuing the command 

```
z = readOneD('sim');
```

would create a matrix `z` where the first row corresponded to the data in file `sim.0`, the second row to the data in file `sim.1`, and so on.

Note that there is a `waterfall()` function built into Matlab. One could use that to display the data by issuing the command `waterfall(z)`. However, the built-in command is arguably overkill. Its hidden-line removal and colorization slow the rendering and do not necessarily aide in the visualization of the data.

The plot shown in Fig. 3.9 did not use Matlab’s `waterfall()` function. Instead, it was generated using a function called `simpleWaterfall()` whose m-file is shown in Program B.2. This command takes three arguments. The first is the matrix which would be created by `readOneD()`, the second is the vertical offset between successive plots, and the third is a vertical scale factor.

Given these two m-files, Fig. 3.9 was generated using the following commands:

```
z = readOneD('sim');
simpleWaterfall(z, 1, 1.9) % vertical offset = 1, scale factor = 1.9
xlabel('Space [spatial index]')
ylabel('Time [frame number]')
```

Program B.1 `readOneD.m` Matlab code to read one-dimensional data from a series of frames.
function z = readOneD(basename)
%readOneD(BASENAME) Read 1D data from a series of frames.
% [Z, dataLength, nFrames] = readOneD(BASENAME) Data
% is read from a series of data files all which have
% the common base name given by the string BASENAME,
% then a dot, then a frame index (generally starting
% with zero). Each frame corresponds to one row of Z.

% read the first frame and establish length of data
nFrames = 0;
filename = sprintf('%s.%d', basename, nFrames);
nFrames = nFrames + 1;
if exist(filename, 'file')
z = dlmread(filename, '
');
dataLength = length(z);
else
    return;
end

% loop through other frames and break out of loop
% when next frame does not exist
while 1
    filename = sprintf('%s.%d', basename, nFrames);
nFrames = nFrames + 1;
if exist(filename, 'file')
zTmp = dlmread(filename, '
');
    if length(zTmp) ~= dataLength % check length matches
        error('Frames have different sizes.')
        break;
    end
    z = [z zTmp]; % append new data to z
else
    break;
end
end

% reshape z to appropriate dimensions
z = reshape(z, dataLength, nFrames - 1);
z = z';
return;

Program B.2 simpleWaterfall.m Matlab function to generate a simple waterfall plot.
function simpleWaterfall(z, offset, scale)
% simpleWaterfall Waterfall plot from offset x-y plots.
% simpleWaterfall(Z, OFFSET, SCALE) Plots each row of z
% where successive plots are offset from each other by
% OFFSET and each plot is scaled vertically by SCALE.

hold off
plot(scale * z(1, :)) % plot the first row
hold on
for i = 2:size(z, 1)
    plot(scale * z(i, :) + offset * (i - 1))
end
hold off

return

A function to generate an animation of one-dimensional data sets is shown in Program B.3. There are multiple ways to accomplish this goal and thus one should keep in mind that Program B.3 is not necessarily the best approach for a particular situation. The function in Program B.3 is called oneDmovie() and it takes three arguments: the base name of the snapshots, and the minimum and maximum values of the vertical axis. The function uses a loop, starting in line 25, to read each of the snapshots. Each snapshot is plotted and the plot recorded as a frame of a Matlab “movie” (see the Matlab command movie() for further details). The oneDmovie() function returns an array of movie frames. As an example, assume the base name is “sim” and the user wants the plots to range from $-1.0$ to $1.0$. The following commands would display the animation 10 times (the second argument to the movie() command controls how often the movie is repeated):

```
reel = oneDmovie('sim',-1,1);
movie(reel,10)
```

An alternative implementation might read all the data first, as was done with the waterfall plot, and then determine the “global” minimum and maximum values from the data itself. This would free the user from specify those value as oneDmovie() currently requires. Such an implementation is left to the interested reader.

---

**Program B.3** oneDmovie.m Matlab function which can be used to generate an animation for multiple one-dimensional data sets. For further information on Matlab movies, see the Matlab command movie.
% basename = common base name of all files
% y_min = minimum value used for all frames
% y_max = maximum value used for all frames
%
% reel = movie which can be played with a command such as:
%     movie(reel, 10)
%     This would play the movie 10 times. To control the frame
%     rate, add a third argument specifying the desired rate.
%
% open the first frame (i.e., first data file).
frame = 1;
filename = sprintf('%s.%d', basename, frame);
fid = fopen(filename, 'rt');

% to work around rendering bug under Mac OS X see:
% <www.mathworks.com/support/solutions/data/1-VW0GM.html?solution=1-VW0GM>
figure; set(gcf, 'Renderer', 'zbuffer');

% provided fid is not -1, there is another file to process
while fid ~= -1
    data = fscanf(fid, '%f'); % read the data
    plot(data) % plot the data
    axis([0 length(data) y_min y_max]) % scale axes appropriately
    reel(frame) = getframe; % capture the frame for the movie

    % construct the next file name and try to open it
    frame = frame + 1;
    filename = sprintf('%s.%d', basename, frame);
    fid = fopen(filename, 'rb');
end

return
Appendix C

Rendering and Animating Two-Dimensional Data

The function shown below is Matlab code that can be used to generate a movie from a sequence of binary (raw) files. The files (or frames) are assumed to be named such that they share a common base name then have a dot followed by a frame number. Here the frame number is assumed to start at zero. The function can have one, two, or three arguments. This first argument is the base name, the second is the value which is used to normalize all the data, and the third argument specifies the number of decades of data to display. Here the absolute value of the data is plotted in a color-mapped image. Logarithmic (base 10) scaling is used so that the value which is normalized to unity will correspond to zero on the color scale and the smallest normalized value will correspond, on the color scale, to the negative of the number of decades (e.g., if the number of decades were three, the smallest value would correspond to $-3$). This smallest normalized value actually corresponds to a normalized value of $10^{-d}$ where $d$ is the number of decades. Thus the (normalized) values shown in the output varying from $10^{-d}$ to $1$. The default normalization and number of decades are 1 and 3, respectively.

Program C.1 raw2movie.m Matlab function to generate a movie given a sequence of raw files.

```matlab
function reel = raw2movie(basename, z_norm, decades)
% raw2movie Creates a movie from "raw" files with a common base
% name.
%
% The absolute value of the data is used together with
% logarithmic scaling. The user may specify one, two, or
% three arguments.
% raw2movie(basename, z_norm, decades) or
% raw2movie(basename, z_norm) or
% raw2movie(basename):
% basename = common base name for all files
% z_norm = value used to normalize all frames, typically this
% would be the maximum value for all the frames.
```
% Default value is 1.
% decades = decades to be used in the display. The normalized
% data is assumed to vary between 1.0 and 10^-decades
% so that after taking the log (base 10), the values
% vary between 0 and -decades. Default value is 3.
%
% return value:
% reel = movie which can be played with a command such as:
%       movie(reel, 10)
% pcolor() is used to generate the frames.
%
% raw file format:
% The raw files are assumed to consist of all floats (in
% binary format). The first two elements specify the horizontal
% and vertical dimensions. Then the data itself is given in
% English book-reading order, i.e., from the upper left corner
% of the image and then scanned left to right. The frame number
% is assumed to start at zero.
%
% set defaults if we have less than three arguments
if nargin < 3, decades = 3; end
if nargin < 2, z_norm = 1.0; end
%
% open the first frame
frame = 0;
filename = sprintf('%s.%d', basename, frame);
fid = fopen(filename, 'rb');
if fid == -1
    error(['raw2movie: initial frame not found: ', filename])
end
%
% to work around rendering bug under Mac OS X implementation.
figure; set(gcf, 'Renderer', 'zbuffer');
%
% provided fid is not -1, there is another file to process
while fid ~= -1
    size_x = fread(fid, 1, 'single');
    size_y = fread(fid, 1, 'single');
    data = flipud(transpose(...
        reshape(...
            fread(fid, size_x * size_y, 'single'), size_x, size_y)...)
    )
%
% plot the data
if decades ~= 0
pcolor(log10(abs((data + realmin) / z_norm)))
shading flat
axis equal
axis([1 size_x 1 size_y])
caxis([-decades 0])
colorbar
else
  pcolor(abs((data + realmin) / z_norm))
  shading flat
  axis equal
  axis([1 size_x 1 size_y])
  caxis([0 1])
colorbar
end

% capture the frame for the movie (Matlab wants index to start
% at 1, not zero, hence the addition of one to the frame)
reel(frame + 1) = getframe;

% construct the next file name and try to open it
frame = frame + 1;
filename = sprintf('%s.%d', basename, frame);
fid = fopen(filename, 'rb');
end
Appendix D

Notation

- $c$: speed of light \textit{in free space}
- $N_{\text{freq}}$: index of spectral component corresponding to a frequency with discretization $N_{\lambda}$
- $N_{\lambda}$: number of points per wavelength for a given frequency (the wavelength is the one pertaining to propagation in free space)
- $N_L$: number of points per skin depth ($L$ for loss)
- $N_P$: number of points per wavelength at peak frequency of a Ricker wavelet
- $N_T$: number of time steps in a simulation
- $S_c$: Courant number ($c\Delta_t/\Delta_x$ in one dimension)
- $s_t$: Temporal shift operator
- $s_x$: Spatial shift operator (in the $x$ direction)
Appendix E

PostScript Primer

E.1 Introduction

PostScript was developed by Adobe Systems Incorporated and is both a page-description language and a programming language. Unlike a JPEG or GIF file which says what each pixel in an image should be, PostScript is “vector based” and thus a PostScript file specifies graphic primitives, such as lines and arcs. There primitives are described by various PostScript commands. The quality of the image described by a PostScript file will depend on the output device. For example, a laser printer with 1200 dots per inch will draw a better curved line than would a laser printer with 300 dots per inch (although both will typically produce very good output).

The goal here is to show how one can easily generate PostScript files which convey information about an FDTD simulation. Thus we are more interested in the page-description aspects of PostScript rather than its programming capabilities. (There is a wonderful book and Web site by Bill Casselman that describe PostScript extremely well while illustrating a host of mathematical concepts. The book is entitled Mathematical Illustrations: A Manual of Geometry and PostScript which you can find at www.math.ubc.ca/~cass/graphics/manual/. It is well worth checking out.)

PostScript is a Forth-like language in that it uses what is known as postfix notation. If you have used an RPN (reverse Polish notation) calculator, you are familiar with postfix notation. You put arguments onto a “stack” and then select an operation which “pops” the arguments from the stack and operates on them. For example, to add 3 and 12 you would enter the following:

\begin{verbatim}
3
<ENTER>
12
+
\end{verbatim}

When 3 is typed on the keypad, it is placed at the top of the stack. It is pushed onto the next stack location by hitting the ENTER key. When 12 is typed, it is put at the top of the stack. Hitting the plus sign tells the calculator you want to add the top two numbers on the stack, i.e., the 3 and 12. These numbers are popped (i.e., taken) from the stack, and the result of the addition (15) is placed at the top of the stack.

Lecture notes by John Schneider. postscript-primer.tex
The PostScript language is much like this. Arguments are given before the operations. Giving arguments before operations facilitates the construction of simple interpreters. PostScript interpreters typically have the task of translating the commands in a PostScript file to some form of viewable graphics. For example, there are PostScript printers which translate (interpret) a PostScript file into a printed page. Most computers have PostScript interpreters which permit the graphics described in a PostScript file to be displayed on the screen. There are free PostScript interpreters available via the Web (you should do a search for GhostScript if you are in need of an interpreter).

E.2 The PostScript File

A file which contains PostScript commands, which we will call a PostScript file, is a plain ASCII file which must start with “%!PS”. These characters are often referred to as a “magic word.” Magic words appear at the start of many computer files and identify the contents of the file. This %!PS magic word identifies the contents of the file as PostScript to the interpreter. (The names of PostScript file often end with the suffix .ps, but the interpreter does not care what the file name is.) The last command in a PostScript file is typically showpage. This command essentially tells the interpreter that all the commands have been given and the page (or screen image or whatever) should be rendered.

What comes between %!PS and showpage are the commands which specify how the page should appear. Before exploring some of these commands it is important to know that a PostScript interpreter, by default, thinks in terms of units of “points” which are not points in the geometric sense, but rather \( \frac{1}{72} \) of an inch. Points are a traditional unit used in the printing industry (thus a “12-point font” is one for which a typical capital letter is \( \frac{12}{72} \) of an inch high). A default “page” is 8.5 by 11 inches and thus 612 by 792 points. The origin is the lower left corner of the page.

E.3 PostScript Basic Commands

The PostScript command moveto takes two arguments: the \( x \) and \( y \) coordinates to which the current point should be moved. You can think of the current point as akin to the point where the tip of a pen is moved. To define a line we can give the command lineto. lineto also takes two arguments: the \( x \) and \( y \) coordinates of the point to which the line should be drawn. In PostScript, after issuing the lineto command we have merely defined the path of the line—we have not actually drawn anything yet. You can think of this as the pen having drawn the line in invisible ink. We have to issue one more command to make the line visible, the stroke command.

A complete PostScript file (which we will identify as a “Program”) which draws a line from the point (100, 200) to the point (300, 600) is shown in Program E.1.

Program E.1 PostScript commands to draw a single tilted line.

\[
%!PS
100 200 moveto
\]
E.3. POSTSCRIPT BASIC COMMANDS

Figure E.1: Simple line rendered by the PostScript commands giving in Program E.1 and E.2. The surrounding box and corner labels have been added for the sake of clarity.

300 600 lineto
stroke
showpage

The image drawn by these commands is shown in Fig. E.1. The surrounding border and coordinate labels have been added for clarity. The only thing which would actually be rendered is the tilted line shown within the border.

Instead of using the command lineto to specify the point to which we want the line to be drawn, the command rlineto can be used where now the arguments specify the relative movement from the current point (hence the “r” for relative). The arguments of rlineto specify the relative displacement from the current point. In the commands in Program E.1, the line which was drawn went 200 points in the \( x \) direction and 400 points in the \( y \) direction from the starting point. Thus instead of writing 300 600 lineto, one could obtain the same result using 200 400 rlineto. PostScript does not care about whitespace and the percent sign is used to indicate the start of a comment (the interpreter ignores everything from the percent sign to the end of the line). Thus, another file which would also yield the output shown in Fig. E.1 is shown in Program E.2. (The magic word must appear by itself on the first line of the file.)

Program E.2 PostScript commands to draw a single tilted line. Here the rlineto command is used. The resulting image is identical to the one produced by Program E.1 and is shown in Fig. E.1.
When creating graphics it is often convenient to redefine the origin of the coordinate system to a point which is more natural to the object being drawn. PostScript allows us to translate the origin to any point on the page using the `translate` command. This command also takes two arguments corresponding to the point in the current coordinate system where the new origin should be located. For example, let us assume we want to think in terms of both positive and negative coordinates. Therefore we wish to place the origin in the middle of the page. This can be accomplished with the command `306 396 translate`. The PostScript commands shown in Program E.3 demonstrate the use of the `translate` command.

Program E.3 PostScript commands which first translate the origin to the center of the page and then draw four lines which “radiate away” from the origin. The corner labels show the corner coordinates after the translation of the origin to the center of the page.

```
%!PS
306 398 translate % translate origin to center of page
100 100 moveto 50 50 rlineto stroke
-100 100 moveto -50 50 rlineto stroke
-100 -100 moveto -50 -50 rlineto stroke
100 -100 moveto 50 -50 rlineto stroke
showpage
```

Program E.3 yields the results shown in Fig. E.2.

As you might imagine, thinking in terms of units of points (1/72 of an inch) is not always convenient. PostScript allows us to scale the dimensions by any desired value using the `scale` command. In fact, one can use a different scale factor in both the $x$ and the $y$ directions and thus `scale` takes two arguments. However, we will stick to using equal scaling in both directions.

In the previous example, all the locations were specified in terms of multiples of 50. Therefore it might make sense to scale the dimensions by a factor of 50 (in both the $x$ and $y$ direction). This scaling should be done after the translation of the origin. We might anticipate that the commands shown in Program E.4 would render the same output as shown in Fig. E.2.

Program E.4 PostScript file where the units are scaled by a factor of 50 in both the $x$ and $y$ dimensions.
Figure E.2: Output rendered by Program E.3 which translates the origin to the center of the page. This output is also produced by Program E.5 and Program E.6.

%!PS
306 398 translate
50 50 scale % scale units in x and y direction by 50
 2 2 moveto 1 1 rlineto stroke
-2 2 moveto -1 1 rlineto stroke
-2 -2 moveto -1 -1 rlineto stroke
 2 -2 moveto 1 -1 rlineto stroke
showpage

However this file yields the output shown in Fig. E.3. Note that the lines which radiate from origin are now much thicker. In fact, they are 50 times thicker than they were in the previous image. By default, a line in PostScript has a thickness of unity i.e., one point. The scale command scaled the line thickness along with all the other dimensions so that now the line thickness is 50 points.

Although we have only given integer dimensions so far, as far as PostScript is concerned all values are actually real numbers (i.e., floating-point numbers). We can control the line thickness with the setlinewidth command which takes a single argument. If we want the line thickness still to be one point, the line thickness should be set to the inverse of the scale factor, i.e., $1/50 = 0.02$. Also, it is worth noting that the stroke command does not have to be given after each drawing command. We just have to ensure that it is given before the end of the file (or before the line style changes to something else). Thus, a PostScript file which scales the dimensions by 50 and produces the same output as shown in Fig. E.2 is shown in Program E.4.
Program E.5 PostScript file where the units are scaled by a factor of 50 and the line thickness is corrected to account for this scaling. Note that a single stroke command is given.

```
%!PS
306 398 translate
50 50 scale
0.02 setlinewidth % correct line thickness to account for scaling
2 2 moveto 1 1 rlineto
-2 2 moveto -1 1 rlineto
-2 -2 moveto -1 -1 rlineto
2 -2 moveto 1 -1 rlineto stroke
showpage
```

PostScript permits the use of named variables and, as we shall see, named procedures. This is accomplished using the def command which takes, essentially, two arguments: the first being the literal string which is the variable or procedure name and the second being the value or procedure (where the procedure would be enclosed in braces). A literal string is a backslash character followed by the string. For example, the following sets the variable scalefactor to 50:

```
/scalefactor 50 def
```

After issuing this command, we can use scalefactor in place of 50 everywhere in the file.

The PostScript language includes a number of mathematical functions. One can add using add, subtract using sub, multiply using mul, and divide using div. Each of these functions
takes two arguments consistent with an RPN calculator. To calculate the inverse of 50, one could issue the following command:

\[ \text{1 50 div} \]

This places 1 on the stack, then 50, and then divides the two. The result, 0.02, remains at the top of the stack.

The program shown in Program E.6 uses the def and div commands and is arguably a bit cleaner and better self-documenting than the one shown in Program E.5. Program E.6 also produces the output shown in Fig. E.2.

---

**Program E.6** PostScript file which uses the def command to define a scale-factor which is set to 50. The inverse of the scale-factor is obtained by using the div command to divide 1 by the scale-factor.

```
%!PS
306 398 translate
% define "scalefactor" to be 50
/scalefactor 50 def
% scale x and y directions by the scale factor
scalefactor scalefactor scale
% set line width to inverse of the scale factor
1 scalefactor div setlinewidth
2 2 moveto 1 1 rlineto
-2 2 moveto -1 1 rlineto
-2 -2 moveto -1 -1 rlineto
2 -2 moveto 1 -1 rlineto stroke
showpage
```

---

The arc command takes five arguments: the \( x \) and \( y \) location of the center of the arc, the radius of the arc, and the angles (in degrees) at which the arc starts and stops. For example, the following command would draw a complete circle of radius 0.5 about the point \((2, 2)\):

\[ 2 2 0.5 0 360 \text{ arc stroke} \]

Let us assume we wish to draw several circles, each of radius 0.5. We only wish to change the center of the circle. Rather than specifying the arc command each time with all its five arguments, we can use the def command to make the program more compact. Consider the program shown in Program E.7. Here the def command is used to say that the literal circle is equivalent to 0.5 0 360 arc stroke, i.e., three of the arguments are given to the arc command—one just has to provide the two missing arguments which are the \( x \) and \( y \) location of the center of the circle. The output produced by this program is shown in Fig. E.4.
In addition to stroke-ing a path, PostScript allows paths to be fill-ed using the fill command. So, instead of drawing a line around the perimeter of the circles shown in Fig. E.4, one can obtain filled circles by issuing the fill command instead of the stroke command. Program E.8 and the corresponding output shown in Fig. E.5 illustrate this.

Program E.8 PostScript file which defines a stroke-ed and fill-ed circle. The corresponding output is shown in Fig. E.5.
The PostScript commands we have considered are shown in Table E.1.

Instead of using an editor to write a PostScript file directly, we can use another program to generate the PostScript file for us. Specifically, let us consider a C program which generates a PostScript file. This program is supposed to demonstrate how one could use PostScript to display a particular aspect of an FDTD grid. For example, let us assume we are using a TM\textsuperscript{z} grid which is 21 cells by 21 cells. There is a PEC cylinder with a radius of 5 which is centered in the grid. We know that the $E_z$ nodes which fall within the cylinder should be set to zero and this zeroing operation would be done with a for-loop. However, precisely which nodes are being set to zero? The code shown in Program E.9 could be incorporated into an FDTD program. This code produces the PostScript output file \texttt{grid.ps} which renders the image shown in Fig. E.6. The first several lines of the file \texttt{grid.ps} are shown in Fragment E.10.
Command | Description
--- | ---
$x$ $y$ moveto | move current point to $(x, y)$
$x$ $y$ lineto | draw a line from current point to $(x, y)$
$\delta_x$ $\delta_y$ rlineto | from current point draw a line over $\delta_x$ and up $\delta_y$
$x$ $y$ translate | translate the origin to the point $(x, y)$
$s_x$ $s_y$ scale | scale $x$ and $y$ coordinates by $s_x$ and $s_y$
stroke | “apply ink” to a previously defined path
fill | fill the interior of a previously defined path
$w$ setlinewidth | set the line width to $w$
$d_1$ $d_2$ div | calculate $d_1/d_2$; result is placed at top of stack
$x_c$ $y_c$ $r$ $a_1$ $a_2$ arc | draw an arc of radius $r$ centered at $(x_c, y_c)$ starting at angle $a_1$ and ending at angle $a_2$ (degrees)
"literal {definition} def" | define the literal string to have the given definition; braces are needed if the definition contains any whitespace

Table E.1: An assortment of PostScript commands and their arguments.

Program E.9  C program which generates a PostScript file. The file draws either a cross or a filled circle depending on whether a node is outside or inside a circular boundary, respectively. The rendered image is shown in Fig. E.6.

```c
/* C program to generate a PostScript file which draws a cross
   * if a point is outside of a circular boundary and draws a
   * filled circle if the point is inside the boundary.
   */

#include <stdio.h>
#include <math.h>

int is_inside_pec(double x, double y);

int main() {
    int m, n;

    FILE *out;

    out = fopen("grid.ps","w"); // output file is "grid.ps"

    /* header material for PostScript file */
    fprintf(out,"%%!PS\n" "306 396 translate\n" "\n" "scalefactor 20 def\n" "scalefactor scalefactor scale\n" "1 scalefactor div setlinewidth\n""
```
E.3. POSTSCRIPT BASIC COMMANDS

"/cross {moveto\n  -.2 0 rmoveto .4 0 rlineto\n  -.2 -.2 rmoveto 0 .4 rlineto stroke} def\n""/circle {.2 0 360 arc fill} def\n"

for (m=-10; m<=10; m++)
  for (n=-10; n<=10; n++)
    if (is_inside_pec(m,n)) {
      fprintf(out,"%d %d circle\n",m,n);
    } else {
      fprintf(out,"%d %d cross\n",m,n);
    }

fprintf(out,"showpage\n");

return 0;

/* Function returns 1 if point (x,y) is inside a circle (or on
 * the perimeter of circle) and returns 0 otherwise. */
int is_inside_pec(double x, double y) {
  double radius = 5.0;
  return x*x + y*y <= radius*radius;
}

---

Fragment E.10  First several lines of the file grid.ps which is produced by Program E.9.

%!PS
306 396 translate
/scalefactor 20 def
scalefactor scalefactor scale
1 scalefactor div setlinewidth
/cross {moveto
  -.2 0 rmoveto .4 0 rlineto
  -.2 -.2 rmoveto 0 .4 rlineto stroke} def
/circle (.2 0 360 arc fill) def
-10 -10 cross
-10 -9 cross
-10 -8 cross
-10 -7 cross
Figure E.6: Grid depiction rendered by the file `grid.ps` which is produced by Program E.9. Crosses correspond to nodes which are outside a circular boundary of radius 5. Filled circles correspond to nodes inside the boundary (or identically on the perimeter of the boundary).
-10 -6 cross
.
.
.