CHAPTER 12

Total field/scattered field formulation

12.1 Introduction

Another technique that introduces far-zone sources (plane wave) into the FDTD computational domain is the Total Field/Scattered Field (TF/SF) method presented by Umashankar and Taflove [47]. In the scattered field (SF) method, discussed before, the scattered fields are defined in the entire computational domain and the incident fields are imposed on the entire computational domain. Unlike the SF method, in the TF/SF method the computational domain is divided into two regions: a total fields region, in which all objects are placed, and a scattered fields region that surrounds the total fields region as shown in Figure 12.1. Then, the incident plane wave is imposed on the boundary between these two regions to compensate for the discontinuity that arises due to the difference in the field values on the two sides of the boundary. The concept will be illustrated for the one-dimensional case first, and then the discussion will be extended to the three-dimensional case in the following sections.

Figure 12.2 illustrates total and scattered field regions and the field components for one-dimensional computational space. The TF/SF boundaries are indicated with the lower and upper nodes, "*li*" and "*ui*," respectively. The field components inside the total field region are considered as the total field components, whereas the field components inside the scattered field region are considered as the scattered field components and they are indicated with a "scat" subscript. The scattered field region does not contain any objects, thus it is free space as material type and it is surrounded by absorbing boundaries such as CPML. Thus the scattered fields satisfy Maxwell's equations for free space, which is translated into a one-dimensional updating equations, for instance, as

$$E^{n+1}_{scat,z}(i) = E^{n}_{scat,z}(i) + C_{ezhy}(i)\left(H^{n+\frac{1}{2}}_{scat,y}(i) - H^{n+\frac{1}{2}}_{scat,y}(i-1)\right), \tag{12.1a}$$

$$H^{n+0.5}_{scat,y}(i) = H^{n-0.5}_{scat,y}(i) + C_{hyez}(i)\left(E^{n}_{scat,z}(i+1) - E^{n}_{scat,z}(i)\right), \tag{12.1b}$$

where

$$C_{hyez}(i) = \frac{\Delta t}{\mu_0 \Delta x}, \quad \text{and} \quad C_{ezhy}(i) = \frac{\Delta t}{\varepsilon_0 \Delta x}. \tag{12.2}$$

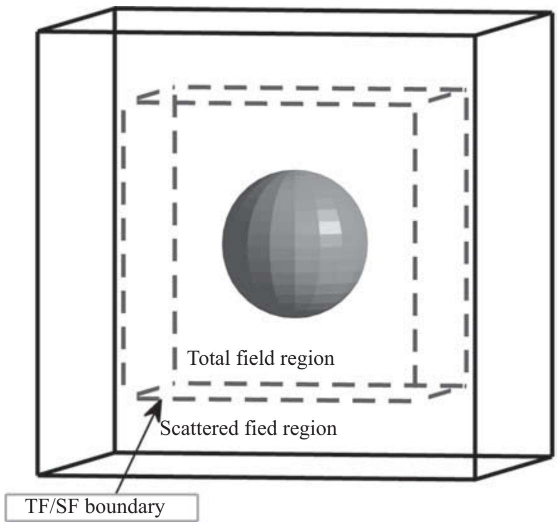

Figure 12.1 Total and scattered field regions for a three-dimensional FDTD computational domain.

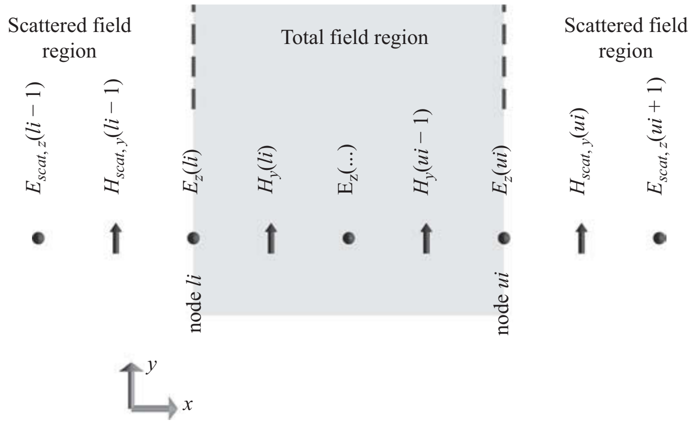

Figure 12.2 Total and scattered field components and regions for a one-dimensional FDTD computational domain.

One should notice that these updating equations are the same as the ones in (1.42) and (1.43), used for total fields, except that they are defined for free space. Since the total and scattered field components are distinct and complementary in space, and they follow the same form of the updating equations, there is no need to use separate arrays to store the total and scattered field components, as well as their respective coefficients. The updating coefficients in (1.42) and (1.43) can be simply set as the ones in (12.2) for respective scattered field components, and the field updates can be performed as usual based on (1.42) and (1.43), while implicitly considering that the field components that correspond to the scattered field region are the scattered field components and the other field components are the total field components.

After the field updates in the entire computational domain, one needs to compensate for the discontinuity between the scattered and total fields on the TF/SF boundary by using the incident field. For instance, when the field component $E_z(li)$ in Figure 12.2 is updated, the first step would be

$$E_z^{n+1}(li) = E_z^n(li) + \frac{\Delta t}{\varepsilon_0 \Delta x}\left(H_y^{n+\frac{1}{2}}(li) - H_{scat,y}^{n+\frac{1}{2}}(li-1)\right). \tag{12.3}$$

Here, the magnetic field component with index $(li-1)$ is a scattered field component, since it is in the scattered field region, where a total field component, $H_y(li-1)$, is needed to be used. Since the required $H_y(li-1)$ is the sum of $H_{scat,y}(li-1)$ and an incident field component, $H_{inc,y}(li-1)$, which can be calculated analytically, the second step required to complete the $E_z^{n+1}(li)$ update would be

$$E_z^{n+1}(li) = E_z^{n+1}(li) - \frac{\Delta t}{\varepsilon_0 \Delta x} H_{inc,y}^{n+\frac{1}{2}}(li-1). \tag{12.4}$$

A similar compensation is needed for the magnetic component, $H_{scat,y}(li-1)$, in the scattered field region. Regular field update performs

$$H_{scat,y}^{n+0.5}(li-1) = H_{scat,y}^{n-0.5}(li-1) + \frac{\Delta t}{\mu_0 \Delta x}\left(E_z^n(li) - E_{scat,z}^n(li-1)\right), \tag{12.5}$$

where $E_z^n(li)$ is used instead of the needed $E_{scat,z}^n(li)$ component. Similar to (12.4), an additional update as

$$H_{scat,y}^{n+0.5}(li-1) = H_{scat,y}^{n+0.5}(li-1) - \frac{\Delta t}{\mu_0 \Delta x} E_{inc,z}^n(li), \tag{12.6}$$

compensates for the needed field component.

The same kind of additional updates are required on the other (right) TF/SF boundary as well. In that case, the additional updates can be listed as

$$E_z^{n+1}(ui) = E_z^{n+1}(ui) + \frac{\Delta t}{\varepsilon_0 \Delta x} H_{inc,y}^{n+\frac{1}{2}}(ui), \tag{12.7}$$

$$H_{scat,y}^{n+0.5}(ui) = H_{scat,y}^{n+0.5}(ui) + \frac{\Delta t}{\mu_0 \Delta x} E_{inc,z}^n(ui). \tag{12.8}$$

As illustrated above, the values of the incident fields are needed only at the two sides of the TF/SF boundaries, unlike the scattered field formulation, which requires the values of the incident fields to be computed in the entire computational domain.

It is straightforward to extend the TF/SF concept to two- and three-dimensional domains. In the three-dimensional case, the incident field corrections are needed on the six faces of a TF/SF boundary surface that surrounds all scatterers in the computational space.

Figures 12.3 and 12.4 illustrate the field components that need compensation using the incident fields after performing the regular field updates. In these figures, the boundaries of the imaginary surface are indicated by the node indices *li*, *lj*, *lk*, *ui*, *uj*, and *uk*, where "*i*", "*j*", and "*k*" denote the indices in *x*, *y*, and *z* directions, respectively, while "*l*" and "*u*"

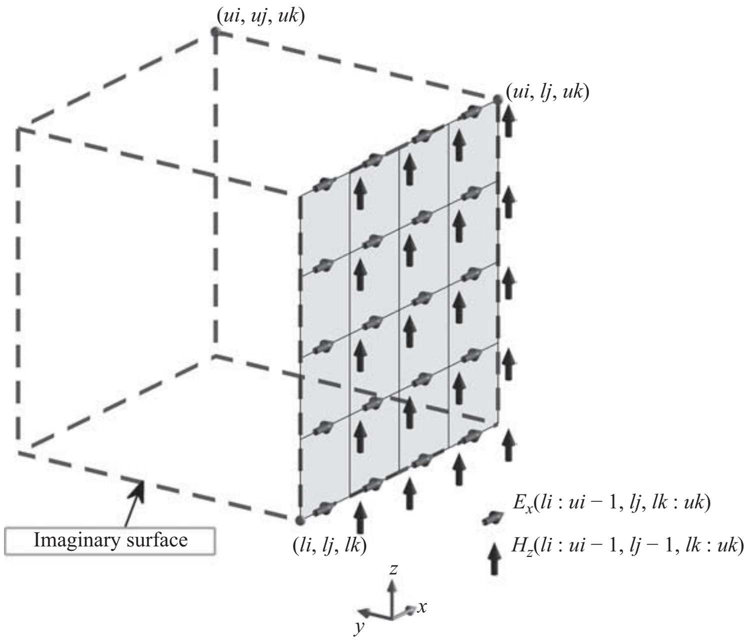
(ui, uj, uk)
(ui, lj, uk)
Imaginary surface
(li, lj, lk)
$E_x(li : ui-1, lj, lk : uk)$
$H_z(li : ui-1, lj-1, lk : uk)$
z
x
y

Figure 12.3 E_x and H_z field components that needs incident field compensation on the *yn* face of the imaginary surface.

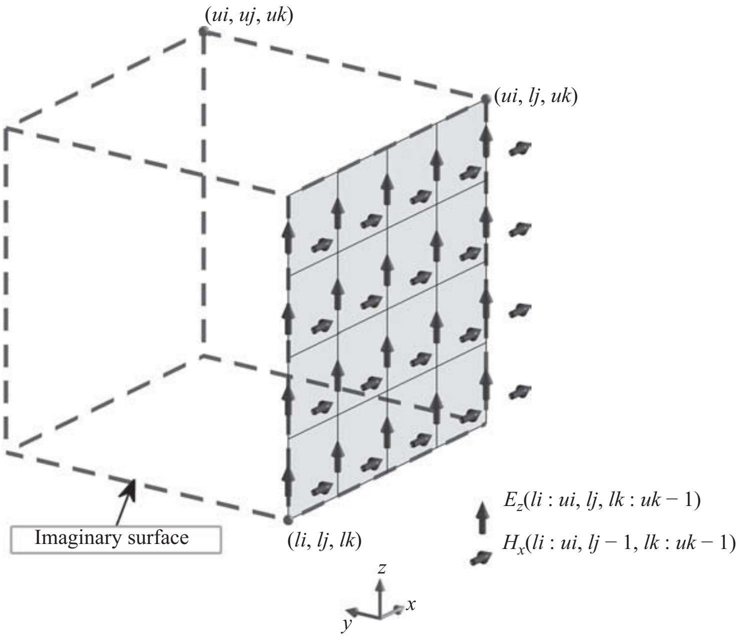
(ui, uj, uk)
(ui, lj, uk)
Imaginary surface
(li, lj, lk)
$E_z(li : ui, lj, lk : uk-1)$
$H_x(li : ui, lj-1, lk : uk-1)$
z
x
y

Figure 12.4 E_z and H_x field components that needs incident field compensation on the *yn* face of the imaginary surface.

denote lower and upper limits of the boundaries. One can visualize in Figure 12.3 that the additional field updates for the *yn* boundary would be as

$$E_x^{n+1}(li:ui-1,\, lj,\, lk:uk) = E_x^{n+1}(li:ui-1,\, lj,\, lk:uk) - \frac{\Delta t}{\varepsilon_0 \Delta x} H_{inc,z}^{n+\frac{1}{2}}(li:ui-1,\, lj-1,\, lk:uk), \tag{12.9}$$

and

$$H_{scat,z}^{n+0.5}(li:ui-1,\, lj-1,\, lk:uk) = H_{scat,z}^{n+0.5}(li:ui-1,\, lj-1,\, lk:uk) - \frac{\Delta t}{\mu_0 \Delta x} E_{inc,x}^{n}(li:ui-1,\, lj,\, lk:uk), \tag{12.10}$$

where ":" denotes the range of indices similar to its usage in FORTRAN and MATLAB®. Similarly, one can visualize in Figure 12.4 that the additional field updates would be

$$E_z^{n+1}(li:ui,\, lj,\, lk:uk-1) = E_z^{n+1}(li:ui,\, lj,\, lk:uk-1) + \frac{\Delta t}{\varepsilon_0 \Delta x} H_{inc,x}^{n+\frac{1}{2}}(li:ui,\, lj-1,\, lk:uk-1), \tag{12.11}$$

and

$$H_{scat,x}^{n+0.5}(li:ui,\, lj-1,\, lk:uk-1) = H_{scat,x}^{n+0.5}(li:ui,\, lj-1,\, lk:uk-1) + \frac{\Delta t}{\mu_0 \Delta x} E_{inc,z}^{n}(li:ui,\, lj,\, lk:uk-1). \tag{12.12}$$

Equations (12.9)–(12.12) can be modified for the *yp* boundary as

$$E_x^{n+1}(li:ui-1,\, uj,\, lk:uk) = E_x^{n+1}(li:ui-1,\, uj,\, lk:uk) + \frac{\Delta t}{\varepsilon_0 \Delta x} H_{inc,z}^{n+\frac{1}{2}}(li:ui-1,\, uj,\, lk:uk), \tag{12.13}$$

$$H_{scat,z}^{n+0.5}(li:ui-1,\, uj,\, lk:uk) = H_{scat,z}^{n+0.5}(li:ui-1,\, uj,\, lk:uk) + \frac{\Delta t}{\mu_0 \Delta x} E_{inc,x}^{n}(li:ui-1,\, uj,\, lk:uk), \tag{12.14}$$

$$E_z^{n+1}(li:ui,\, uj,\, lk:uk-1) = E_z^{n+1}(li:ui,\, uj,\, lk:uk-1) - \frac{\Delta t}{\varepsilon_0 \Delta x} H_{inc,x}^{n+\frac{1}{2}}(li:ui,\, uj,\, lk:uk-1), \tag{12.15}$$

$$H_{scat,x}^{n+0.5}(li:ui,\, uj,\, lk:uk-1) = H_{scat,x}^{n+0.5}(li:ui,\, uj,\, lk:uk-1) - \frac{\Delta t}{\mu_0 \Delta x} E_{inc,z}^{n}(li:ui,\, uj,\, lk:uk-1). \tag{12.16}$$

Equations for other boundaries can be obtained similarly.

12.2 MATLAB® implementation of the TF/SF formulation

In Section 11.4, the MATLAB implementation of a scattered field formulation was presented including the definition of an incident plane wave, initialization of the incident fields, initialization of updating coefficients and calculation of the scattered fields. Thus it will be easy to modify the code of the scattered field formulation and develop a code for the TF/SF formulation since there is a significant overlap among the elements of these methods. In particular, the way the incident field is defined and implemented is the same except that the incident fields are defined on the entire problem space in the former, while they are defined on only both sides of the TF/SF surface in the latter.

12.2.1 Definition and initialization of incident fields

A plane wave with a Gaussian waveform will be considered as the incident field in this implementation. The definition of an incident plane wave was illustrated in Listing 11.1. The same definition can be retained. However, the initialization of the incident fields will be significantly different for the implementation for the TF/SF formulation since the fields need to be known only around the six sides of the rectangular surface defining the TF/SF boundary.

The first task in the initialization of the incident plane wave is to determine the location of the TF/SF boundary. This surface should reside in the air gap between the objects and the absorbing boundary regions enclosing the problem space. Listing 12.1 shows the part of the initialization subroutine ***initialize_incident_plane_wave***, where this boundary is placed three cells away from CPML absorbing boundaries and the values of the node indices *li*, *lj*, *lk*, *ui*, *uj*, and *uk* are updated accordingly.

Listing 12.1 initialize_incident_plane_wave.m: Locate the TF/SF boundary

```
% leaving three cells between the outer boundary and the TF/SF boundary
li = 3;    lj = 3;    lk = 3;
ui = nx-3; uj = ny-3; uk = nz-3;
if strcmp(boundary.type_xn, 'cpml')
   li = boundary.cpml_number_of_cells_xn + 3;
end
if strcmp(boundary.type_yn, 'cpml')
   lj = boundary.cpml_number_of_cells_yn + 3;
end
if strcmp(boundary.type_zn, 'cpml')
   lk = boundary.cpml_number_of_cells_zn + 3;
end
if strcmp(boundary.type_xp, 'cpml')
   ui = nx-boundary.cpml_number_of_cells_xp - 3;
end
if strcmp(boundary.type_yp, 'cpml')
   uj = ny-boundary.cpml_number_of_cells_yp - 3;
end
if strcmp(boundary.type_zp, 'cpml')
   uk = nz-boundary.cpml_number_of_cells_zp - 3;
end
```

The next step is to allocate incident field arrays for the six sides of the TF/SF surface. For instance, as illustrated in Figures 12.3 and 12.4, arrays for $E_{inc,x}$, $E_{inc,z}$, $H_{inc,x}$, and $H_{inc,z}$ need to be defined for the *yn* boundary. Each of these arrays is three dimensional, while its size in the *y* dimension is equal to one, thus making it effectively a two-dimensional array. Similarly, four incident field arrays are needed for each of other sides, thus in total 24 arrays are defined as illustrated in Listing 12.2.

Then the amplitudes of the incident electric and magnetic field waveforms are transformed from spherical coordinates to Cartesian coordinates to yield **Exi0**, **Eyi0**, **Ezi0**, **Hxi0**, **Hyi0**, and **Hzi0** as shown in Listing 12.3. Moreover, the components of the propagation vector are calculated and stored in **k_vec_x**, **k_vec_y**, **k_vec_z** parameters. Also, the spatial shift required for the plane wave, which ensures that the wave is outside the computational domain when the time-marching loop starts and lets the wave enter to the computational domain on its leading front as the time-marching loop proceeds, is calculated and stored as the parameter **l_0**. It should be noted that these calculations are the same as the corresponding ones in Listing 11.2.

Listing 11.3 also includes the calculation of three-dimensional arrays **k_dot_r_ex**, **k_dot_r_ey**, **k_dot_r_ez**, **k_dot_r_hx**, **k_dot_r_hy**, and **k_dot_r_hz** as discussed in Section 11.4.2. These arrays need to be defined on the six faces of TF/SF boundary as two-dimensional arrays for the case of TF/SF formulation. For instance, Listing 12.4 shows the allocation and initialization of these arrays for the *yn* side of the TF/SF boundary.

Listing 12.2 initialize_incident_plane_wave.m: Allocate incident field arrays

```
Exi_yn = zeros(ui-li,1,uk-lk+1);
Exi_yp = zeros(ui-li,1,uk-lk+1);
Exi_zn = zeros(ui-li,uj-lj+1,1);
Exi_zp = zeros(ui-li,uj-lj+1,1);

Eyi_xn = zeros(1,uj-lj,uk-lk+1);
Eyi_xp = zeros(1,uj-lj,uk-lk+1);
Eyi_zn = zeros(ui-li+1,uj-lj,1);
Eyi_zp = zeros(ui-li+1,uj-lj,1);

Ezi_xn = zeros(1,uj-lj+1,uk-lk);
Ezi_xp = zeros(1,uj-lj+1,uk-lk);
Ezi_yn = zeros(ui-li+1,1,uk-lk);
Ezi_yp = zeros(ui-li+1,1,uk-lk);

Hzi_yn = zeros(ui-li,1,uk-lk+1);
Hzi_yp = zeros(ui-li,1,uk-lk+1);
Hyi_zn = zeros(ui-li,uj-lj+1,1);
Hyi_zp = zeros(ui-li,uj-lj+1,1);

Hzi_xn = zeros(1,uj-lj,uk-lk+1);
Hzi_xp = zeros(1,uj-lj,uk-lk+1);
Hxi_zn = zeros(ui-li+1,uj-lj,1);
Hxi_zp = zeros(ui-li+1,uj-lj,1);

Hyi_xn = zeros(1,uj-lj+1,uk-lk);
Hyi_xp = zeros(1,uj-lj+1,uk-lk);
Hxi_yn = zeros(ui-li+1,1,uk-lk);
Hxi_yp = zeros(ui-li+1,1,uk-lk);
```

Listing 12.3 initialize_incident_plane_wave.m: Calculate amplitudes and propagation vector

```
% calculate the amplitude factors for field components
theta_incident = incident_plane_wave.theta_incident*pi/180;
phi_incident = incident_plane_wave.phi_incident*pi/180;
E_theta = incident_plane_wave.E_theta;
E_phi = incident_plane_wave.E_phi;
eta_0 = sqrt(mu_0/eps_0);
Exi0 = E_theta * cos(theta_incident) * cos(phi_incident) ...
   - E_phi * sin(phi_incident);
Eyi0 = E_theta * cos(theta_incident) * sin(phi_incident) ...
   + E_phi * cos(phi_incident);
Ezi0 = -E_theta * sin(theta_incident);
Hxi0 = (-1/eta_0)*(E_phi * cos(theta_incident) ...
   * cos(phi_incident) + E_theta * sin(phi_incident));
Hyi0 = (-1/eta_0)*(E_phi * cos(theta_incident) ...
   * sin(phi_incident) - E_theta * cos(phi_incident));
Hzi0 = (1/eta_0)*(E_phi * sin(theta_incident));

% calculate spatial shift, l_0, required for incident plane wave
r0 =[fdtd_domain.min_x fdtd_domain.min_y fdtd_domain.min_z;
fdtd_domain.min_x fdtd_domain.min_y fdtd_domain.max_z;
fdtd_domain.min_x fdtd_domain.max_y fdtd_domain.min_z;
fdtd_domain.min_x fdtd_domain.max_y fdtd_domain.max_z;
fdtd_domain.max_x fdtd_domain.min_y fdtd_domain.min_z;
fdtd_domain.max_x fdtd_domain.min_y fdtd_domain.max_z;
fdtd_domain.max_x fdtd_domain.max_y fdtd_domain.min_z;
fdtd_domain.max_x fdtd_domain.max_y fdtd_domain.max_z;];

k_vec_x =  sin(theta_incident)*cos(phi_incident);
k_vec_y =  sin(theta_incident)*sin(phi_incident);
k_vec_z =  cos(theta_incident);

k_dot_r0 = k_vec_x * r0(:,1) ...
   + k_vec_y * r0(:,2) ...
   + k_vec_z * r0(:,3);

l_0 = min(k_dot_r0)/c;
```

Listing 12.4 initialize_incident_plane_wave.m: Calculate $\hat{k} \cdot \bar{r}$ arrays

```
k_dot_r_ex_yn = zeros(ui-li,1,uk-lk+1);
k_dot_r_ez_yn = zeros(ui-li+1,1,uk-lk);
k_dot_r_hx_yn = zeros(ui-li+1,1,uk-lk);
k_dot_r_hz_yn = zeros(ui-li,1,uk-lk+1);

% calculate k.r for every field component on the TF/SF boundary
for mk=lk:uk
      for mi=li:ui-1
      mii = mi - li + 1;
      mjj = 1;
      mkk = mk - lk + 1;
```

```
  mj = lj;
      x = min_x + dx * (mi-0.5);
      y = min_y + dy * (mj-1);
      z = min_z + dz * (mk-1);

      k_dot_r_ex_yn(mii,mjj,mkk) = ...
         (x*k_vec_x + y*k_vec_y + z*k_vec_z)/c;

      y = min_y + dy * (mj-1.5);
      k_dot_r_hz_yn(mii,mjj,mkk) = ...
         (x*k_vec_x + y*k_vec_y + z*k_vec_z)/c;

   end
end

for mi=li:ui
   for mk=lk:uk-1
      mjj = 1;
      mii = mi - li + 1;
      mkk = mk - lk + 1;

      mj = lj;
      x = min_x + dx * (mi-1);
      y = min_y + dy * (mj-1);
      z = min_z + dz * (mk-0.5);
      k_dot_r_ez_yn(mii,mjj,mkk) = ...
         (x*k_vec_x + y*k_vec_y + z*k_vec_z)/c;

      y = min_y + dy * (mj-1.5);
      k_dot_r_hx_yn(mii,mjj,mkk) = ...
         (x*k_vec_x + y*k_vec_y + z*k_vec_z)/c;

   end
end

% embed spatial shift in k.r
k_dot_r_ex_yn = k_dot_r_ex_yn - l_0;
k_dot_r_ez_yn = k_dot_r_ez_yn - l_0;
k_dot_r_hx_yn = k_dot_r_hx_yn - l_0;
k_dot_r_hz_yn = k_dot_r_hz_yn - l_0;
```

12.2.2 Updating incident fields

The incident field arrays and other auxiliary arrays are allocated and initialized in the subroutine ***initialize_incident_plane_wave*** as discussed in the previous section. Now these arrays are ready for use in the time-marching loop of the FDTD method, which is implemented in the subroutine ***run_fdtd_time_marching_loop***. The incident electric and magnetic field components on the TF/SF boundary need to be recalculated at every iteration of the time-marching loop before they are used to update the total or scattered fields on both sides of the boundary. Listing 12.5 shows the subroutine ***update_incident_fields***, which is used to recalculate these boundary field components. First, the total electric and magnetic field

Listing 12.5 update_incident_fields.m

```
tm = current_time + dt/2;
te = current_time + dt;

% if waveform is Gaussian waveforms
tau = incident_plane_wave.tau;
t_0 = incident_plane_wave.t_0;

Exi_yn = Exi0 * exp(-((tm - t_0 - k_dot_r_ex_yn )/tau).^2);
Exi_yp = Exi0 * exp(-((tm - t_0 - k_dot_r_ex_yp )/tau).^2);
Exi_zn = Exi0 * exp(-((tm - t_0 - k_dot_r_ex_zn )/tau).^2);
Exi_zp = Exi0 * exp(-((tm - t_0 - k_dot_r_ex_zp )/tau).^2);

Eyi_zn = Eyi0 * exp(-((tm - t_0 - k_dot_r_ey_zn )/tau).^2);
Eyi_zp = Eyi0 * exp(-((tm - t_0 - k_dot_r_ey_zp )/tau).^2);
Eyi_xn = Eyi0 * exp(-((tm - t_0 - k_dot_r_ey_xn )/tau).^2);
Eyi_xp = Eyi0 * exp(-((tm - t_0 - k_dot_r_ey_xp )/tau).^2);

Ezi_xn = Ezi0 * exp(-((tm - t_0 - k_dot_r_ez_xn )/tau).^2);
Ezi_xp = Ezi0 * exp(-((tm - t_0 - k_dot_r_ez_xp )/tau).^2);
Ezi_yn = Ezi0 * exp(-((tm - t_0 - k_dot_r_ez_yn )/tau).^2);
Ezi_yp = Ezi0 * exp(-((tm - t_0 - k_dot_r_ez_yp )/tau).^2);

Hxi_yn = Hxi0 * exp(-((te - t_0 - k_dot_r_hx_yn )/tau).^2);
Hxi_yp = Hxi0 * exp(-((te - t_0 - k_dot_r_hx_yp )/tau).^2);
Hxi_zn = Hxi0 * exp(-((te - t_0 - k_dot_r_hx_zn )/tau).^2);
Hxi_zp = Hxi0 * exp(-((te - t_0 - k_dot_r_hx_zp )/tau).^2);

Hyi_zn = Hyi0 * exp(-((te - t_0 - k_dot_r_hy_zn )/tau).^2);
Hyi_zp = Hyi0 * exp(-((te - t_0 - k_dot_r_hy_zp )/tau).^2);
Hyi_xn = Hyi0 * exp(-((te - t_0 - k_dot_r_hy_xn )/tau).^2);
Hyi_xp = Hyi0 * exp(-((te - t_0 - k_dot_r_hy_xp )/tau).^2);

Hzi_xn = Hzi0 * exp(-((te - t_0 - k_dot_r_hz_xn )/tau).^2);
Hzi_xp = Hzi0 * exp(-((te - t_0 - k_dot_r_hz_xp )/tau).^2);
Hzi_yn = Hzi0 * exp(-((te - t_0 - k_dot_r_hz_yn )/tau).^2);
Hzi_yp = Hzi0 * exp(-((te - t_0 - k_dot_r_hz_yp )/tau).^2);
```

components are defined at time instants that are referred to as "*te*" and "*tm*," respectively, in Listing 12.5. One should notice that "*te*" used to calculate incident magnetic field components. Incident magnetic field components are needed at the time instants at which total/scattered electric field components are defined since these components are added to (or subtracted from) the total/scattered electric field components as (12.9), (12.11), (12.13), and (12.15). Similarly, "*tm*" used to calculate incident electric field components since these components are added to (or subtracted from) the total/scattered magnetic field components as (12.10), (12.12), (12.14), and (12.16).

12.2.3 Updating fields on both sides of the TF/SF boundary

In the FDTD time-marching loop, the magnetic field components in the entire problem space are updated in the subroutine ***update_magnetic_fields***, as shown in Listing 12.6,

Listing 12.6 update_magnetic_fields.m

```
% update magnetic fields

current_time  = current_time + dt/2;

Hx  = Chxh.*Hx+Chxey.*(Ey(1:nxp1,1:ny,2:nzp1)-Ey(1:nxp1,1:ny,1:nz)) ...
    + Chxez.*(Ez(1:nxp1,2:nyp1,1:nz)-Ez(1:nxp1,1:ny,1:nz));

Hy  = Chyh.*Hy+Chyez.*(Ez(2:nxp1,1:nyp1,1:nz)-Ez(1:nx,1:nyp1,1:nz)) ...
    + Chyex.*(Ex(1:nx,1:nyp1,2:nzp1)-Ex(1:nx,1:nyp1,1:nz));

Hz  = Chzh.*Hz+Chzex.*(Ex(1:nx,2:nyp1,1:nzp1)-Ex(1:nx,1:ny,1:nzp1))  ...
    + Chzey.*(Ey(2:nxp1,1:ny,1:nzp1)-Ey(1:nx,1:ny,1:nzp1));

if incident_plane_wave.enable
   xui = incident_plane_wave.ui;
   xuj = incident_plane_wave.uj;
   xuk = incident_plane_wave.uk;
   xli  = incident_plane_wave.li;
   xlj  = incident_plane_wave.lj;
   xlk = incident_plane_wave.lk;
    % Hx_yn
        Hx(xli:xui,xlj-1,xlk:xuk-1) = ...
        Hx(xli:xui,xlj-1,xlk:xuk-1) + dt/(mu_0*dy) * Ezi_yn(:,1,:);
    % Hx_yp
        Hx(xli:xui,xuj,xlk:xuk-1) = ...
        Hx(xli:xui,xuj,xlk:xuk-1) - dt/(mu_0*dy) * Ezi_yp(:,1,:);
    % Hx_zn
        Hx(xli:xui,xlj:xuj-1,xlk-1) = ...
        Hx(xli:xui,xlj:xuj-1,xlk-1) - dt/(mu_0*dz) * Eyi_zn(:,:,1);
    % Hx_zp
        Hx(xli:xui,xlj:xuj-1,xuk) = ...
        Hx(xli:xui,xlj:xuj-1,xuk) + dt/(mu_0*dz) * Eyi_zp(:,:,1);

    % Hy_xn
        Hy(xli-1,xlj:xuj,xlk:xuk-1) = ...
        Hy(xli-1,xlj:xuj,xlk:xuk-1) - dt/(mu_0*dx) * Ezi_xn(1,:,:);
    % Hy_xp
        Hy(xui,xlj:xuj,xlk:xuk-1) = ...
        Hy(xui,xlj:xuj,xlk:xuk-1) + dt/(mu_0*dx) * Ezi_xp(1,:,:);
    % Hy_zn
        Hy(xli:xui-1,xlj:xuj,xlk-1) = ...
        Hy(xli:xui-1,xlj:xuj,xlk-1) + dt/(mu_0*dz) * Exi_zn(:,:,1);
    % Hy_zp
        Hy(xli:xui-1,xlj:xuj,xuk) = ...
        Hy(xli:xui-1,xlj:xuj,xuk) - dt/(mu_0*dz) * Exi_zp(:,:,1);

    % Hz_yn
        Hz(xli:xui-1, xlj-1,xlk:xuk) = ...
        Hz(xli:xui-1, xlj-1,xlk:xuk) - dt/(mu_0*dy) * Exi_yn(:,1,:);
    % Hz_yp
        Hz(xli:xui-1, xuj,xlk:xuk) = ...
        Hz(xli:xui-1, xuj,xlk:xuk) + dt/(mu_0*dy) * Exi_yp(:,1,:);
    % Hz_xn
        Hz(xli-1,xlj:xuj-1,xlk:xuk) = ...
        Hz(xli-1,xlj:xuj-1,xlk:xuk) + dt/(mu_0*dx) * Eyi_xn(1,:,:);
    % Hz_xp
        Hz(xui,xlj:xuj-1,xlk:xuk) = ...
        Hz(xui,xlj:xuj-1,xlk:xuk) - dt/(mu_0*dx) * Eyi_xp(1,:,:);
end
```

using the usual updating equations. These field components will be implicitly regarded as total fields inside the TF/SF boundaries and scattered fields outside. Once the update is completed, the incident field components can be added or subtracted from the boundary components as illustrated in Listing 12.6. In the code the lines denoted as updating **Hz_yn**, **Hx_yn**, **Hz_yp**, and **Hx_yp** correspond to (12.10), (12.12), (12.14), and (12.16), respectively.

Similarly, the electric field components in the entire problem space are updated in the subroutine ***update_electric_fields***, as shown in Listing 12.7, using the usual updating equations. Once the update is completed, the incident field components can be added or

Listing 12.7 update_electric_fields.m

```
% update electric fields except the tangential components
% on the boundaries

current_time      =current_time + dt/2;

Ex(1:nx,2:ny,2:nz) =Cexe(1:nx,2:ny,2:nz).*Ex(1:nx,2:ny,2:nz) ...
                   + Cexhz(1:nx,2:ny,2:nz).*...
                     (Hz(1:nx,2:ny,2:nz)-Hz(1:nx,1:ny-1,2:nz)) ...
                   + Cexhy(1:nx,2:ny,2:nz).*...
                     (Hy(1:nx,2:ny,2:nz)-Hy(1:nx,2:ny,1:nz-1));

Ey(2:nx,1:ny,2:nz) =Ceye(2:nx,1:ny,2:nz).*Ey(2:nx,1:ny,2:nz) ...
                   + Ceyhx(2:nx,1:ny,2:nz).*  ...
                     (Hx(2:nx,1:ny,2:nz)-Hx(2:nx,1:ny,1:nz-1)) ...
                   + Ceyhz(2:nx,1:ny,2:nz).*  ...
                     (Hz(2:nx,1:ny,2:nz)-Hz(1:nx-1,1:ny,2:nz));

Ez(2:nx,2:ny,1:nz)  =Ceze(2:nx,2:ny,1:nz).*Ez(2:nx,2:ny,1:nz) ...
                    + Cezhy(2:nx,2:ny,1:nz).*  ...
                      (Hy(2:nx,2:ny,1:nz)-Hy(1:nx-1,2:ny,1:nz)) ...
                    + Cezhx(2:nx,2:ny,1:nz).*...
                      (Hx(2:nx,2:ny,1:nz)-Hx(2:nx,1:ny-1,1:nz));

if incident_plane_wave.enable
   xui = incident_plane_wave.ui;
   xuj = incident_plane_wave.uj;
   xuk = incident_plane_wave.uk;
   xli = incident_plane_wave.li;
   xlj = incident_plane_wave.lj;
   xlk = incident_plane_wave.lk;
    % Ex_yn
        Ex(xli:xui-1,xlj,xlk:xuk) = ...
        Ex(xli:xui-1,xlj,xlk:xuk) - dt/(eps_0*dy) * Hzi_yn(:,1,:);
    % Ex_yp
        Ex(xli:xui-1,xuj,xlk:xuk) = ...
        Ex(xli:xui-1,xuj,xlk:xuk) + dt/(eps_0*dy) * Hzi_yp(:,1,:);
    % Ex_zn
        Ex(xli:xui-1,xlj:xuj,xlk) = ...
        Ex(xli:xui-1,xlj:xuj,xlk) + dt/(eps_0*dz) * Hyi_zn(:,:,1);
    % Ex_zp
        Ex(xli:xui-1,xlj:xuj,xuk) = ...
        Ex(xli:xui-1,xlj:xuj,xuk) - dt/(eps_0*dz) * Hyi_zp(:,:,1);

    % Ey_xn
        Ey(xli,xlj:xuj-1,xlk:xuk) = ...
        Ey(xli,xlj:xuj-1,xlk:xuk) + dt/(eps_0*dx) * Hzi_xn(1,:,:);
    % Ey_xp
        Ey(xui,xlj:xuj-1,xlk:xuk) = ...
        Ey(xui,xlj:xuj-1,xlk:xuk) - dt/(eps_0*dx) * Hzi_xp(1,:,:);
    % Ey_zn
        Ey(xli:xui,xlj:xuj-1,xlk) = ...
        Ey(xli:xui,xlj:xuj-1,xlk) - dt/(eps_0*dz) * Hxi_zn(:,:,1);
    % Ey_zp
        Ey(xli:xui,xlj:xuj-1,xuk) = ...
        Ey(xli:xui,xlj:xuj-1,xuk) + dt/(eps_0*dz) * Hxi_zp(:,:,1);

    % Ez_xn
        Ez(xli,xlj:xuj,xlk:xuk-1) = ...
        Ez(xli,xlj:xuj,xlk:xuk-1) - dt/(eps_0*dx) * Hyi_xn(1,:,:);
    % Ez_xp
        Ez(xui,xlj:xuj,xlk:xuk-1) = ...
        Ez(xui,xlj:xuj,xlk:xuk-1) + dt/(eps_0*dx) * Hyi_xp(1,:,:);
    % Ez_yn
        Ez(xli:xui,xlj,xlk:xuk-1) = ...
        Ez(xli:xui,xlj,xlk:xuk-1) + dt/(eps_0*dy) * Hxi_yn(:,1,:);
    % Ez_yp
        Ez(xli:xui,xuj,xlk:xuk-1) = ...
        Ez(xli:xui,xuj,xlk:xuk-1) - dt/(eps_0*dy) * Hxi_yp(:,1,:);
end
```

subtracted from the boundary components as illustrated in Listing 12.7. In the code, the lines denoted as updating **Ex_yn**, **Ez_yn**, **Ex_yp**, and **Ez_yp** correspond to (12.9), (12.11), (12.13), and (12.15), respectively.

12.3 Simulation examples

The presented TF/SF formulation can be used to solve general scattering problems in which the objects float in free space, that is, objects do not touch the boundaries or do not penetrate into the absorbing boundaries of the problem space. Although the same types of problems can be solved by the scattered field formulation, discussed in Chapter 11, as well, solution using the TF/SF formulation is much faster while it also requires less memory. Two examples are presented in the following sections.

12.3.1 Fields in an empty problem space

The distribution of total and scattered fields in an empty problem space is illustrated in this example. A brick of size 20, 20, 100 mm and material type of air is surrounded by 10 cells of air gap and 8 cells of CPML in all directions. The cell size of the grid is 1 mm on a side. An incident field that propagates in $\theta = 30^\circ$ and $\phi = 90^\circ$ direction with a Gaussian waveform is defined as the source in the subroutine ***define_sources_and_lumped_elements*** as shown in Listing 12.8. The transient fields are captured and displayed using animation on a *yz* plane-cut. Figure 12.5 shows the magnitude of the electric fields at 130, 230, and 330 time steps. One can notice that a wave of Gaussian waveform travels in the $\theta = 30^\circ$ direction as the time-marching loop proceeds. The field displayed inside the TF/SF boundary is the total field and any field that would be displayed outside the TF/SF boundary is the scattered field. In this example, the scattered field does not exist due to the absence of scatterers in the problem space therefore the displayed field is actually the incident field, which is also the total field. One can also notice the discontinuity of the field on the TF/SF boundary; the scattered field is zero outside the TF/SF boundary.

Listing 12.8 define_sources_and_lumped_elements.m

```
% Define incident plane wave, angles are in degrees
incident_plane_wave.E_theta = 1;
incident_plane_wave.E_phi = 0;
incident_plane_wave.theta_incident = 30;
incident_plane_wave.phi_incident = 90;
incident_plane_wave.waveform_type = 'gaussian';
incident_plane_wave.number_of_cells_per_wavelength = 20;
incident_plane_wave.enable = true;
```

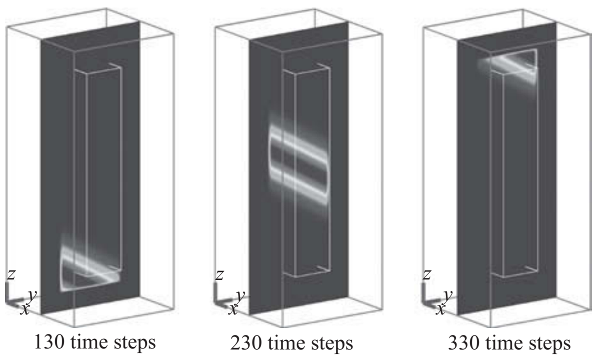

Figure 12.5 Electric field distribution on a *yz* plane-cut at three progressive time instants.

12.3.2 Scattering from a dielectric sphere

Scattering from a dielectric sphere is presented in Section 11.5.1, where bistatic RCS of a sphere is calculated using the scattered field formulation. The same example is repeated here by employing the TF/SF formulation. The RCS on *xy*, *xz*, and *yz* planes are obtained the same as the ones shown in Figures 11.6, 11.7, and 11.8, respectively, using the TF/SF formulation. This example is repeated using both formulations on a personal computer with Intel(R) Core(TM)2 Quad CPU Q9550 running at 2.83 GHz and calculation times are recorded as 10 min using the scattered field formulation while it is recorded as about 7 minutes using the TF/SF formulation.

The electric fields on a *xz* plane are captured and displayed in Figure 12.6 at progressive time steps of the simulation with TF/SF formulation. One can notice the discontinuity on the TF/SF boundary between the total and scattered fields in these images.

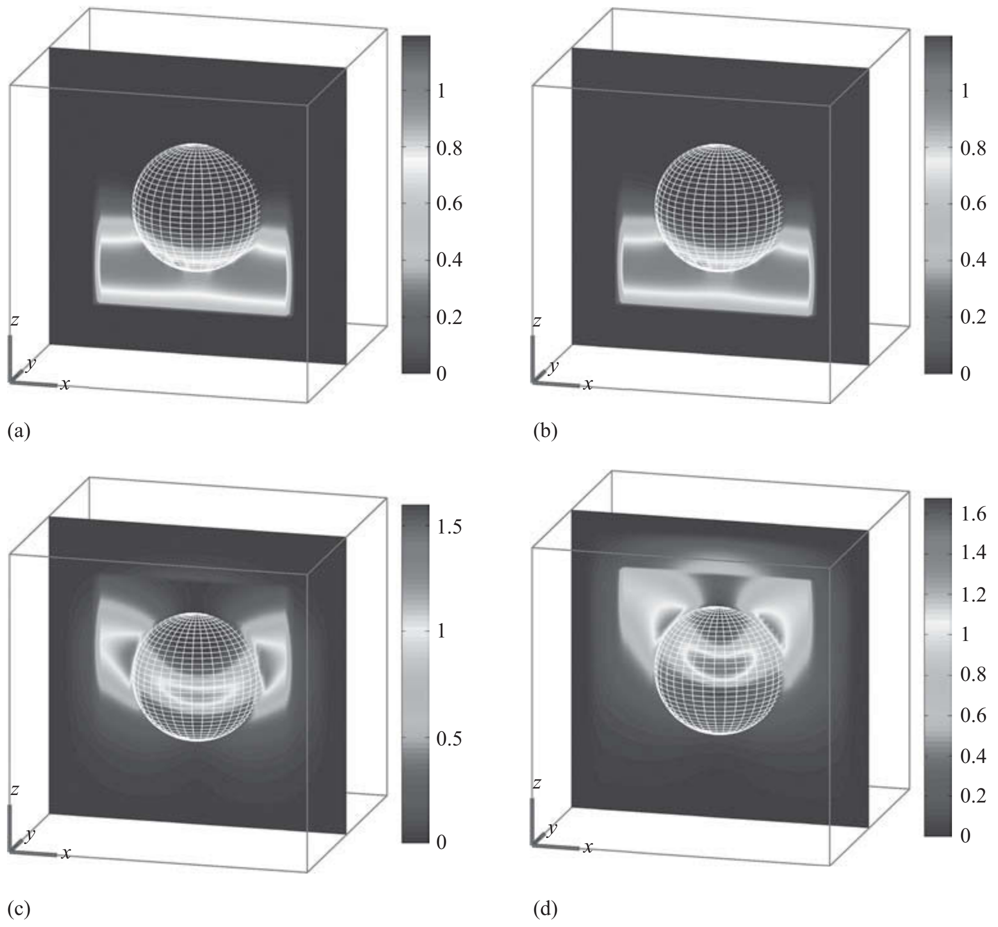

Figure 12.6 Electric field distribution on the *xz* plane: (a) at time step 120; (b) at time step 140; (c) at time step 160; and (d) at time step 180.

12.4 Exercises

12.1 Consider the problem in Section 12.3.2. The outputs definition file ***define_output_parameters*** includes the definition of far-field calculation parameters. The variable that indicates the position of the imaginary NF–FF surface **is number_of_cells_from_outer_boundary**. This variable can be set such that the NF–FF surface is inside the TF/SF boundary or outside the TF/SF boundary. If the NF–FF surface is inside the TF/SF boundary the total field is used for far-field calculations, otherwise, the scattered field is used for calculations. Run the simulation and obtain the RCS of the sphere at 1 GHz once using the total field and once using the scattered field. Verify that the RCS data are the same in both cases. You should obtain the same results as the ones in Figures 11.6–11.8. Make sure that the NF–FF surface does not overlap with the CPML boundaries. Notice that this exercise is similar to Exercise 11.1.

12.2 Consider the dipole antenna in Exercise 9.1. Replace the voltage source at the terminal of the antenna by 50°Ohms resistor. Run the simulation using a theta polarized plane wave that propagates in the theta $= 90°$ direction. Capture sampled voltage at the terminal of the dipole and store it as V_{90°. Then, rerun the simulation using a theta polarized plane wave that propagates in the theta $= 45°$ direction, capture sampled voltage at the terminal of the dipole and store it as V_{45°. Transform the sampled voltages to frequency domain and find their ratio $V_R = V_{45^\circ}/\mathrm{V}_{90^\circ}$.

At very low frequencies, the dipole antenna acts like an ideal dipole. Verify that the voltage ratio V_R is $\sin(\theta)$, the radiation pattern expression of an ideal dipole, where $\theta = 45°$.

At 4.5 GHz, the dipole antenna acts like a half-wave dipole. Verify that the voltage ratio V_R is $\cos\left(\frac{\pi}{2}\cos\theta\right)/\sin(\theta)$, the radiation pattern expression of a half-wave dipole, where $\theta = 45°$.