Angora User's Guide

A finite-difference time-domain (FDTD) electromagnetic simulation software
for version 0.18.5 and later, May 2012

İlker R. Çapoğlu

Table of Contents

Angora: A finite-difference time-domain simulation package 1

1 Getting Started 2

2 Downloading 6

3 Compilation and Installation 7
- 3.1 Enabling MPI Support 8
- 3.2 Building the Documentation 8

4 Execution 9
- 4.1 Parallel Execution 9
- 4.2 Check Mode 9

5 Configuration Format 10
- 5.1 Variable Assignment 10
- 5.2 Variable Types 10
  - 5.2.1 Integer Values 10
  - 5.2.2 Floating-Point Values 10
  - 5.2.3 Boolean Values 11
  - 5.2.4 String Values 11
  - 5.2.5 Groups 11
  - 5.2.6 Arrays 12
  - 5.2.7 Lists 12
  - 5.2.8 Comments 12
  - 5.2.9 Include Directives 13

6 Configuration Variables 14
- 6.1 Template Configuration File 14
- 6.2 Grid Properties 14
  - 6.2.1 Courant Factor 15
  - 6.2.2 Spatial Step Size 15
  - 6.2.3 Grid Dimensions 15
  - 6.2.4 Perfectly-Matched Layer (PML) 15
  - 6.2.5 Number of Time Steps 16
  - 6.2.6 Coordinate Origin 16
  - 6.2.7 Dynamic Range 17
- 6.3 Shapes 18
  - 6.3.1 Rectangular Boxes 19

6.3.2 Spheres ... 20
6.4 Materials ... 21
6.5 Simulation Space ... 22
6.5.1 Objects ... 23
6.5.2 Planar Layers ... 24
6.5.3 Random Materials ... 25
6.5.4 File Input ... 29
6.5.5 Ground Planes ... 33
6.6 Waveforms ... 34
6.6.1 Gaussian Waveforms ... 35
6.6.2 Differentiated-Gaussian Waveforms ... 36
6.6.3 Modulated-Gaussian Waveforms ... 37
6.7 Point Sources ... 38
6.8 Near-Field-to-Far-Field Transformer ... 40
6.8.1 Time-Domain Near-Field-to-Far-Field-Transformer ... 40
6.8.1.1 HDF5 Content of Time-Domain NFFFT Output ... 44
6.8.2 Phasor-Domain Near-Field-to-Far-Field-Transformer ... 45
6.8.2.1 HDF5 Content of Phasor-Domain NFFFT Output ... 52
6.9 Optical Imaging ... 53
6.9.1 Optical Image File HDF5 Content ... 61
6.10 Incident Beams ... 62
6.10.1 Plane Waves ... 63
6.10.2 Focused Laser Beams ... 67
6.11 Recording ... 73
6.11.1 Movie Recording ... 74
6.11.1.1 Movie File Format ... 77
6.11.2 Line Recording ... 78
6.11.2.1 Line File Format ... 81
6.11.3 Field-Value Recording ... 82
6.11.3.1 Field-Value File HDF5 Content ... 84
6.12 Paths ... 85
6.13 Logging ... 85
6.14 Multiple Simulation Runs ... 86
6.15 Miscellaneous ... 88
6.15.1 Auto-Saving the Configuration ... 88

7 References ... 89

List of Figures ... 90

Indices ... 91

Configuration Variable Index ... 91
Concept Index ... 93

Angora: A finite-difference time-domain simulation package

This is the user's guide for Angora, a software package that computes numerical solutions to electromagnetic radiation and scattering problems. It is based on the finite-difference time-domain (FDTD) method, which one of the most popular approaches for solving Maxwell's equations of electrodynamics.

1 Getting Started

Angora simulations are run by constructing a text file, called the *configuration file* that specifies all aspects of the simulation. This file is then given as a command-line option to the Angora executable `angora`; which reads the configuration file and produces the desired output (see Chapter 4 [Execution], page 9).

Let's start with a simple example. In the following, we will show how Angora can be used to solve the problem of electromagnetic scattering from a sphere illuminated by a plane wave. The geometry of the scattering problem is shown in Figure 1.1.

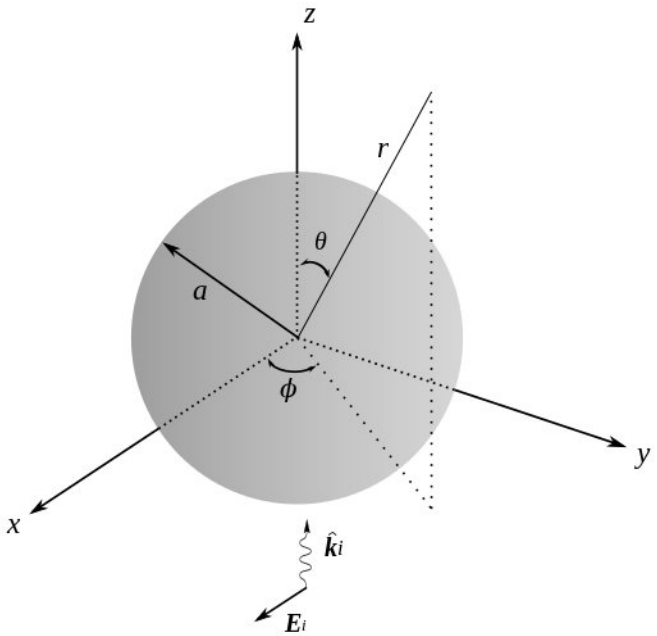

Figure 1.1: Scattering from a sphere illuminated by a plane wave incident from the -z direction.

We start by creating a configuration file; say '`sph_sc.cfg`'. This file will be populated by configuration options listed in the following. Some basic parameters of our simulation are determined by the following lines (see Section 6.2 [Grid Properties], page 14 for details):

```
dx = 20e-9;
courant = 0.98;
grid_dimension_x = 1e-6;
grid_dimension_y = 1e-6;
grid_dimension_z = 1e-6;
pml_thickness_in_cells = 5;
num_of_time_steps = 1500;
```

The first variable, `dx`, is the uniform spatial step size in the FDTD discretization. The second variable, `courant`, is the ratio of the time step to the maximum time step allowable by the Courant condition. The next three variables determine the physical size of the simulation grid in meters. The thickness of the absorbing layer (PML) is determined by the `pml_thickness_in_cells` variable. The last line specifies the number of time steps in the simulation.

The sphere from which the electromagnetic plane wave will be scattered is created in two steps. First, we define a spherical "shape object" using the `Spheres` variable (see Section 6.3.2 [Spheres], page 20):

```
Shapes:
{
    Spheres:
    (
        {
            shape_tag = "mysphere";
            center_coord_x = 0;
            center_coord_y = 0;
            center_coord_z = 0;
            radius = 320e-9;
        }
    );
};
```

Next, the material filling the sphere is defined using the `Materials` variable (see Section 6.4 [Materials], page 21):

```
Materials:
(
    {
        material_tag = "sph_mat";
        rel_permittivity = 2.25;
        electric_conductivity = 3e4; //in Siemens/m
        rel_permeability = 1.7;
        magnetic_conductivity = 4.2578e9; //in Ohm/m
    }
);
```

The shape and material definitions are then combined in the `Objects` variable, and the sphere is placed in the grid (see Section 6.5.1 [Objects], page 23):

```
SimulationSpace:
{
    Objects:
    (
        {
            material_tag = "sph_mat";
            shape_tag = "mysphere";
        }
    );
};
```

With the above defitions, we have created a sphere of radius 320 nm and made of the material specified by `"sph_mat"`. Next, we define the waveform of the incident plane wave using the `Waveforms` variable:

```
Waveforms:
{
```

```
    ModulatedGaussianWaveforms:
    (
        {
            waveform_tag = "mywaveform";
            modulation_type = "sine";
            tau = 2.12662e-15;
            f_0 = 5.88878e14;
        }
    );
};
```

We then create the plane wave incident from the -z direction with the above waveform as its electric field using the `PlaneWaves` variable (see Section 6.10.1 [Plane Waves], page 63):

```
TFSF:
{
    PlaneWaves:
    (
        {
            theta = 180;
            phi = 0;
            psi = 90;
            waveform_tag = "mywaveform";
        }
    );
};
```

Finally, we create a near-field-to-far-field transformer to calculate the scattered field in the far zone using the `PhasorDomainNFFFT` variable (see Section 6.8 [Near-Field-to-Far-Field Transformer], page 40):

```
PhasorDomainNFFFT:
(
    {
        num_of_lambdas = 1;
        lambda_min = 509.09e-9;
        lambda_max = 509.1e-9;
        direction_spec = "theta-phi";
        num_of_dirs_1 = 360;
        dir1_min = 0;
        dir1_max = 360;
        num_of_dirs_2 = 1;
        dir2_min = 0;
        dir2_max = 0;
    }
);
```

With the above definitions, the far field is calculated at the free-space wavelength 509.1 nm, and 360 equally-spaced angles on the xz plane. The output of the near-field-to-far-field transformer is in HDF5 format, which can be read and manipulated using freely-available tools. For more information, see Section 6.8 [Near-Field-to-Far-Field Transformer], page 40.

The absolute value of the phasor component of the far-zone electric field (normalized by 1/r) at 509.1 nm is shown in a polar plot in Figure 1.2.

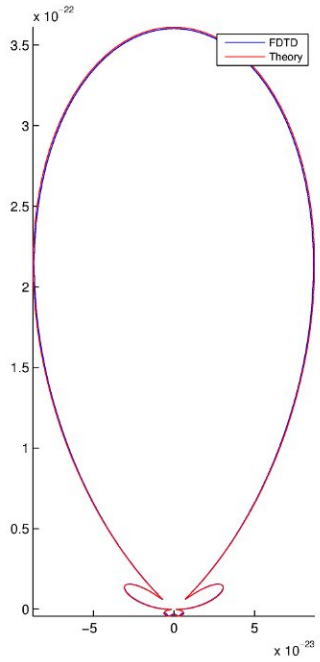

Figure 1.2: The absolute scattered electric field phasor amplitude on the xz plane at 509.1 nm.

The scattered electric field can also be obtained theoretically using Mie theory (see [Matzler02], page 89), which is shown alongside the Angora solution in the above figure.

2 Downloading

Angora is currently only available for the **GNU/Linux operating system**. If you would like to port Angora to another operating system, please contact us at capoglu@angorafdtd.org. Contributions are always welcome.

The latest version of Angora can be found at http://angorafdtd.org in source code format, as well as binary format for x86_64 GNU/Linux systems.

3 Compilation and Installation

If you will be using Angora on a 64-bit x86 architecture with the GNU/Linux operating system, you can simply download the binary version (both non-parallel and OpenMPI-based parallel versions available) from the Angora website and start running simulations right away.

If there is no precompiled Angora binary available for your system, you will have to compile it from source. You will require the the following libraries on your system to compile Angora: blitz++, libconfig, hdf5, and boost. If possible, use the package manager for your specific GNU/Linux distribution (such as Synaptic in Ubuntu) to install the libraries directly from the package repository. Most major distributions provide these libraries in their package repositories. If you do not have root access to your system, you can install these libraries in your home directory. The installation instructions for the libraries usually provide detailed information on how to do this. For local installation, the usual trick is to set the installation path by specifying the `prefix` variable in the `Makefile`s. This is done either by using the '`--prefix=`*`local-path`*' option when calling the package's `configure` script, or customizing `make` at the final stage with the '`prefix=`*`local-path`*' command option.

Once the dependency libraries are installed, the Angora package is ready for compilation. Extract the package '`angora-`*`package-version`*`.tar.gz`' using `tar`, and enter the created directory:

```
johndoe@mysystem:~$ tar xvf angora-package-version.tar.gz
johndoe@mysystem:~$ cd angora-package-version
```

Run the `configure` script in this directory to create the `Makefile`s required to build the package:

```
johndoe@mysystem:~/angora-package-version$ ./configure
```

If any of the dependency libraries was installed in a local directory, then add the option '`--with-`*`library-name=local-path-to-library`*' to the above command line. For example, if the `blitz++` library was installed in '`/home/johndoe/blitz-0.9`', then the option to add is '`--with-blitz=/home/johndoe/blitz-0.9`'. Type '`./configure --help`' in the directory '`angora-`*`package-version`*' for information on specifying the paths to the other dependency libraries.

After the `configure` script finishes execution, compile and install Angora using the `make` command:

```
johndoe@mysystem:~/angora-package-version$ make
```

If your system has multiple cores, you can speed up the compilation by executing `make` in parallel. For example, you can use all 4 cores of your system by typing, instead of the above line,

```
johndoe@mysystem:~/angora-package-version$ make -j 4
```

This might take a couple of minutes, depending on your system. After `make` finishes, the executable `angora` will be located in the directory '`angora-`*`package-version`*'. If you wish to *install* the package globally so that it can be run from anywhere, type

```
johndoe@mysystem:~/angora-package-version$ sudo make install
```

Obviously, this requires super-user privileges on your system. By default, the package is installed in '`/usr/local`'; so the binary will reside in '`/usr/local/bin`'. If you don't have super-user privileges, you can install Angora in a local directory '`full-path-to-inst-dir`' by typing

```
johndoe@mysystem:~/angora-package-version$ make prefix=full-path-to-inst-
dir install
```

The location '`full-path-to-inst-dir`' should be an absolute path. After this, the binary `angora` will be located in the directory '`full-path-to-inst-dir/bin/`'.

3.1 Enabling MPI Support

Parallel execution on multiple processors or cores is supported by Angora, provided that the MPI (Message Passing Interface) libraries are installed on your system (e.g., OpenMPI or MPICH2 or other). A precompiled binary version of Angora based on the OpenMPI implementation is available on the Angora website.

If you are compiling Angora from source, you'll have to enable the MPI feature at compile time. This feature is *disabled* by default. You can enable MPI functionality in Angora by adding the option '`--with-mpi`' to the `configure` command line:

```
johndoe@mysystem:~/angora-package-version$ ./configure --with-mpi
```

For more information on launching Angora simulations on multiple processors or cores using MPI, see Section 4.1 [Parallel Execution], page 9.

3.2 Building the Documentation

The GNU `info` documentation for Angora is automatically built and installed by `make`. If you have the texi2html and latex2html utilities installed, you can create an HTML version of the Angora documentation by typing

```
johndoe@mysystem:~/angora-package-version$ make html
```

If you have the `texi2dvi` command available (provided as part of the GNU Texinfo package), you can also build a PDF version of the Angora documentation by typing

```
johndoe@mysystem:~/angora-package-version$ make pdf
```

Once built, both the HTML and PDF versions of the documentation will be located in the subdirectory '`doc/`'.

4 Execution

Angora operates by reading a text file, called the *configuration file*, that specifies the details of the simulation. Every aspect of the simulation is configured by a related *configuration variable* (or *variable* in short) in the configuration file; which comprises either a single line or a number of lines. In general, an Angora simulation is run by putting the name of the configuration file pertaining to the simulation as a command line option when calling the `angora` executable:

```
johndoe@mysystem:~/angora-package-version$ ./angora path-to-config-file
```

If the Angora executable is run without any command-line options, it looks for the configuration file named '`angora.cfg`' in the same directory from which the executable is run. See Chapter 6 [Configuration Variables], page 14, for details on configuration files.

4.1 Parallel Execution

If Angora is compiled with MPI support, then the standard MPI launcher (`mpirun`) can be used to execute the Angora binary `angora` in parallel:

```
johndoe@mysystem:~/angora-package-version$ mpirun -n num-of-processors
./angora path-to-config-file
```

For example, to run the simulation configured by '`mysimulation.cfg`' using Angora version `0.9` on 8 processors, one should type

```
johndoe@mysystem:~/angora-package-version$ mpirun -n 8 ./angora mysimulation.cfg
```

MPI support should be enabled in compile time in order to run simulations in parallel. For details, see Section 3.1 [Enabling MPI Support], page 8. If you are using the OpenMPI-based precompiled binary version of Angora, then the OpenMPI shared libraries must be in your path before the binary '`angora`' can be executed. This can either be done by installing OpenMPI globally (using a package manager etc.), or adding the path to the OpenMPI shared libraries to your `LD_LIBRARY_PATH` environment variable.

4.2 Check Mode

Angora can check a configuration file for syntactic and semantic errors, without actually running the simulation. To do this, simply run Angora with the '`--check`' or '`-c`'option:

```
johndoe@mysystem:~/angora-package-version$ ./angora --check path-to-config-
file
```

This reports any errors in the configuration-file syntax or invalid configuration options (see Section 6.1 [Template Configuration File], page 14). The actual size of the simulation does not have any effect on this operation; therefore it can be run on a single processor with little memory.

5 Configuration Format

Angora uses the libconfig library to read configuration variables regarding the simulation from a text file. The text file, called the *configuration file*, has to conform to the libconfig grammar; which is explained in greater detail at `http://www.hyperrealm.com/libconfig/libconfig_manual.html`. Here, we will provide the minimum information necessary to write configuration files for Angora simulations.

5.1 Variable Assignment

A variable in a configuration file is set using the following assignment:

```
name=value;
```

or:

```
name:value;
```

The trailing semicolon is required. Whitespace is not significant. Here, *name* is the name of the variable, and *value* is its value; which may be a scalar value, an array, a group, or a list. See Section 5.2 [Variable Types], page 10, for information on these value types.

The order in which variables are specified in the configuration file is insignificant, except within the `SimulationSpace` variable (see Section 6.5 [Simulation Space], page 22). The sub-variables of the `SimulationSpace` variable are processed in the order of appearance in the configuration file. This is necessary because the user needs to be able to control the order in which objects are placed in the grid, and predict the regions within an object that will be overwritten by another object.

5.2 Variable Types

Angora simulation variables can be assigned C++-type *scalar* values, as well as more complex values of type *group*, *array*, and *list*. The latter types are defined by the libconfig library. Some of the text in this section is copied verbatim from the libconfig manual. The `libconfig` library, along with its documentation, is distributed under the GNU Lesser Public License.

5.2.1 Integer Values

Integers can be represented in one of two ways: as a series of one or more decimal digits (‘`0`’ - ‘`9`’), with an optional leading sign character (‘`+`’ or ‘`-`’); or as a hexadecimal value consisting of the characters ‘`0x`’ followed by a series of one or more hexadecimal digits (‘`0`’ - ‘`9`’, ‘`A`’ - ‘`F`’, ‘`a`’ - ‘`f`’).

Examples:

```
n_sx = 3;
offset = -4;
address = 0xFFFF;
```

5.2.2 Floating-Point Values

Floating point values consist of a series of one or more digits, one decimal point, an optional leading sign character (‘`+`’ or ‘`-`’), and an optional exponent. An exponent consists of the letter ‘`E`’ or ‘`e`’, an optional sign character, and a series of one or more digits.

Except in special circumstances, floating-point values in Angora are read and processed in '`double`' precision, which corresponds to roughly 15 decimal digits.

Examples:

```
f = 1.0;
origin = -3e-6;
prefactor = 5E10;
```

5.2.3 Boolean Values

Boolean values may have one of the following values: '`true`', '`false`', or any mixed-case variation thereof.

Examples:

```
include_first_value = true;
include_last_value = FaLsE;
```

5.2.4 String Values

String values consist of arbitrary text delimited by double quotes. Literal double quotes can be escaped by preceding them with a backslash: '`\"`'. The escape sequences '`\\`', '`\f`', '`\n`', '`\r`', and '`\t`' are also recognized, and have the usual meaning.

In addition, the '`\x`' escape sequence is supported; this sequence must be followed by *exactly two* hexadecimal digits, which represent an 8-bit ASCII value. For example, '`\xFF`' represents the character with ASCII code 0xFF.

No other escape sequences are currently supported.

Adjacent strings are automatically concatenated, as in C/C++ source code. This is useful for formatting very long strings as sequences of shorter strings. For example, the following constructs are equivalent:

- `"The quick brown fox jumped over the lazy dog."`
- `"The quick brown fox"`
  `" jumped over the lazy dog."`
- `"The quick" /* comment */ " brown fox " // another comment`
  `"jumped over the lazy dog."`

5.2.5 Groups

A group has the form:

```
{
   name=value;
   other_name=other_value;
   ...
}
```

Notice the curly brackets '`{}`' around the variable assignments. Groups can contain any number of variable assignments (see Section 5.1 [Variable Assignment], page 10), but each variable must have a unique name within the group.

Example:

```
{
    shape_tag = "mysphere";
    center_coord_x = 5e-6;
    center_coord_y = 5e-6;
    center_coord_z = 5e-6;
    radius = 4e-6;
}
```

5.2.6 Arrays

An array has the form:

```
[ value, value, ... ]
```

Notice the square brackets '`[]`' delimiting the comma-separated values. An array may have zero or more elements, but the elements must all be **scalar** values of the **same type**.

Examples:

```
disabled_runs = [0,1,3];
output_variables = ["Ex","Ey"];
```

5.2.7 Lists

A list has the form:

```
( value, value, ... )
```

Notice the parantheses '`()`' delimiting the comma-separated values. A list may have zero or more elements, each of which can be a scalar value, an array, a group, or another list. The values in a list can be of *different types*; however, in Angora, the list type is exclusively used to contain a collection of *group* values. In Angora, the list type semantically represents a collection of objects, each with a collection of properties set within their respective group value. Here is an example:

```
Materials:
(
    {
        material_tag = "mat1";
        rel_permittivity = 2.0;
    },
    {
        material_tag = "mat2";
        rel_permittivity = 2.5;
    }
);
```

Here, the list structure named `Materials` contains two groups (each delimited by curly brackets '`{}`') separated by a comma. This defines two materials with different sets of properties.

5.2.8 Comments

Three types of comments are allowed within a configuration:

- Script-style comments. All text beginning with a '`#`' character to the end of the line is ignored.

- C-style comments. All text, including line breaks, between a starting '`/*`' sequence and an ending '`*/`' sequence is ignored.
- C++-style comments. All text beginning with a '`//`' sequence to the end of the line is ignored.

As expected, comment delimiters appearing within quoted strings are treated as literal text.

```
# Here's a comment
MyGroup:
(/* This is
    also a comment */
    {
        this_property = "myvalue";
        // Another comment
    }
);
```

5.2.9 Include Directives

A configuration file may "include" the contents of another file using an *include directive*. This directive has the effect of inlining the contents of the named file at the point of inclusion.

An include directive must appear on its own line in the input. It has the form:

`@include "`*`filename`*`"`

Any backslashes or double quotes in the file name must be escaped as '`\\`' and '`\"`', respectively.

For example, consider the following two configuration files:

```
# file: limits.cfg
back_coord_x = -5e-6;
front_coord_x = 6e-6;
left_coord_y = -5e-6;
right_coord_y = 6e-6;
lower_coord_z = -3e-6;
upper_coord_z = 4e-6;
```

```
# file: mysim.cfg
RectangularBoxes:
(
    {
        shape_tag = "mybox";
        @include "limits.cfg"
    }
);
```

Include files may be nested to a maximum of 10 levels; exceeding this limit results in a runtime error.

6 Configuration Variables

The variable assignments (or *settings* in libconfig terminology) in a configuration file reside either at the uppermost level (called the *Global* level) or within a group structure (see Section 5.2.5 [Groups], page 11). In the following, configuration variables will be characterized as either being a *Global variable*, or a *Sub-variable of ParentVariable*; where *ParentVariable* is the next parent variable upward in the hierarchy that has a name. The variable *ParentVariable* can either be a *group* or a *list* (see Section 5.2.5 [Groups], page 11 and Section 5.2.7 [Lists], page 12). Quite often, the immediate parent of a variable assignment is an unnamed group; therefore the *ParentVariable* of that assignment is the list that contains this unnamed group. For example, the *ParentVariable* of the variable `material_tag` in the example in Section 5.2.7 [Lists], page 12 is `Materials`, since its immediate parent is an unnamed group, but the list structure containing the unnamed group has a name (which is `Materials`). On the other hand, the variable `Materials` is a *Global variable*; since it is assigned at the uppermost level in a configuration file, outside any enclosing structure.

The configuration variable names are case sensitive; meaning that `Materials` and `materials` are not the same.

Angora throws an error message for any missing variable or misspelled variable name. This is crucial for ensuring that no optional configuration variable is omitted because of a typo. The valid variable names are read into the Angora source code in compile time from a template file '`config_all.cfg`'. This file, although not required at the time of execution, is distributed with Angora for reference (see Section 6.1 [Template Configuration File], page 14).

6.1 Template Configuration File

A file named '`config_all.cfg`' is included in the Angora distribution, which includes all the valid configuration variables. All variables in a given configuration file are checked against '`config_all.cfg`' and labeled invalid if a corresponding variable does not exist in '`config_all.cfg`'. It should be stressed, however, that '`config_all.cfg`' is **not** necessary for the execution of Angora, but *is* necessary for its compilation. This is because '`config_all.cfg`' is read into the source code of Angora in the compilation stage. The file '`config_all.cfg`' is only distributed as a reference for the user's convenience.

The file '`config_all.cfg`' is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'.

6.2 Grid Properties

Angora currently only supports a rectangular, Cartesian FDTD grid with equal grid spacing in the x, y, and z directions. Mesh refinement is not yet supported; therefore the grid spacing is uniform across the grid.

6.2.1 Courant Factor

`floating-point courant` [Global variable]

Angora adopts a slightly modified form for the Courant factor, defined as

$$\sqrt{3}\frac{c\Delta t}{\Delta x}$$

where c=299792458 m/s is the speed of light in vacuum, and Δt and Δx are the *temporal* and *spatial* step sizes (see Section 6.2.2 [Spatial Step Size], page 15). The Courant factor should be less than `1.0` for stability. A common value for `courant` is `0.98`.

6.2.2 Spatial Step Size

`floating-point dx` (*units: m*) [Global variable]

The spatial step size in the FDTD grid is specified by the `dx` variable. Currently only cubic FDTD cells are supported; therefore the spatial step sizes in the x, y, and z direction are all determined by `dx`.

6.2.3 Grid Dimensions

`floating-point grid_dimension_x` (*units: m*) [Global variable]

`floating-point grid_dimension_y` (*units: m*) [Global variable]

`floating-point grid_dimension_z` (*units: m*) [Global variable]

`integer grid_dimension_x_in_cells` [Global variable]

`integer grid_dimension_y_in_cells` [Global variable]

`integer grid_dimension_z_in_cells` [Global variable]

These variables determine the size of the Cartesian FDTD grid. The dimensions of the grid can be specified either in meters, or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the dimensions are given in meters, the number of FDTD cells in the Cartesian FDTD grid in the x, y, and z directions are rounded to the closest integer. If no perfectly-matched layers are specified (see Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15), the total number of FDTD cells in the three-dimensional FDTD grid is equal to (grid_dimension_x_in_cells) x (grid_dimension_y_in_cells) x (grid_dimension_z_in_cells).

6.2.4 Perfectly-Matched Layer (PML)

`floating-point pml_thickness` (*units: m*) [Global variable]

`integer pml_thickness_in_cells` [Global variable]

This variable sets the thickness of the perfectly-matched layers (PMLs) around the grid in all directions. Further customization of the PML thickness is not yet supported. The thickness can be specified either in meters, or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

Typical PML thicknesses are `5` to `10` grid cells. If you do not want to place a PML layer around the grid, just assign `pml_thickness=0`. Without a PML layer, the boundary of

the FDTD grid acts as a perfect electric conductor (PEC). Other boundary conditions (perfect magnetic conductor, periodic, etc.) will also be supported in the future.

With a PML definition, the total number of FDTD cells in the three-dimensional FDTD grid becomes

(grid_dimension_x_in_cells+2*pml_thickness_in_cells)

x (grid_dimension_y_in_cells+2*pml_thickness_in_cells)

x (grid_dimension_z_in_cells+2*pml_thickness_in_cells)

The computational burden per FDTD cell associated with the PML layer is roughly three times that of the main grid.

Angora implements the convolution PML (CPML) formulation of the complex-frequency shifted (CFS) PML (see [Roden00], page 89; [Kuzuoglu96], page 89.)

`floating-point cpml_feature_size` (*units:m, default:* `max(grid_dimension_x,grid_dimension_y,grid_dimension_z)`) [Global variable]

`floating-point cpml_feature_size_in_cells` (*default:* `max(grid_dimension_x_in_cells,grid_dimension_y_in_cells,grid_dimension_z_in_cells)`) [Global variable]

This variable specifies the maximum size of the scattering or radiating structure in the FDTD grid. This size can be specified either in meters, or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

This information is used to determine the frequency-shifting parameter α in the CFS-PML formulation. Following Berenger's derivation (see [Berenger02], page 89), this parameter is defined as

$$\alpha = c\epsilon/w$$

where c is the velocity of propagation in the medium, ϵ is the absolute permittivity (in F/m) in the medium, and w is the maximum size of the structure.

The above relationship follows essentially from the low-frequency behavior of the CFS-PML. At low frequencies where the evanescent field around the structure dominates, the CFS-PML reduces to a real stretch of coordinates without any absorption. This helps the termination of evanescent fields, which are poorly handled by ordinary PMLs.

6.2.5 Number of Time Steps

`integer num_of_time_steps` [Global variable]

This variable determines the number of time steps in the FDTD simulation.

6.2.6 Coordinate Origin

`floating-point origin_x` (*units:m, default:* (*grid_dimension_x*+*2***pml_thickness*)/*2*+*1*) [Global variable]

`floating-point origin_y` (*units:m, default:* (*grid_dimension_y*+*2***pml_thickness*)/*2*+*1*) [Global variable]

`floating-point origin_z` (*units:m, default:* (*grid_dimension_z*+2**pml_thickness*)/2+1) [Global variable]

`integer origin_x_in_cells` (*default:* (*grid_dimension_x_in_cells*+2**pml_thickness_in_cells*)/2+1) [Global variable]

`integer origin_y_in_cells` (*default:* (*grid_dimension_y_in_cells*+2**pml_thickness_in_cells*)/2+1) [Global variable]

`integer origin_z_in_cells` (*default:* (*grid_dimension_z_in_cells*+2**pml_thickness_in_cells*)/2+1) [Global variable]

These variables set the origin of the coordinate system in the simulation. All other coordinates in a configuration file are taken as relative to this origin. The coordinates can be specified either in meters, or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. These three numbers represent the Cartesian coordinates of the origin from the back-left-lower corner of the grid. In Figure 6.1, the location of the coordinate origin in the FDTD grid is shown for (`origin_x_in_cells`,`origin_y_in_cells`,`origin_z_in_cells`)=(2,3,2). The FDTD grid is composed of (3x5x3) grids, and only the back (y=z=0), left (x=z=0), and lower (x=y=0) surfaces are shown in the figure.

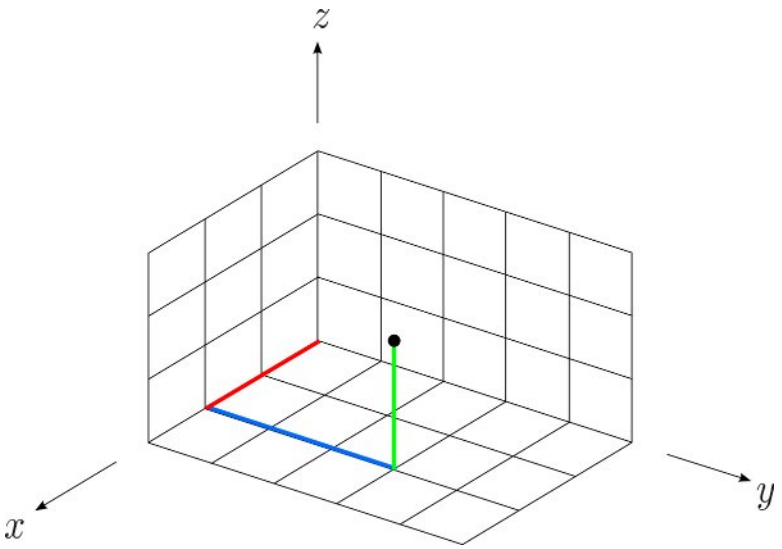

Figure 6.1: The location of the coordinate origin in the FDTD grid for (`origin_x_in_cells`,`origin_y_in_cells`,`origin_z_in_cells`)=(2,3,2).

If the coordinates are given in meters, they are rounded to the closest integer multiple of the spatial step size (see Section 6.2.2 [Spatial Step Size], page 15).

6.2.7 Dynamic Range

The following two variables are only relevant in movie recording (see Section 6.11.1 [Movie Recording], page 74), wherein the floating-point field values on the movie frames are sometimes discretized to fit into 1 byte.

`floating-point max_field_value` (*default:* `1.0`) [Global variable]
This value specifies the maximum field value used in the discretization for 1-byte movie recording (see Section 6.11.1 [Movie Recording], page 74).

`floating-point dB_accuracy` (*default: automatic*) [Global variable]
This value specifies the dynamic range (in dB) to be used in the discretization for 1-byte movie recording (see Section 6.11.1 [Movie Recording], page 74). For example,

```
dB_accuracy = -60;
```

tells Angora to discretize the field values in a dynamic range between the maximum field value (specified by `max_field_value` above) and 60dB below that value. If `dB_accuracy` is not specified, Angora tries to set this value automatically, based on its best guess on the useful accuracy range in the simulation. This value can also be read from the output of the movie recorder (see Section 6.11.1 [Movie Recording], page 74).

6.3 Shapes

`group Shapes` [Global variable]
In Angora, a geometrical shape and the material filling that shape are two distinct and independent elements of the definition of an object. The first of these elements is defined in the `Shapes` variable, which is a group (see Section 5.2.5 [Groups], page 11).

```
Shapes:
{
    RectangularBoxes:
    (
        ...
        ...
    );
    Spheres:
    (
        ...
        ...
    );

    ...
    ...

};
```

In this example, two sub-variables `RectangularBoxes` and `Spheres` of the `Shapes` group are shown. These are both list variables (see Section 5.2.7 [Lists], page 12).

Currently, rectangular boxes and spheres are the only basic shape classes defined in Angora. Unions, intersections, and geometrical transformations of shapes, as well as more basic shape classes will be added to Angora in the future. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

6.3.1 Rectangular Boxes

`group RectangularBoxes` [Sub-variable of Shapes]

Rectangular boxes are defined using the `RectangularBoxes` variable, which is a list structure under the `Shapes` group.

```
Shapes:
{
    RectangularBoxes:
    (
        {
            shape_tag = "mybox";
            back_coord_x = -5e-6;
            front_coord_x = 6e-6;
            left_coord_y = -5e-6;
            right_coord_y = 6e-6;
            lower_coord_z = -3e-6;
            upper_coord_z = 4e-6;
        },
        {
            ...
            ...
        }
    );
};
```

In this example, two rectangular box shapes are defined in two respective unnamed groups; only the first being shown in complete detail.

`string shape_tag` [Sub-variable of RectangularBoxes]

This string variable assigns a *name* to the particular shape, so it can be referred to later in the configuration file.

`floating-point back_coord_x` (*units: m*) [Sub-variable of RectangularBoxes]

`floating-point front_coord_x` (*units: m*) [Sub-variable of RectangularBoxes]

`floating-point left_coord_y` (*units: m*) [Sub-variable of RectangularBoxes]

`floating-point right_coord_y` (*units: m*) [Sub-variable of RectangularBoxes]

`floating-point lower_coord_z` (*units: m*) [Sub-variable of RectangularBoxes]

`floating-point upper_coord_z` (*units: m*) [Sub-variable of RectangularBoxes]

`floating-point back_coord_x_in_cells` [Sub-variable of RectangularBoxes]

`floating-point` [Sub-variable of RectangularBoxes]
`front_coord_x_in_cells`

`floating-point` [Sub-variable of RectangularBoxes]
`left_coord_y_in_cells`

`floating-point` [Sub-variable of RectangularBoxes]
`right_coord_y_in_cells`

`floating-point` [Sub-variable of RectangularBoxes]
`lower_coord_z_in_cells`

`floating-point` [Sub-variable of RectangularBoxes]
`upper_coord_z_in_cells`

These variables determine the minimum and maximum Cartesian coordinates of the box in the x, y, and z directions relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

6.3.2 Spheres

`group Spheres` [Sub-variable of Shapes]

Spheres are defined using the `Spheres` variable, which is a list structure under the `Shapes` group.

```
Shapes:
{
    Spheres:
    (
        {
            shape_tag = "mysphere";
            center_coord_x = 5e-6;
            center_coord_y = 5e-6;
            center_coord_z = 5e-6;
            radius = 4e-6;
        },
        {
            ...
            ...
        }
    );
};
```

In this example, two spherical shapes are defined in two respective unnamed groups; only the first being shown in complete detail.

`string shape_tag` [Sub-variable of Spheres]

This string variable assigns a *name* to the particular shape, so it can be referred to later in the configuration file.

`floating-point center_coord_x` (*units: m*) [Sub-variable of Spheres]

`floating-point center_coord_y` (*units: m*) [Sub-variable of Spheres]

`floating-point center_coord_z` (*units: m*) [Sub-variable of Spheres]

`floating-point center_coord_x_in_cells` [Sub-variable of Spheres]

`floating-point center_coord_y_in_cells` [Sub-variable of Spheres]

`floating-point center_coord_z_in_cells` [Sub-variable of Spheres]

These variables determine the Cartesian coordinate of the center of the sphere relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`floating-point radius` (*units: m*) [Sub-variable of Spheres]

`floating-point radius_in_cells` [Sub-variable of Spheres]

This variable determines the radius of the sphere. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

6.4 Materials

Currently, Angora only supports isotropic, non-dispersive materials. Anisotropic and dispersive materials will also be supported in the future. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

`list Materials` [Global variable]

The properties of a certain material type are specified in the `Materials` list (see Section 5.2.7 [Lists], page 12).

```
Materials:
(
    {
        material_tag = "this_material";
        rel_permittivity = 2.0;
        rel_permeability = 1.0;
        electric_conductivity = 0.0;
        magnetic_conductivity = 0.0;
        transparent = false;
    },
    {
        ...
        ...
    }
);
```

In this example, two materials are defined in two respective unnamed groups; only the first being shown in complete detail.

`string material_tag` [Sub-variable of Materials]
This string variable assigns a *name* to the particular material, so it can be referred to later in the configuration file.

`floating-point rel_permittivity` (*default:* `1.0`) [Sub-variable of Materials]
This variable specifies the relative permittivity (or the dielectric constant) of the material. In SI units, the absolute permittivity of the material is this variable multiplied by the permittivity of free space (8.85418782E-12 F/m).

`floating-point rel_permeability` (*default:* `1.0`) [Sub-variable of Materials]
This variable specifies the relative permeability (or the magnetic constant) of the material. In SI units, the absolute permeability of the material is this variable multiplied by the permeability of free space (4piE-7).

`floating-point electric_conductivity` (*units: S/m*) (*default:* `0`) [Sub-variable of Materials]
This variable specifies the electric conductivity (in Siemens/m or Mho/m) of the material.

`floating-point magnetic_conductivity` (*units: Ohm/m*) (*default:* `0`) [Sub-variable of Materials]
This variable specifies the magnetic conductivity (in Ohm/m) of the material.

`boolean transparent` (*default:* `false`) [Sub-variable of Materials]
If set to `false`, any unspecified constitutive parameter is set to its default value. If set to `true`, unspecified constitutive parameters become *transparent*, meaning that when an object made up of this material is placed in the grid, those constitutive parameters are kept unchanged.

6.5 Simulation Space

`group SimulationSpace` [Global variable]
The `SimulationSpace` group is where all the objects inside the simulation space are defined. If no `SimulationSpace` group is specified in the configuration file, the FDTD simulation space consists entirely of vacuum.

```
SimulationSpace:
{
    Objects:
    (
        ...
        ...
    );
    RandomMaterials:
    {
        ...
        ...
```

```
        };

        ...
        ...

    };
```

In the above example, only two of the sub-variables of the `SimulationSpace` group, `Objects` and `RandomMaterials`, are shown. The sub-variable `Objects` is a list (see Section 5.2.7 [Lists], page 12), whereas `RandomMaterials` is a group (see Section 5.2.5 [Groups], page 11).

The definitions in the `SimulationSpace` group are processed **in the order of placement**. Thus, the user has complete control over which object is placed in the simulation space first. As a consequence of this first-come-first-serve policy, objects can overwrite regions of the simulation space occupied by other objects.

6.5.1 Objects

`list Objects` [Sub-variable of SimulationSpace]

The `Objects` list defines material objects to be placed in the simulation grid. An *object* in this context is defined as a combination of two abstract ingredients: A previously-defined shape (see Section 6.3 [Shapes], page 18), and a previously-defined material to fill that shape (see Section 6.4 [Materials], page 21). The shape and material are referred to using their shape and material tags, which are string variables assigned to them in their definitions.

Here is an example:

```
SimulationSpace:
{
    Objects:
    (
        {
            material_tag = "this_material";
            shape_tag = "mysphere";
        },
        {
            ...
            ...
        }
    );
};
```

`string material_tag` [Sub-variable of Objects]

This string variable specifies the material that makes up the object. It should match a previously-defined tag in a `Materials` definition (see Section 6.4 [Materials], page 21).

`string shape_tag` [Sub-variable of Objects]
This string variable specifies the geometrical shape of the object. It should match a previously-defined tag in a `Shapes` definition (see Section 6.3 [Shapes], page 18).

6.5.2 Planar Layers

`list MaterialSlabs` [Sub-variable of SimulationSpace]
The purpose of the `MaterialSlab` list is to introduce **planar stratification** into the simulation grid. Currently, Angora only supports planar stratification along the z direction. The handling of planar layers will be further improved in the future. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

Here is an example:

```
SimulationSpace:
{
    MaterialSlabs:
    (
        {
            material_tag = "material1";
            min_coord = 1e-6;
            max_coord = "max";
        },
        {
            ...
            ...
        }
    );
};
```

In the above example, a material slab composed of `material1` is placed in the grid.

`string material_tag` [Sub-variable of MaterialSlabs]
This variable specifies the material that makes up the slab. It should match a previously-defined tag in a `Materials` definition (see Section 6.4 [Materials], page 21).

`floating-point/string min_coord` [Sub-variable of MaterialSlabs]

`floating-point/string max_coord` [Sub-variable of MaterialSlabs]

`integer/string min_coord_in_cells` [Sub-variable of MaterialSlabs]

`integer/string max_coord_in_cells` [Sub-variable of MaterialSlabs]
These two floating-point variables specify the lower and upper coordinates of the material slab with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. These variables can also be assigned the string values `"min"` or `"max"`; which correspond to the lower and upper boundaries of the simulation grid, respectively. If the coordinates correspond to non-integer cell positions, they are rounded to the

nearest multiple of the spatial step size. However, the tangential components of the electric tensor properties and the normal component of the magnetic tensor properties are suitably interpolated (see [Hwang01], page 89).

If the FDTD grid is terminated by absorbing PML boundaries (see Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15), then the `MaterialSlab` definitions effectively create *infinite planar layers* that extend horizontally toward infinity. When the `"min"` or `"max"` strings are assigned as lower or upper coordinates of the slab, the `MaterialSlab` definition amounts to placing a *half space*. When the `MaterialSlab` variable is used, the incident beams (see Section 6.10 [Incident Beams], page 62) and the scattered far field (see Section 6.8 [Near-Field-to-Far-Field Transformer], page 40) are both calculated as if the material slab horizontally extends toward infinity.

6.5.3 Random Materials

`group RandomMaterials` [Sub-variable of SimulationSpace]

Independent samples from a random distribution of material properties with a specified correlation function can be generated and placed into the simulation grid using the `RandomMaterials` group. It contains sub-variables in the form of lists (see Section 5.2.7 [Lists], page 12) that correspond to specific correlation functions. Currently, only the *Whittle-Matern* family of correlation functions is supported. More correlation functions can be added in the future. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

Although the spatial correlation of the generated random material regions can vary, the joint probability density function of the material region is always a *multivariate normal (Gaussian) function*.

`list WhittleMaternCorrelated` [Sub-variable of RandomMaterials]

The Whittle-Matern family of correlations (see [Rogers09], page 89) is a three-parameter isotropic stochastic model that can represent a wide range of spatial correlations. The Whittle-Matern correlation function $B(r)$ for two points separated in space by a distance of r is given by the formula

$$B(r) = \sigma^2 \frac{2^{5/2-m}(r/l_c)^{m-3/2}}{\Gamma(m-3/2)} K_{m-3/2}(r/l_c)$$

where $K_{m-3/2}(\cdot)$ is the modified Bessel function of the second kind and order (m-3/2).

- m: The shape parameter that determines the overall behavior of the correlation function. As m->infinity, the function approaches a Gaussian distribution. If m=2, the function reduces to a decaying exponential. For m<3/2, the distribution acquires an inverse power law dependence near the origin; approximating a fractal distribution. For more details, see [Rogers09], page 89.
- l_c: (For m>3/2:) The correlation length. (For m<=3/2:) Loosely, the outer length scale where the fractal approximation no longer holds.

- σ: (For m>3/2:) The standard deviation of the distribution at a given point (r=0). (For m<=3/2:) In this range, the correlation function enters the fractal regime with an inverse-power-law dependence at the origin (see [Rogers09], page 89). The meaning of σ becomes more subtle in this regime. It can loosely be associated with the amplitude of the correlation between two points separated by l_c.

The `WhittleMaternCorrelated` list creates regions with random material properties described by the Whittle-Matern correlation function above. Here is an example of its usage:

```
SimulationSpace:
{
    RandomMaterials:
    {
        WhittleMaternCorrelated:
        (
            {
                constitutive_param_type = "rel_permittivity";
                mean = 1.33;
                std_dev = 0.05;
                corr_len = 100e-9;
                m = 2.0;
                shape_tag = "rand_mat_shape";
                random_seed = 0;
            },
            {
                ...
                ...
            }
        );
    };

};
```

`string` [Sub-variable of WhittleMaternCorrelated]
`constitutive_param_type`

The Whittle-Matern correlation function can describe the relative permittivity, relative permeability, electric conductivity (in Siemens/m), or magnetic conductivity (Ohm/m) of the material region. This is specified by assigning `"rel_permittivity"`, `"rel_permeability"`, `"electric_conductivity"`, or `"magnetic_conductivity"` to the `constitutive_param_type` string variable.

The constitutive parameters other than the one specified are *not changed*. As a result, different random constitutive parameter distributions can be superimposed using multiple random material definitions:

```
SimulationSpace:
{
```

```
        RandomMaterials:
        {
            WhittleMaternCorrelated:
            (
                {
                    constitutive_param_type = "rel_permittivity";
                    mean = 1.33;
                    std_dev = 0.05;
                    corr_len = 100e-9;
                    m = 2.0;
                    shape_tag = "rand_mat_shape";
                },
                {
                    constitutive_param_type = "rel_permeability";
                    mean = 1.1;
                    std_dev = 0.05;
                    corr_len = 100e-9;
                    m = 2.0;
                    shape_tag = "rand_mat_shape";
                }
            );
        };

    };
```

Here, a random permittivity distribution and a random permeability distribution are overlaid within the same region in the grid.

`floating-point mean` [Sub-variable of WhittleMaternCorrelated]
(*units: none or S/m*)

A baseline constant value equal to `mean` is added to the constitutive parameter described by the Whittle-Matern correlation function. If `mean=0`, then the generated random distribution will have zero mean. However, this will not necessarily be reflected to the actual constitutive parameter values in the grid; since Angora will automatically clip the constitutive parameters (permittivity, permeability, conductivity, etc.) from below to either unity or zero to avoid instabilities. For this reason, `mean` should be high enough to avoid this clipping as much as possible. As a rule of thumb, `mean` should be 5 to 6 times the standard deviation (`std_dev`) above unity or zero.

`floating-point std_dev` [Sub-variable of WhittleMaternCorrelated]
(*units: none or S/m*)

This variable specifies the σ parameter in the definition of the Whittle-Matern correlation function.

`floating-point corr_len` [Sub-variable of WhittleMaternCorrelated]
(*units: m*)

`floating-point corr_len_in_cells` [Sub-variable of WhittleMaternCorrelated]
This variable specifies the l_c parameter in the definition of the Whittle-Matern correlation function. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`floating-point m` [Sub-variable of WhittleMaternCorrelated]
This variable specifies the m parameter in the definition of the Whittle-Matern correlation function.

`string shape_tag` [Sub-variable of WhittleMaternCorrelated]
This string variable specifies the geometrical shape of the region occupied by the random material. It should match a previously-defined tag in a `Shapes` definition (see Section 6.3 [Shapes], page 18).

`integer/string random_seed` [Sub-variable of WhittleMaternCorrelated]
(*default: determined by system time*)
If you would like to create *exactly the same random distribution* each time the simulation is run, you can assign an integer value to the `random_seed` variable. Otherwise, you **should not** define this variable. This value is used to initialize the random-number generator in Angora. If the same seed is used to initialize the random-number generator, the same sequence of random numbers will be generated each time, resulting in the same random distribution.

If multiple simulation runs are present (see Section 6.14 [Multiple Simulation Runs], page 86), you can create different random samples for each simulation run by assigning the string value `"run_index"` to `random_seed`. This will initialize the intenal random-number generator with the run index (ranging from `0` to `number_of_runs-1`) of each run. This way, a different random distribution will be obtained in each simulation run; but a distribution for a given simulation run will be *fixed* in subsequent executions of Angora.

In Figure 6.2, a 2D slice of an example zero-mean sample distribution generated by `WhittleMaternCorrelated` is shown in grayscale.

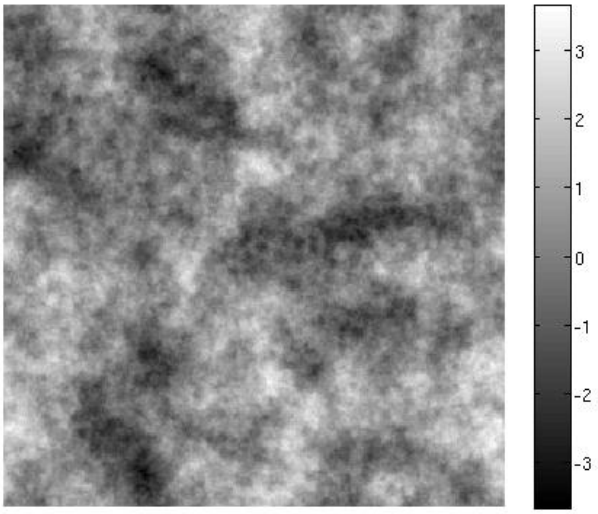

Figure 6.2: A 2D slice of an example zero-mean sample distribution. This distribution can be assigned to different constitutive parameters of a material.

6.5.4 File Input

`list MaterialsFromFiles` [Sub-variable of SimulationSpace]

Material information within rectangular regions of the FDTD simulation grid can be read from files using a `MaterialsFromFiles` list. This feature of Angora is still under development. The user interface for this feature may change in the future, or be superseded by another, more general interface. Currently, only a single constitutive parameter can be read from a file; and dispersive or anisotropic materials are not supported. These issues will be handled more comprehensively in a future version. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

The material file should be in a simple custom binary format that Angora can recognize. The order and type of each variable in the file is explained below:

- '`x-extent`': The extent of the array in the x direction in grid cells (integer, 4 bytes)
- '`y-extent`': The extent of the array in the y direction in grid cells (integer, 4 bytes)
- '`z-extent`': The extent of the array in the z direction in grid cells (integer, 4 bytes)
- A floating-point array of length (`x-extent`) x (`y-extent`) x (`z-extent`). Each value in this array is either of type `double` (8 bytes) or `float` (4 bytes), depending on the `datatype` variable (see [datatype], page 33). The floating-point array should be laid out in the file in *column-major* order, meaning that the x dimension is looped over first, then the y dimension, and finally the z dimension. This ordering is illustrated in Figure 6.3. The elements of the 2x2x2 array are

numbered from 0 to 7. These elements should be laid out in the binary file in the same order:

```
...... 0  1  2  3  4  5  6  7  ......
```

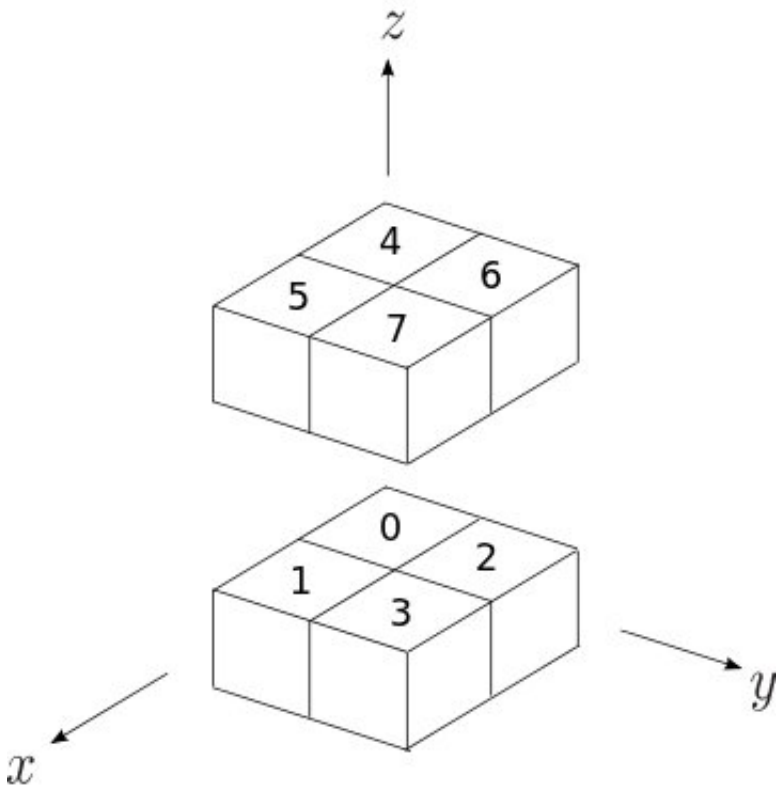

Figure 6.3: The illustration of the column-major ordering of a three-dimensional array. The values indicated by numbers should be laid out in the file in the same order.

Here is an example usage of `MaterialsFromFiles`:

```
SimulationSpace:
{
    MaterialsFromFiles:
    (
        {
            file_name = "path_to_file/materialfile";
            append_run_index_to_name = true;
            file_extension = "mat";
            constitutive_param_type = "rel_permittivity";
            anchor = "center";
            coord_x = 0;
            coord_y = 0;
            coord_z = 0;
```

```
                datatype = "double";
                max_new_materials = 1000;
            },
            {
                ...
                ...
            }
        );
    };
```

`string file_name` [Sub-variable of MaterialsFromFiles]
This string specifies the name of the binary file from which the material information will be read. Path information can be prepended to the file name, as shown in the example above. This path is interpreted as being relative to `input_dir` (see Section 6.12 [Paths], page 85), unless it is preceded by a slash '`/`'.

`string file_extension` (*default:* `""`) [Sub-variable of MaterialsFromFiles]
This is the extension of the material file to be read. In the above example, the file to be read is '`path_to_file/materialfile.mat`'.

`boolean append_run_index_to_name` [Sub-variable of MaterialsFromFiles]
This boolean flag becomes useful if there are multiple simulation runs (see Section 6.14 [Multiple Simulation Runs], page 86), and a different file needs to be read in each run. This can be accomplished by appending the run index (which ranges from `0` to `number_of_runs-1`) to the file name specified by `file_name`. For example, if there are 3 simulation runs (`number_of_runs` is 3) the above assignment will tell Angora to read the file '`path_to_file/materialfile0.mat`' in the first run, '`path_to_file/materialfile1.mat`' in the second, and '`path_to_file/materialfile2.mat`' in the third.

This variable is required for all simulations (hence no default value) to help the user prevent easy mistakes such as reading the same file for all simulation runs unintentionally, reading '`path_to_file/materialfile0.mat`' instead of '`path_to_file/materialfile.mat`', etc.

`string constitutive_param_type` [Sub-variable of MaterialsFromFiles]
The values read from the input file can be assigned to one of the following constitutive parameters: relative permittivity, relative permeability, electric conductivity, or magnetic conductivity. This is determined by assigning `"rel_permittivity"`, `"rel_permeability"`, `"electric_conductivity"`, or `"magnetic_conductivity"` to the `constitutive_param_type` string variable. Electric conductivity is assumed to be in Siemens/m, and magnetic conductivity is assumed to be in Ohm/m.

The constitutive parameters other than the one specified are *not changed*. As a result, different constitutive parameter distributions can be superimposed using multiple file-input definitions:

```
SimulationSpace:
{
    MaterialsFromFiles:
    (
        {
            file_name = "permittivity_file";
            append_run_index_to_name = true;
            constitutive_param_type = "rel_permittivity";
            coord_x = 0;
            coord_y = 0;
            coord_z = 0;
            datatype = "double";
        },
        {
            file_name = "conductivity_file";
            append_run_index_to_name = true;
            constitutive_param_type = "electric_conductivity";
            coord_x = 0;
            coord_y = 0;
            coord_z = 0;
            datatype = "double";
        }
    );
};
```

Here, the contents of the files '`permittivity_file`' and '`conductivity_file`' are interpreted as the relative permittivity and electric conductivity of the same region, respectively.

`string anchor` (*default:* `"center"`) [Sub-variable of MaterialsFromFiles]

This string defines an anchor point inside the rectangular-box-shaped region that is to be read from this file. This anchor is then assigned a coordinate in the FDTD grid, determining the position of the rectangular box in the grid. Valid values for `anchor` are:

- `"center"`: center of the box
- `"BLL"`: back-left-lower corner of the box
- `"BLU"`: back-left-upper corner of the box
- `"BRL"`: back-right-lower corner of the box
- `"BRU"`: back-right-upper corner of the box
- `"FLL"`: front-left-lower corner of the box
- `"FLU"`: front-left-upper corner of the box
- `"FRL"`: front-right-lower corner of the box
- `"FRU"`: front-right-upper corner of the box

Here, as usual, "back"/"front" refers to the x coordinate, "left"/"right" refers to the y coordinate, and "lower"/"upper" refers to the z coordinate.

`floating-point coord_x` (*units: m*) [Sub-variable of MaterialsFromFiles]

`floating-point coord_y` (*units: m*) [Sub-variable of MaterialsFromFiles]

`floating-point coord_z` (*units: m*) [Sub-variable of MaterialsFromFiles]

`integer coord_x_in_cells` [Sub-variable of MaterialsFromFiles]

`integer coord_y_in_cells` [Sub-variable of MaterialsFromFiles]

`integer coord_z_in_cells` [Sub-variable of MaterialsFromFiles]

These values determine the Cartesian x,y, and z coordinates of the anchor point (see above) assigned to the rectangular region to be read from the file. The coordinates are measured with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinates correspond to non-integer cell positions, the closest integer positions are chosen.

`string datatype` [Sub-variable of MaterialsFromFiles]

The datatype for the values read from the file is determined by this variable. It should be either `"double"` (8 bytes) or `"float"` (4 bytes).

`integer max_new_materials` (*default:* `1000`) [Sub-variable of MaterialsFromFiles]

Internally, Angora uses *material indexing* to reduce memory use for material arrays. Every constitutive parameter in the grid can only take a distinct set of values, represented by a variable of type `unsigned short` (2 bytes) that can range from `0` to `65,535`. Instead of storing a floating-point value (which is usually 4 or 8 bytes) for a permittivity value at a point, Angora stores an *index* that represents the permittivity at that point. The same applies to other constitutive parameters (relative permeability, electric conductivity, etc.)

Each time a material region is read into the FDTD grid using `MaterialsFromFiles`, a fixed number of new constitutive parameter values are defined between the minimum and maximum values found in the file. Because of this discretization, some loss of information is inevitable. The number of new materials is determined by the variable `max_new_materials`; which is by equal to 1000 default. With the default value, the upper limit for the number of materials will be reached after about 65 material regions are inserted into the grid. If you wish to insert more material regions, and the dynamic ranges of constitutive parameters in your material files are not large, you can decrease `max_new_materials`. Alternatively, you may consider combining multiple material regions into a single region.

6.5.5 Ground Planes

`list GroundPlanes` [Sub-variable of SimulationSpace]

Infinitely thin perfect-electric-conductor (PEC) sheets can be placed in the grid using a `GroundPlanes` list. Currently, only z-oriented (parallel to the xy plane) sheets at integer (full-cell) positions are supported.

```
SimulationSpace:
{
    GroundPlanes:
    (
        {
            coord = 0;
        },
        {
            ...
            ...
        }
    );
};
```

`floating-point coord` (*units: m*) [Sub-variable of GroundPlanes]

`integer coord_in_cells` [Sub-variable of GroundPlanes]

This variable specifies the z-coordinate of the ground plane with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

The `GroundPlanes` variable also updates the layering (stratification) information in the grid, much like `MaterialSlabs` (see Section 6.5.2 [Planar Layers], page 24).

6.6 Waveforms

`group Waveforms` [Global variable]

In Angora, a *time waveform* is defined as a self-contained structure that can be used by other structures; such as a Hertzian dipole source or a plane-wave injector. The library of available time waveforms will be expanded in the future. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

An example usage of `Waveforms`:

```
Waveforms:
{
    GaussianWaveforms:
    (
        {
            ...
        }
    );

    DifferentiatedGaussianWaveforms:
    (
        {
            ...
        }
```

```
        );
        ...
        ...
    };
```

6.6.1 Gaussian Waveforms

`list GaussianWaveforms` [Sub-variable of Waveforms]

This variable is used to define Gaussian time waveforms given by the formula

$$f(t) = A \ \exp\left(\frac{-(t - n_\tau \tau)^2}{2\tau^2}\right)$$

The peak, 10%-amplitude (-20 dB power), and 1%-amplitude (-40 dB power) frequencies in the spectrum of the Gaussian are $\omega = 0$, $\omega = 2.15/\tau$, $\omega = 3.035/\tau$, respectively.

Gaussian waveforms are defined as follows:

```
Waveforms:
{
    GaussianWaveforms:
    (
        {
            waveform_tag = "my_waveform";
            amplitude = 1.0;
            tau = 2.1291e-15;
            delay = 3;
        },
        {
            ...
            ...
        }
    );
};
```

`string waveform_tag` [Sub-variable of GaussianWaveforms]

This is the string tag by which the waveform will later be referred to by another structure that requires a time waveform in its definition.

`floating-point amplitude` (*default:* `1.0`) [Sub-variable of GaussianWaveforms]

This specifies the variable A in the above equation defining the Gaussian waveform.

`floating-point tau` (*units: sec*) [Sub-variable of GaussianWaveforms]

This specifies the variable τ in the above equation defining the Gaussian waveform.

`floating-point delay` (*default:* `0.0`) [Sub-variable of GaussianWaveforms]

This specifies the variable n_τ in the above equation defining the Gaussian waveform.

6.6.2 Differentiated-Gaussian Waveforms

`list DifferentiatedGaussianWaveforms` [Sub-variable of Waveforms]

This variable is used to define differentiated Gaussian time waveforms, given by the formula

$$f(t) = A\,\frac{d^n}{dt^n}\left[\exp\left(\frac{-(t-n_\tau\tau)^2}{2\tau^2}\right)\right] = A\,(\frac{-1}{\tau\sqrt{2}})^n H_n\left(\frac{t-n_\tau\tau}{\tau\sqrt{2}}\right)\exp\left(\frac{-(t-n_\tau\tau)^2}{2\tau^2}\right)$$

where $H_n(x)$ are the (physicists') Hermite polynomials.

The peak frequency in the spectrum of the differentiated-Gaussian is $\omega = 1/\tau$, the 10%-amplitude (-20 dB power) frequencies are $\omega = 0.06/\tau$ and $\omega = 2.76/\tau$; and the 1%-amplitude (-40 dB power) frequencies are $\omega = 0.006/\tau$ and $\omega = 3.57/\tau$.

Differentiated Gaussian waveforms are defined as follows:

```
Waveforms:
{
    DifferentiatedGaussianWaveforms:
    (
        {
            waveform_tag = "my_waveform";
            amplitude = 1.0;
            tau = 2.1291e-15;
            delay = 3;
            n_diff = 3;
        },
        {
            ...
            ...
        }
    );
};
```

`string waveform_tag` [Sub-variable of DifferentiatedGaussianWaveforms]

This is the string tag by which the waveform will later be referred to by another structure that requires a time waveform in its definition.

`floating-point` [Sub-variable of DifferentiatedGaussianWaveforms]
`amplitude` (*default:* `1.0`)

This specifies the variable A in the above equation defining the differentiated Gaussian waveform.

`floating-point tau` [Sub-variable of DifferentiatedGaussianWaveforms]
(*units: sec*)

This specifies the variable τ in the above equation defining the differentiated Gaussian waveform.

`floating-point delay` [Sub-variable of DifferentiatedGaussianWaveforms]
(*default:* `0.0`)

This specifies the variable n_τ in the above equation defining the differentiated Gaussian waveform.

`integer n_diff` [Sub-variable of DifferentiatedGaussianWaveforms]
This specifies the order of differentiation n in the above equation defining the differentiated Gaussian waveform.

6.6.3 Modulated-Gaussian Waveforms

`list ModulatedGaussianWaveforms` [Sub-variable of Waveforms]
This variable is used to define sinusoidally-modulated Gaussian time waveforms, given by the formula

$$f(t) = A\ g(2\pi f_0(t - n_\tau\tau) + \phi) \exp\left(\frac{-(t - n_\tau\tau)^2}{2\tau^2}\right)$$

where the function $g(t)$ is a sinusoidal function, being either $\sin(t)$ or $\cos(t)$.

The peak frequency in the spectrum of the modulated-Gaussian is $\omega = \omega_0 = 2\pi f_0$, the 10%-amplitude (-20 dB power) frequencies are $\omega = \omega_0 \pm 2.15/\tau$; and the 1%-amplitude (-40 dB power) frequencies are $\omega = \omega_0 \pm 3.035/\tau$. A MATLAB script named '`mod_gaussian_wf.m`' is distributed as part of the Angora package, which calculates the center frequency f_0 and the time constant τ of a modulated-Gaussian waveform that has the desired lower and upper cutoff wavelengths, and the desired amount of attenuation at these wavelengths. It also outputs the -40 dB wavelength and the suggested maximum spatial time step in the simulation (which is the -40 dB wavelength divided by 15). This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here).

Modulated Gaussian waveforms are defined as follows:

```
Waveforms:
{
    ModulatedGaussianWaveforms:
    (
        {
            waveform_tag = "my_waveform";
            modulation_type = "sine";
            amplitude = 1.0;
            tau = 2.1291e-15;
            f_0 = 5.8929e14;
            delay = 3;
            phase = 90;
            differentiated = false;
        },
        {
            ...
            ...
        }
    );
};
```

`string waveform_tag` [Sub-variable of ModulatedGaussianWaveforms]
This is the string tag by which the waveform will later be referred to by another structure that requires a time waveform in its definition.

`string modulation_type` [Sub-variable of ModulatedGaussianWaveforms]
If assigned `"sine"`, the modulation function $g(t)$ in the above equation becomes a sine. If assigned `"cosine"`, it becomes a cosine.

`floating-point amplitude` [Sub-variable of ModulatedGaussianWaveforms]
(*default:* `1.0`)
This specifies the variable A in the above equation defining the modulated Gaussian waveform.

`floating-point tau` (*units: sec*) [Sub-variable of ModulatedGaussianWaveforms]
This specifies the variable τ in the above equation defining the modulated Gaussian waveform.

`floating-point f_0` (*units: Hz*) [Sub-variable of ModulatedGaussianWaveforms]
This specifies the modulation frequency f_0 in the above equation defining the modulated Gaussian waveform.

`floating-point delay` [Sub-variable of ModulatedGaussianWaveforms]
(*default:* `0.0`)
This specifies the variable n_τ in the above equation defining the modulated Gaussian waveform.

`floating-point phase` [Sub-variable of ModulatedGaussianWaveforms]
(*units: degrees, default:* `0.0`)
This specifies the extra phase ϕ in the above equation defining the modulated Gaussian waveform. This phase should be specified in *degrees*, which is then converted internally to radians, which are the actual units of ϕ .

`boolean differentiated` [Sub-variable of ModulatedGaussianWaveforms]
(*default:* `false`)
If set to `true`, the waveform in the above equation is differentiated once with respect to time.

6.7 Point Sources

`list PointSources` [Global variable]
"Infinitesimal" electric dipole sources (also called *Hertzian* dipoles) can be simulated in Angora using the `PointSources` list.

A Hertzian dipole at position (x_0, y_0, z_0) is characterized by the following current distribution in space:

$$J(x, y, z; t) = \mathbf{a}\, j_0(t)\, \delta(x - x_0)\delta(y - y_0)\delta(z - z_0)$$

where $\delta(x)$ is the Dirac delta function. The vector **a** determines the orientation of the dipole, which can be along the x, y, or z directions. The prefactor $j_0(t)$ is called the *current moment* of the dipole, with the units (Ampere*m).

Here is an example usage of `PointSources`:

```
PointSources:
(
    {
        coord_x = 0;
        coord_y = 0;
        coord_z = 0;
        source_orientation = "y_directed";
        waveform_tag = "moment_waveform";
        j_0 = 1.0;
    },
    {
        ...
        ...
    }
);
```

`floating-point coord_x` (*units: m*) [Sub-variable of PointSources]

`floating-point coord_y` (*units: m*) [Sub-variable of PointSources]

`floating-point coord_z` (*units: m*) [Sub-variable of PointSources]

`integer coord_x_in_cells` [Sub-variable of PointSources]

`integer coord_y_in_cells` [Sub-variable of PointSources]

`integer coord_z_in_cells` [Sub-variable of PointSources]

These variables specify the Cartesian x, y, and z coordinates of the Hertzian dipole with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinates correspond to non-integer cell positions, the closest integer positions are chosen.

`string source_orientation` [Sub-variable of PointSources]

This string specifies the spatial orientation of the Hertzian dipole. It should be `"x_directed"`, `"y_directed"`, or `"z_directed"`.

`string waveform_tag` [Sub-variable of PointSources]

This string variable specifies the waveform of the current moment $j_0(t)$. The waveform is interpreted in (Ampere*m) units. This should match a previously-defined tag in a `Waveforms` definition (see Section 6.6 [Waveforms], page 34).

`floating-point j_0` (*units: Ampere/m, default:* `1.0`) [Sub-variable of PointSources]

This is an extra prefactor applied to the current moment waveform $j_0(t)$.

6.8 Near-Field-to-Far-Field Transformer

In many electromagnetic problems, it is of interest to calculate the radiated (or far-zone) field scattered from (or radiated by) the structures inside the grid. The radiated field is defined as the asymptotic form of the electric field at large distances, which decays as $1/r$ and propagates locally like a plane wave. Although the radial dependence is trivial in the far field, the angular dependence is highly variable. In finite numerical solution methods such as FDTD and FEM, it is only the near-field that is available in the computation grid. It is hugely impractical to extend the computation grid to large distances where the field assumes an asymptotic form. Luckily, certain theorems of electromagnetics (Huygens' principle, equivalence theorem, etc.) allow the calculation of the far field using this near field information. This procedure is called a **near-field-to-far-field transform** (NFFFT). There are two main types of NFFFTs. In the first type, the far-field waveforms are calculated directly in time domain. In the second, the frequency (or phasor) components in the Fourier decomposition of the far-field waveforms are calculated at a number of frequencies. Angora features both time-domain and phasor domain NFFFTs.

6.8.1 Time-Domain Near-Field-to-Far-Field-Transformer

In the *time-domain* NFFFT, the far-field waveforms (normalized by the distance r, and advanced in time by r/c) are calculated directly using time-domain Green's functions for the particular space. Currently, the time-domain NFFFT supports up to *three* lossless infinite planar material layers with only permittivity variations.

The time-domain NFFFT should be used when the far-field waveforms are to be computed over only a few observation directions. The additional computational burden per observation direction is much larger than that of the phasor-domain NFFFT. The format used for the time-domain far-field output is **HDF5** (Hierarchical Data Format) (`http://www.hdfgroup.org/HDF5/`). The HDF5 format was chosen for its standard interface, and the availability of free software tools for inspecting and modifying HDF5 output. The HDF5 output created by the time-domain NFFFT is explained in more detail in Section 6.8.1.1 [HDF5 Content of Time-Domain NFFFT Output], page 44.

The radiated electric field can be expressed in the form

$$\bar{E}^r(r,\theta,\phi;t) = \bar{E}(\theta,\phi;t-r/c)/r \ .$$

The time-domain NFFFT only calculates the *angle and time-dependent* part of the above expression, namely, $\bar{E}(\theta,\phi;t)$.

`string td_nffft_output_dir` (*default:* `"nffft/td"`) [Global variable]

This determines the subdirectory in which all the time-domain-NFFFT output will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `output_dir` (see Section 6.12 [Paths], page 85).

```
td_nffft_output_dir = "nffft/td";
TimeDomainNFFFT:
{
        ...
        ...
};
```

`list TimeDomainNFFFT` [Global variable]

Time-domain NFFFTs are defined inside a `TimeDomainNFFFT` list, each within its own group:

```
TimeDomainNFFFT:
(
    {
        theta = 36;
        phi = 57;
        write_hertzian_dipole_far_field = false;
        nffft_back_margin_x_in_cells = 3;
        nffft_front_margin_x_in_cells = 3;
        nffft_left_margin_y_in_cells = 3;
        nffft_right_margin_y_in_cells = 3;
        nffft_lower_margin_z_in_cells = 3;
        nffft_upper_margin_z_in_cells = 3;
        far_field_origin_x = 0;
        far_field_origin_y = 0;
        far_field_origin_z = 0;
        far_field_dir = "my_dir";
        far_field_file_name = "FarField_td";
        far_field_file_extension = "hd5";
    },
    {
        ...
        ...
    }
);
```

`floating-point theta` (*units: degrees*) [Sub-variable of TimeDomainNFFFT]

`floating-point phi` (*units: degrees*) [Sub-variable of TimeDomainNFFFT]

The direction at which the time-domain far field will be calculated is expressed in terms of the traditional spherical-coordinate variables (θ, ϕ). The first of these angles is the *zenith angle*, while the second is the *azimuth angle*. In Figure 6.4, the definitions of these angles are shown schematically. The `theta` variable specifies the zenith angle in degrees. Although this angle is traditionally defined between 0 and 180deg, `theta` can be assigned any negative or positive value. The `phi` variable specifies the azimuth angle in degrees. Although this angle is traditionally defined between 0 and 360deg, `phi` can be assigned any negative or positive value.

`boolean write_hertzian_dipole_far_field` (*default:* `false`) [Sub-variable of TimeDomainNFFFT]

If set to `true`, the theoretical far field waveforms due to the Hertzian point sources (see Section 6.7 [Point Sources], page 38) in the simulation grid are also written into the output file. Any planar stratification up to three lossless layers with permittivity variations is accounted for in the calculation of the

theoretical far field, but the scattering from any other structure inside the grid is ignored. As such, this feature can be (and has been) used to test the time-domain NFFFT.

`floating-point` [Sub-variable of TimeDomainNFFFT]
`nffft_back_margin_x` (*units:m*)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`nffft_front_margin_x` (*units:m*)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`nffft_left_margin_y` (*units:m*)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`nffft_right_margin_y` (*units:m*)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`nffft_lower_margin_z` (*units:m*)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`nffft_upper_margin_z` (*units:m*)

`integer` [Sub-variable of TimeDomainNFFFT]
`nffft_back_margin_x_in_cells` (*default:* `3`)

`integer` [Sub-variable of TimeDomainNFFFT]
`nffft_front_margin_x_in_cells` (*default:* `3`)

`integer` [Sub-variable of TimeDomainNFFFT]
`nffft_left_margin_y_in_cells` (*default:* `3`)

`integer` [Sub-variable of TimeDomainNFFFT]
`nffft_right_margin_y_in_cells` (*default:* `3`)

`integer` [Sub-variable of TimeDomainNFFFT]
`nffft_lower_margin_z_in_cells` (*default:* `3`)

`integer` [Sub-variable of TimeDomainNFFFT]
`nffft_upper_margin_z_in_cells` (*default:* `3`)

The near field is collected over the surface of a rectangular prism in the grid, called the *NFFFT surface*. This surface should enclose all the scattering and/or radiating structures in the grid, as well as the total-field/scattered-field surface (see Section 6.10 [Incident Beams], page 62). By default, this rectangular box is placed 3 grid cells away from the PML boundary (see Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15). You can specify different margins to reduce the computational burden associated with the NFFFT. This burden is directly proportional to the surface area of the box. The margins can be specified in meters or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If given in meters, the margins are rounded to the nearest multiple of the spatial step size.

`floating-point` [Sub-variable of TimeDomainNFFFT]
`far_field_origin_x` (*units: m, default:* `0`)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`far_field_origin_y` (*units: m, default:* `0`)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`far_field_origin_z` (*units: m, default:* `0`)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`far_field_origin_x_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`far_field_origin_y_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of TimeDomainNFFFT]
`far_field_origin_z_in_cells` (*default:* `0`)

These variables set the coordinates of the point relative to which the far field will be calculated. The distance r in [eq:far_field_angle_dependence_time_domain], page 40 is with respect to this point. The coordinates of the far-field origin are with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`string far_field_dir` (*default:* `""`) [Sub-variable of TimeDomainNFFFT]
This determines the subdirectory in which this individual far-field file will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `td_nffft_output_dir` (see [td_nffft_output_dir], page 40). By default, no subdirectory is created inside `td_nffft_output_dir`.

`string far_field_file_name` [Sub-variable of TimeDomainNFFFT]
(*default:* `"FarField_td"`)
This determines the base string in the full name of the far-field file. Other information is appended to the name of the file to differentiate individual far-field files (see the example below).

`string far_field_file_extension` [Sub-variable of TimeDomainNFFFT]
(*default:* `"hd5"`)
This is the extension of the far-field file name. If assigned the value `""`, no extension is added to the file. The HDF5 extension `"hd5"` is applied by default.

Here is an example far-field file name:

```
FarField_td_0_1.hd5
```

The base string in the name of the file (`"FarField_td"`) is specified by the `far_field_file_name` variable. The two integers that follow are the run index (see Section 6.14 [Multiple Simulation Runs], page 86) and the index of the time-domain NFFFT inside the `TimeDomainNFFFT` list. For example, if there are two groups (two NFFFTs) in the `TimeDomainNFFFT` list, the first one will write into

```
FarField_td_0_0.hd5
```

while the second will write into

`FarField_td_0_1.hd5`

If there are two simulation runs (i.e., `number_of_runs` is equal to 2 – see Section 6.14 [Multiple Simulation Runs], page 86), then the files created in the second run will have `1` instead of `0` as the first integer in the above file names. Finally, the extension (`"hd5"`) of the line files is determined by the variable `far_field_file_extension`.

6.8.1.1 HDF5 Content of Time-Domain NFFFT Output

The HDF5 file created as the output of the time-domain NFFFT can be viewed and modified using freely-available tools. One of these tools is HDFView, provided by the HDF Group. MATLAB also has built-in functions and tools that handle HDF5 files. For reference, a MATLAB script named '`hdf5_read.m`' is distributed as part of the Angora package, which reads an HDF5 dataset from an HDF5 file into a MATLAB array. This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here). For example, if you want to read the dataset named `theta` from the file '`my_file.hd5`', use

```
>> theta = hdf5_read('my_file.hd5','theta');
```

In MATLAB R2011a and later, there is a high-level built-in function `h5read` that could be used for the same purpose.

The HDF5 datasets in the far-field file are the following:

- '`angora_version`': Integer array of length 3 with the major version, minor version, and revision numbers of the Angora package used to create the file.
- '`theta`': A floating-point value for the spherical zenith angle at which the far field is calculated (in radians).
- '`phi`': A floating-point value for the spherical azimuth angle at which the far field is calculated (in radians).
- '`time_step`': A floating-point value specifying the temporal step in the simulation (in sec).
- '`initial_time_value`': A floating-point value specifying the time value corresponding to the beginning of the simulation (in sec). This is usually a negative value, since time waveforms frequently begin before t=0.
- Floating-point arrays with the waveforms of different components of the vector radiated electric field. See Figure 6.4 for a graphical illustration of the unit vectors in spherical coordinates. Note that only the angle and time-dependent part of the radiated electric field is calculated (see [eq:far_field_angle_dependence_time_domain], page 40). Because the (1/r) dependence has been factored out, the units are in Volts.

  '`E_theta`': 1-D array with the theta component of the radiated electric field.

  - '`E_phi`': 1-D array with the phi component of the radiated electric field.
- If the theoretical far field due to Hertzian point sources is also calculated (namely, `write_hertzian_dipole_far_field` is `true`):

  '`E_theta_th`': 1-D array with the *theoretical* theta component of the radiated electric field created by the Hertzian dipoles in the simulation grid.

- '`E_phi_th`': 1-D array with the *theoretical* phi component of the radiated electric field created by the Hertzian dipoles in the simulation grid.

6.8.2 Phasor-Domain Near-Field-to-Far-Field-Transformer

The *phasor-domain* NFFFT calculates the amplitude of the far field at individual frequencies using Fourier decomposition. This NFFFT supports free space as well as **infinite planar layered media** with arbitrary permittivity, permeability and conductivity profiles (see [Capoglu12], page 89). Infinite planar layers are created using the `MaterialSlabs` variable (see Section 6.5.2 [Planar Layers], page 24).

The phasor-domain NFFFT in Angora calculates far-field values over a two-dimensional array of observation directions and a range of wavelengths; resulting in a three-dimensional array. The spacing of the wavelengths and the arrangement of observation directions is highly configurable. The format used for the phasor-domain far-field output is **HDF5** (Hierarchical Data Format) (`http://www.hdfgroup.org/HDF5/`). The HDF5 format was chosen for its standard interface, and the availability of free software tools for inspecting and modifying HDF5 output. The HDF5 output created by the phasor-domain NFFFT is explained in more detail in Section 6.8.2.1 [HDF5 Content of Phasor-Domain NFFFT Output], page 52.

Angora uses the engineering convention $\exp(j\omega t)$ for time-harmonic quantities. The time-domain data on the surface of a rectangular prism in the grid, called the *NFFFT surface*, is decomposed into its phasor components by a numerical approximation to the temporal Fourier transform $F(\omega) = \frac{1}{2\pi}\int f(t)\exp(-j\omega t)dt$. Because of the $\frac{1}{2\pi}$ term, the phasor quantities $F(\omega)$ correspond to the true *Fourier components* of the time-domain quantities on the NFFFT surface. These quantities are then inserted into phasor-domain electromagnetic theorems linking the near field to the far field. The complex phasor output data therefore also corresponds to the Fourier components of the far-field temporal waveforms.

In the phasor domain, the radiated electric field can be expressed in the form

$$\bar{E}^r(r,\theta,\phi) = \bar{E}(\theta,\phi)\exp(-jkr)/r \ .$$

The phasor-domain NFFFT only calculates the *angle-dependent* part of the above expression, namely, $\bar{E}(\theta,\phi)$. In Figure 6.4, the angles and unit vectors are shown for spherical coordinates.

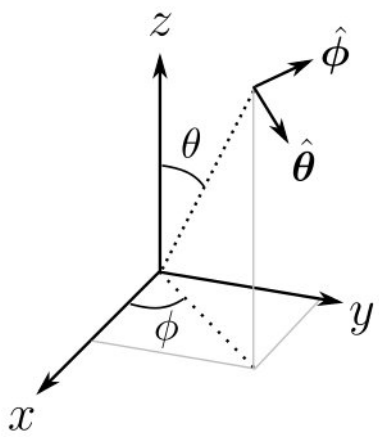

Figure 6.4: The angles and unit vectors for spherical coordinates.

`string pd_nffft_output_dir` (*default:* `"nffft/pd"`) [Global variable]
This determines the subdirectory in which all the phasor-domain-NFFFT output will be placed. Unless it has a slash '/' up front; this path is interpreted as being relative to `output_dir` (see Section 6.12 [Paths], page 85).

```
pd_nffft_output_dir = "nffft/pd";
PhasorDomainNFFFT:
{
        ...
        ...
};
```

`list PhasorDomainNFFFT` [Global variable]
Phasor-domain NFFFTs are defined inside a `PhasorDomainNFFFT` list, each within its own group:

```
PhasorDomainNFFFT:
(
    {
        num_of_lambdas = 10;
        lambda_min = 400e-9;
        lambda_max = 700e-9;
        lambda_spacing_type = "k-linear";
        do_not_include_first_lambda = false;
        do_not_include_last_lambda = false;
        direction_spec = "theta-phi";
        num_of_dirs_1 = 9;
        dir1_min=0.0;
        dir1_max=90.0;
        num_of_dirs_2 = 361;
        dir2_min=0.0;
        dir2_max=360.0;
        limit_to_s = 1.0;
```

```
            write_hertzian_dipole_far_field = false;
            nffft_back_margin_x_in_cells = 3;
            nffft_front_margin_x_in_cells = 3;
            nffft_left_margin_y_in_cells = 3;
            nffft_right_margin_y_in_cells = 3;
            nffft_lower_margin_z_in_cells = 3;
            nffft_upper_margin_z_in_cells = 3;
            far_field_origin_x = 0.0;
            far_field_origin_y = 0.0;
            far_field_origin_z = 0.0;
            far_field_dir = "my_dir";
            far_field_file_name = "FarField_pd";
            far_field_file_extension = "hd5";
        },
        {
            ...
            ...
        }
    );
```

`integer num_of_lambdas` [Sub-variable of PhasorDomainNFFFT]
: This specifies the number of wavelengths (in vacuum) at which the far field will be calculated.

`floating-point lambda_min` (*units:* *m*) [Sub-variable of PhasorDomainNFFFT]

`floating-point lambda_min_in_cells` [Sub-variable of PhasorDomainNFFFT]
: This value sets the lower limit of the wavelength range (in vacuum) over which the far field is calculated. The far field may or may not be calculated at the wavelength `lambda_min`, depending on the variable `do_not_include_first_lambda`. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`floating-point lambda_max` (*units:* *m*) [Sub-variable of PhasorDomainNFFFT]

`floating-point lambda_max_in_cells` [Sub-variable of PhasorDomainNFFFT]
: This value sets the upper limit of the wavelength range (in vacuum) over which the far field is calculated. The far field may or may not be calculated at the wavelength `lambda_max`, depending on the variable `do_not_include_last_lambda`. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`string lambda_spacing_type` [Sub-variable of PhasorDomainNFFFT]
: This string specifies how the wavelengths will be spaced between the two end points determined by `lambda_min`, `lambda_max`, `do_not_include_first_lambda`, and `do_not_include_last_lambda`.

- `"lambda-linear"`: The wavelengths are spaced lineary between the two end points.
- `"k-linear"`: The wavenumbers $k = 2\pi/\lambda$ are spaced linearly between the two end points. Since the wavenumber is also equal to $k = \omega/c$, where ω is the radian frequency, this causes the frequencies to be spaced linearly as well.
- `"log"`: The logarithms of the wavelengths (therefore the logarithms of the wavenumbers) are spaced linearly between the two end points.

`boolean` [Sub-variable of PhasorDomainNFFFT]
`do_not_include_first_lambda` (*default:* `false`)

`boolean` [Sub-variable of PhasorDomainNFFFT]
`do_not_include_last_lambda` (*default:* `false`)

Let's assume that `lambda_spacing_type` is `lambda-linear`. For `k-linear` and `log`, replace `lambda_min` in the following by $2\pi/$ `lambda_min` and log(`lambda_min`), respectively. The same applies to `lambda_max`.

- If `do_not_include_first_lambda=false` and `do_not_include_last_lambda=false`: The interval between `lambda_min` and `lambda_max` is divided into `(num_of_lambdas-1)` equal intervals. A total of `num_of_lambdas` wavelengths are placed linearly at the boundaries between the intervals, including both endpoints `lambda_min` and `lambda_max`.
- If `do_not_include_first_lambda=true` and `do_not_include_last_lambda=false`: The interval between `lambda_min` and `lambda_max` is divided into `num_of_lambdas` equal intervals. A total of `num_of_lambdas` wavelengths are placed linearly at the boundaries between the intervals, excluding the endpoint `lambda_min`.
- If `do_not_include_first_lambda=false` and `do_not_include_last_lambda=true`: The interval between `lambda_min` and `lambda_max` is divided into `num_of_lambdas` equal intervals. A total of `num_of_lambdas` wavelengths are placed linearly at the boundaries between the intervals, excluding the endpoint `lambda_max`.
- If `do_not_include_first_lambda=true` and `do_not_include_last_lambda=true`: The interval between `lambda_min` and `lambda_max` is divided into `num_of_lambdas` equal intervals. A total of `num_of_lambdas` wavelengths are placed at the midpoints of each interval.

`string direction_spec` [Sub-variable of PhasorDomainNFFFT]

This string specifies how the observation directions are arranged in a two-dimensional array.

- `"theta-phi"`: The first dimension is the spherical polar angle θ , defined as the angle between the observation direction and the z-axis. The second dimension is the spherical azimuth angle ϕ , defined as the angle between the x-axis and the projection of the observation direction onto the xy-plane. See Figure 6.4 for a graphical illustration. These angles are spaced linearly between their respective endpoints.

- `"dircosx-dircosy-upper"` or `"dircosx-dircosy-lower"`: The first dimension is the x-direction-cosine $s_x = \sin\theta\cos\phi$ while the second dimension is the y-direction-cosine $s_y = \sin\theta\sin\phi$. These direction cosines are spaced linearly between their respective endpoints. The suffix `"-upper"` or `"-lower"` determines whether the observation direction is in the upper half space (+z direction) or the lower half space (-z direction).

`integer num_of_dirs_1` [Sub-variable of PhasorDomainNFFFT]

This is the number of observation directions over the first dimension of the two-dimensional observation-direction array. If `direction_spec` is `"theta-phi"`, this is the number of θ values; otherwise, the number of x-direction-cosines $s_x = \sin\theta\cos\phi$.

`floating-point dir1_min` [Sub-variable of PhasorDomainNFFFT]

`floating-point dir1_max` [Sub-variable of PhasorDomainNFFFT]

These are the minimum/maximum values of either the θ angle (in degrees), or the x-direction-cosine (between -1 and 1).

`integer num_of_dirs_2` [Sub-variable of PhasorDomainNFFFT]

This is the number of observation directions over the second dimension of the two-dimensional observation-direction array. If `direction_spec` is `"theta-phi"`, this is the number of ϕ values; otherwise, the number of y-direction-cosines $s_y = \sin\theta\sin\phi$.

`floating-point dir2_min` [Sub-variable of PhasorDomainNFFFT]

`floating-point dir2_max` [Sub-variable of PhasorDomainNFFFT]

These are the minimum/maximum values of either the ϕ angle (in degrees), or the y-direction-cosine (between -1 and 1).

`floating-point limit_to_s` [Sub-variable of PhasorDomainNFFFT]
(*default:* `1`)

If `direction_spec` is `"dircosx-dircosy-upper"` or `"dircosx-dircosy-lower"` (the direction cosines are used to specify the observation directions), it might happen that some combinations of s_x and s_y do not correspond to a real observation direction, since $s_x^2 + s_y^2 > 1$. Such direction cosines are automatically assigned a far-field value of zero. If you would like to limit the direction cosines further into a narrower observation cone, you can choose the value of `limit_to_s` to be smaller than `1.0`. Then, the far field corresponding to the direction cosines satisfying $s_x^2 + s_y^2 >$(limit_to_s) are assigned a zero value. Although this could also be done in post processing, eliminating some observation directions in this way removes the burden of computing them in the first place. Specifying a `limit_to_s` value corresponds to reducing the *numerical aperture* in a microscope objective.

`boolean` [Sub-variable of PhasorDomainNFFFT]
`write_hertzian_dipole_far_field` (*default:* `false`)

If set to `true`, the theoretical far field due to the Hertzian point sources (see Section 6.7 [Point Sources], page 38) in the simulation grid is also written into

the output file. Any planar stratification is accounted for in the calculation of the theoretical far field, but the scattering from any other structure inside the grid is ignored. As such, this feature can be (and has been) used to test the phasor-domain NFFFT.

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`nffft_back_margin_x` (*units:m*)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`nffft_front_margin_x` (*units:m*)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`nffft_left_margin_y` (*units:m*)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`nffft_right_margin_y` (*units:m*)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`nffft_lower_margin_z` (*units:m*)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`nffft_upper_margin_z` (*units:m*)

`integer` [Sub-variable of PhasorDomainNFFFT]
`nffft_back_margin_x_in_cells` (*default:* `3`)

`integer` [Sub-variable of PhasorDomainNFFFT]
`nffft_front_margin_x_in_cells` (*default:* `3`)

`integer` [Sub-variable of PhasorDomainNFFFT]
`nffft_left_margin_y_in_cells` (*default:* `3`)

`integer` [Sub-variable of PhasorDomainNFFFT]
`nffft_right_margin_y_in_cells` (*default:* `3`)

`integer` [Sub-variable of PhasorDomainNFFFT]
`nffft_lower_margin_z_in_cells` (*default:* `3`)

`integer` [Sub-variable of PhasorDomainNFFFT]
`nffft_upper_margin_z_in_cells` (*default:* `3`)

The near field is collected over the surface of a rectangular prism in the grid, called the *NFFFT surface*. This surface should enclose all the scattering and/or radiating structures in the grid, as well as the total-field/scattered-field surface (see Section 6.10 [Incident Beams], page 62). By default, this rectangular box is placed 3 grid cells away from the PML boundary (see Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15). You can specify different margins to reduce the computational burden associated with the NFFFT. This burden is directly proportional to the surface area of the box. The margins can be specified in meters or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If given in meters, the margins are rounded to the nearest multiple of the spatial step size.

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`far_field_origin_x` (*units: m, default:* `0`)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`far_field_origin_y` (*units: m, default:* `0`)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`far_field_origin_z` (*units: m, default:* `0`)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`far_field_origin_x_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`far_field_origin_y_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of PhasorDomainNFFFT]
`far_field_origin_z_in_cells` (*default:* `0`)

These variables set the coordinates of the point relative to which the far field will be calculated. The distance r in [eq:far_field_angle_dependence], page 45 is with respect to this point. The coordinates of the far-field origin are with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`string far_field_dir` (*default:* `""`) [Sub-variable of PhasorDomainNFFFT]
This determines the subdirectory in which this individual far-field file will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `pd_nffft_output_dir` (see [pd_nffft_output_dir], page 46). By default, no subdirectory is created inside `pd_nffft_output_dir`.

`string far_field_file_name` [Sub-variable of PhasorDomainNFFFT]
(*default:* `"FarField_pd"`)
This determines the base string in the full name of the far-field file. Other information is appended to the name of the file to differentiate individual far-field files (see the example below).

`string far_field_file_extension` [Sub-variable of PhasorDomainNFFFT]
(*default:* `"hd5"`)
This is the extension of the far-field file name. If assigned the value `""`, no extension is added to the file. The HDF5 extension `"hd5"` is applied by default.

Here is an example far-field file name:

```
FarField_pd_0_1.hd5
```

The base string in the name of the file (`"FarField_pd"`) is specified by the `far_field_file_name` variable. The two integers that follow are the run index (see Section 6.14 [Multiple Simulation Runs], page 86) and the index of the phasor-domain NFFFT inside the `PhasorDomainNFFFT` list. For example, if there are two groups (two NFFFTs) in the `PhasorDomainNFFFT` list, the first one will write into

```
FarField_pd_0_0.hd5
```

while the second will write into

`FarField_pd_0_1.hd5`

If there are two simulation runs (i.e., `number_of_runs` is equal to 2 – see Section 6.14 [Multiple Simulation Runs], page 86), then the files created in the second run will have `1` instead of `0` as the first integer in the above file names. Finally, the extension (`"hd5"`) of the line files is determined by the variable `far_field_file_extension`.

6.8.2.1 HDF5 Content of Phasor-Domain NFFFT Output

The HDF5 file created as the output of the phasor-domain NFFFT can be viewed and modified using freely-available tools. One of these tools is HDFView, provided by the HDF Group. MATLAB also has built-in functions and tools that handle HDF5 files. For reference, a MATLAB script named '`hdf5_read.m`' is distributed as part of the Angora package, which reads an HDF5 dataset from an HDF5 file into a MATLAB array. This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here). For example, if you want to read the dataset named `lambda` from the file '`my_file.hd5`', use

```
>> lambda = hdf5_read('my_file.hd5','lambda');
```

In MATLAB R2011a and later, there is a high-level built-in function `h5read` that could be used for the same purpose.

The HDF5 datasets in the far-field file are the following:

- '`angora_version`': Integer array of length 3 with the major version, minor version, and revision numbers of the Angora package used to create the file.
- '`lambda`': 1-D array with the recorded free-space wavelength values (in m).
- If `direction_spec` is `"theta-phi"`:
  - '`theta`': 1-D array with the theta values.
  - '`phi`': 1-D array with the phi values.
- If `direction_spec` is `"dircosx-dircosy-upper"` or `"dircosx-dircosy-lower"`:
  - '`dircos_x`': 1-D array with the x-direction-cosine values.
  - '`dircos_y`': 1-D array with the y-direction-cosine values.
- Floating-point arrays with the real and imaginary parts of different components of the vector radiated electric field. See Figure 6.4 for a graphical illustration of the unit vectors in spherical coordinates. Note that only the angle-dependent part of the radiated electric field is calculated (see [eq:far_field_angle_dependence], page 45). Because the (1/r) dependence has been factored out, the units are in Volts. The first dimension is the wavelength, the second is either theta or the x-direction-cosine, and the third is either phi or the y-direction-cosine.

  '`E_theta_r`','`E_theta_i`': 3-D arrays with the real and imaginary parts of the theta component of the radiated electric field.
  - '`E_phi_r`','`E_phi_i`': 3-D arrays with the real and imaginary parts of the phi component of the radiated electric field.
- If the theoretical far field due to Hertzian point sources is also calculated (namely, `write_hertzian_dipole_far_field` is `true`):

'`E_theta_th_r`','`E_theta_th_i`': 3-D arrays with the *theoretical* real and imaginary parts of the theta component of the radiated electric field created by the Hertzian dipoles in the simulation grid.

- '`E_phi_th_r`','`E_phi_th_i`': 3-D arrays with the *theoretical* real and imaginary parts of the phi component of the radiated electric field created by the Hertzian dipoles in the simulation grid.

6.9 Optical Imaging

Angora can synthesize numerical **optical images** created by an ideal imaging system. The image is calculated in the form of a field distribution on a two-dimensional plane in the image space; which is assumed homogeneous. In photolithography, this image distribution is commonly called an *aerial image*.

The optical axis of the imaging system is currently limited to the z axis. The collection can be either through the +z or -z direction, allowing the simulation of reflection or transmission-mode imaging without changing the illumination scheme. Angora internally utilizes a near-field-to-far-field transformer (NFFFT) (see Section 6.8 [Near-Field-to-Far-Field Transformer], page 40) to calculate the optical image.

In Figure 6.5, a simplified representation is shown for the optical imaging geometry. The illumination scheme is not shown in the figure, and the collection is assumed to be through the upper half space. The entire optical system is represented by a single lens, although the system may comprise multiple lenses, apertures, stops, etc. The only assumptions regarding the optical system is that it satisfies Abbe's sine condition, and it is telecentric (see [Capoglu12b], page 89). Telecentricity implies that the entrance pupil is actually at infinity, although it is shown at a finite distance for ease of presentation in the figure.

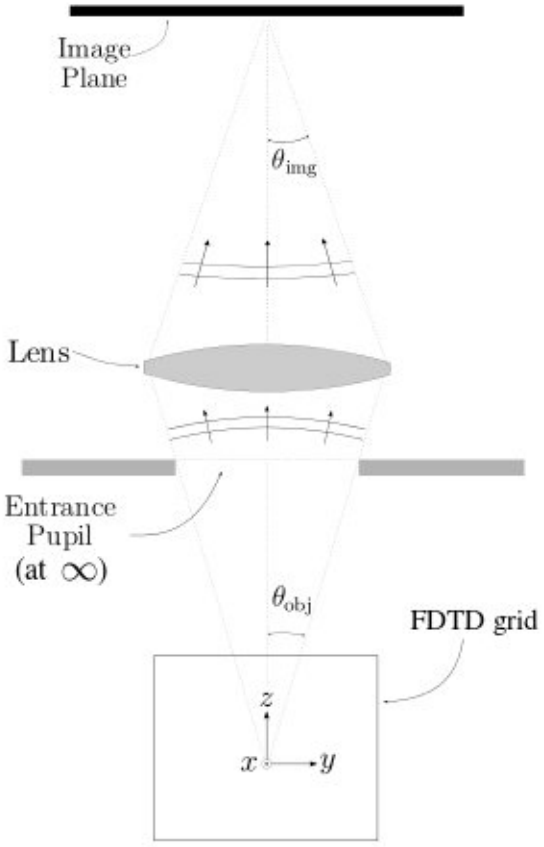

Figure 6.5: A simplified depiction of the optical imaging geometry.

The format used for the optical imaging output is **HDF5** (Hierarchical Data Format) (`http://www.hdfgroup.org/HDF5/`). The HDF5 format was chosen for its standard interface, and the availability of free software tools for inspecting and modifying HDF5 output. The HDF5 content of the optical image file is explained in more detail in Section 6.9.1 [Optical Image File HDF5 Content], page 61.

`string imaging_output_dir` (*default:* `"imaging/"`) [Global variable]
: This determines the subdirectory in which all the optical-imaging output will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `output_dir` (see Section 6.12 [Paths], page 85).

```
imaging_output_dir = "imaging";
OpticalImages:
{
        ...
        ...
};
```

`list OpticalImages` [Global variable]
: Optical images are defined in an `OpticalImages` list:

```
OpticalImages:
(
    {
        output_data = ["E_x_tot","E_y_tot","E_z_tot",
                       "E_x_sca","E_y_sca","E_z_sca",
                       "E_x_unsca","E_y_unsca","E_z_unsca",
                       "intensity_tot",
                       "intensity_sca",
                       "intensity_unsca"];
        num_of_lambdas = 5;
        lambda_min = 400e-9;
        lambda_max = 700e-9;
        lambda_spacing_type = "k-linear";
        do_not_include_first_lambda = false;
        do_not_include_last_lambda = false;
        ap_half_angle = 36.87;
        magnification = 40.0;
        image_space_refr_index = 1.0;
        image_expansion_factor_x = 1.0;
        image_expansion_factor_y = 1.0;
        image_oversampling_rate_x = 1.0;
        image_oversampling_rate_y = 1.0;
        coll_half_space = "upper";
        nffft_back_margin_x_in_cells = 3;
        nffft_front_margin_x_in_cells = 3;
        nffft_left_margin_y_in_cells = 3;
        nffft_right_margin_y_in_cells = 3;
        nffft_lower_margin_z_in_cells = 3;
        nffft_upper_margin_z_in_cells = 3;
        image_origin_x = 0.0;
        image_origin_y = 0.0;
        image_origin_z = 0.0;
        image_dir = "";
        image_file_name = "Image";
        image_file_extension = "hd5";
    },
    {
        ...
        ...
    }
);
```

`string-array output_data` [Sub-variable of OpticalImages]

This array of strings determines what will be included in the final output file. Any combination of the following strings can be listed in the array.

- `"E_x_sca"`, `"E_y_sca"`, `"E_z_sca"`: The x, y, and z components of the *scattered* electric field of the image. This is the electric field that is scattered

or generated by the structures inside the simulation grid. The incident beams (see Section 6.10 [Incident Beams], page 62) and the reflections and transmissions from the infinite planar layer interfaces are *not* included in the scattered field.

- `"E_x_unsca"`, `"E_y_unsca"`, `"E_z_unsca"`: The x, y, and z components of the *unscattered* electric field of the image. This is the electric field that would be created at the image plane in the absence of any scatterer inside the simulation grid except the infinite planar layers. The portion of the incident beams (see Section 6.10 [Incident Beams], page 62) and the reflections and transmissions from the infinite planar layer interfaces that fall into the collection aperture contribute to the unscattered field.
- `"E_x_tot"`, `"E_y_tot"`, `"E_z_tot"`: The x, y, and z components of the *total* electric field of the image, defined as the sum of the scattered and unscattered fields above.
- `"intensity_sca"`: The *scattered* light intensity at the image plane (in W/m^2), defined as $I_{sca} = n_{img}|E_{sca}|^2/\eta_0$ where n_{img} is the image-side refractive index, and η_0 is the free-space wave impedance (=376.7303...Ohms). E_{sca} is the scattered electric field vector.
- `"intensity_unsca"`: The *unscattered* light intensity at the image plane (in W/m^2), defined as $I_{unsca} = n_{img}|E_{unsca}|^2/\eta_0$ where n_{img} is the image-side refractive index, and η_0 is the free-space wave impedance (=376.7303...Ohms). E_{unsca} is the unscattered electric field vector.
- `"intensity_tot"`: The *total* light intensity at the image plane (in W/m^2), defined as $I_{tot} = n_{img}|E_{tot}|^2/\eta_0$ where n_{img} is the image-side refractive index, and η_0 is the free-space wave impedance (=376.7303...Ohms). E_{tot} is the total electric field vector.

For example, if the `output_data` array is

```
output_data = ["E_x_sca","intensity_tot"];
```

then only the x-component of the scattered electric field of the image and the total light intensity of the image are recorded in the output.

`integer num_of_lambdas` [Sub-variable of OpticalImages]

This specifies the number of wavelengths (in vacuum) at which the optical image will be calculated.

`floating-point lambda_min` (*units: m*) [Sub-variable of OpticalImages]

`floating-point lambda_min_in_cells` [Sub-variable of OpticalImages]

This value sets the lower limit of the wavelength range (in vacuum) over which the optical image is calculated. The optical image may or may not be calculated at the wavelength `lambda_min`, depending on the variable `do_not_include_first_lambda`. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`floating-point lambda_max` (*units: m*) [Sub-variable of OpticalImages]

`floating-point lambda_max_in_cells` [Sub-variable of OpticalImages]
This value sets the upper limit of the wavelength range (in vacuum) over which the optical image is calculated. The optical image may or may not be calculated at the wavelength `lambda_max`, depending on the variable `do_not_include_last_lambda`. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`string lambda_spacing_type` [Sub-variable of OpticalImages]
This string specifies how the wavelengths will be spaced between the two end points determined by `lambda_min`, `lambda_max`, `do_not_include_first_lambda`, and `do_not_include_last_lambda`.

- `"lambda-linear"`: The wavelengths are spaced lineary between the two end points.
- `"k-linear"`: The wavenumbers $k = 2\pi/\lambda$ are spaced linearly between the two end points. Since the wavenumber is also equal to $k = \omega/c$, where ω is the radian frequency, this causes the frequencies to be spaced linearly as well.
- `"log"`: The logarithms of the wavelengths (therefore the logarithms of the wavenumbers) are spaced linearly between the two end points.

`boolean do_not_include_first_lambda` [Sub-variable of OpticalImages]
(*default:* `false`)

`boolean do_not_include_last_lambda` [Sub-variable of OpticalImages]
(*default:* `false`)
Let's assume that `lambda_spacing_type` is `lambda-linear`. For `k-linear` and `log`, replace `lambda_min` in the following by $2\pi/$ `lambda_min` and log(`lambda_min`), respectively. The same applies to `lambda_max`.

- If `do_not_include_first_lambda=false` and `do_not_include_last_lambda=false`: The interval between `lambda_min` and `lambda_max` is divided into `(num_of_lambdas-1)` equal intervals. A total of `num_of_lambdas` wavelengths are placed linearly at the boundaries between the intervals, including both endpoints `lambda_min` and `lambda_max`.
- If `do_not_include_first_lambda=true` and `do_not_include_last_lambda=false`: The interval between `lambda_min` and `lambda_max` is divided into `num_of_lambdas` equal intervals. A total of `num_of_lambdas` wavelengths are placed linearly at the boundaries between the intervals, excluding the endpoint `lambda_min`.
- If `do_not_include_first_lambda=false` and `do_not_include_last_lambda=true`: The interval between `lambda_min` and `lambda_max` is divided into `num_of_lambdas` equal intervals. A total of `num_of_lambdas` wavelengths are placed linearly at the boundaries between the intervals, excluding the endpoint `lambda_max`.

- If `do_not_include_first_lambda=true` and `do_not_include_last_lambda=true`: The interval between `lambda_min` and `lambda_max` is divided into `num_of_lambdas` equal intervals. A total of `num_of_lambdas` wavelengths are placed at the midpoints of each interval.

`floating-point ap_half_angle` (*units: degrees*) [Sub-variable of OpticalImages]

This is the half-angle of the collection cone over which the far field is collected. This angle is represented by θ_{obj} in Figure 6.5

`floating-point magnification` (*default:* `1`) [Sub-variable of OpticalImages]

This is the absolute value of the *lateral magnification* of the optical imaging system. If greater than `1`, the imaging system shows a magnified image of the object. This is the case in *microscopy*, where the magnification ranges from 10 to 100. If less than `1`, the image is a de-magnified version of the object, This is the case in *photolithography*, where a de-magnified image of a mask is projected on a photoresist. Typical magnifications in photolithography are 0.1 to 0.25.

`floating-point image_space_refr_index` (*default:* `1`) [Sub-variable of OpticalImages]

This variable specifies the refractive index of the image space, assumed to be homogeneous.

`floating-point image_expansion_factor_x` (*default:* `1`) [Sub-variable of OpticalImages]

`floating-point image_expansion_factor_y` (*default:* `1`) [Sub-variable of OpticalImages]

By default, the optical image will only span the lateral (x-y) dimensions of the FDTD grid. The x and y dimensions of the image can be increased or decreased using these two factors. Setting these factors greater than 1 will reduce the aliasing effects in the numerical computation of the image, but linearly increase the computational burden associated with the far-field computation. This is because the far-field has to be collected at a denser set of observation directions for a larger image. The technical details of this are the subject of *sampling theory*, and are explained in [Capoglu12b], page 89.

`floating-point image_oversampling_rate_x` (*default:* `1`) [Sub-variable of OpticalImages]

`floating-point image_oversampling_rate_y` (*default:* `1`) [Sub-variable of OpticalImages]

Angora tries to automatically determine the minimum number of far-field collection directions to accurately synthesize the optical image. By default, the number of pixels in the final image is the same as the number of far-field collection directions. As a result, the image is sampled very economically; causing a pixelated appearance. A finer image can be synthesized by scaling the sampling rate in the x and y directions by modifying `image_oversampling_rate_x` and `image_oversampling_rate_y`, respectively. For example, setting

`image_oversampling_rate_x`=10 results in 10 times the default number of pixels in the x direction. Choosing high values for these two factors do not really cause much degradation in performance, since only the post-processing (post-simulation) computational burden is affected. The post-processing burden is usually a small fraction of the burden associated with the whole simulation.

In sampling theory, the minimum number of pixels needed to represent the image is determined by the *Nyquist sampling rate* for the image (see [Capoglu12b], page 89). There is a well-defined Nyquist sampling rate associated with every optical image, since they are constrained in spatial frequency content by the wavelength of illumination. If desired, the default image (the one obtained with `image_oversampling_rate_x`=`image_oversampling_rate_y`=1) can be made arbitrarily fine through *bandlimited interpolation* in post-processing.

`string coll_half_space` [Sub-variable of OpticalImages]

Although the collection apparatus seems to be situated in the upper (+z) half space in Figure 6.5, it can also be situated in the lower (-z) half space. This is specified by assigning the string `"upper"` or `"lower"` to the `coll_half_space` variable, respectively. In Figure 6.6, these two imaging geometries are shown separately. Note that the image is inverted, but the image-space coordinates (x' and y') are also inverted with respect to the object space.

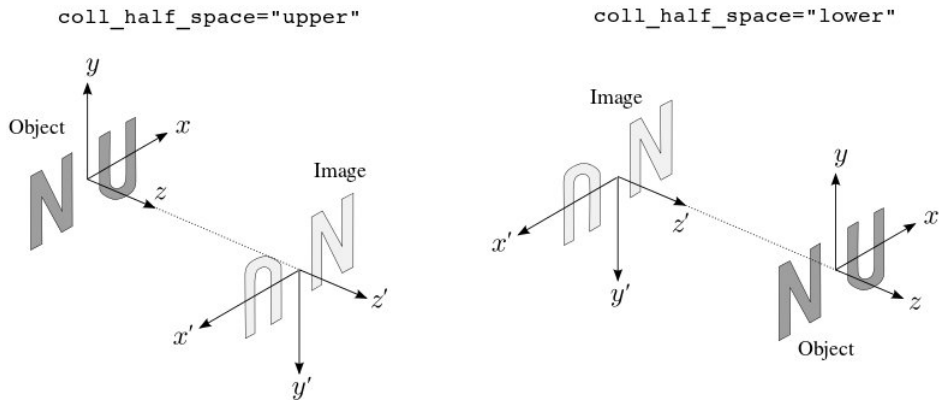

Figure 6.6: Collection of the scattered light in the upper half space (left figure), or the lower half space (right figure) for the calculation of the optical image.

`floating-point nffft_back_margin_x` [Sub-variable of OpticalImages]
(*units:m*)

`floating-point nffft_front_margin_x` [Sub-variable of OpticalImages]
(*units:m*)

`floating-point nffft_left_margin_y` [Sub-variable of OpticalImages]
(*units:m*)

`floating-point nffft_right_margin_y` [Sub-variable of OpticalImages]
(*units:m*)

`floating-point nffft_lower_margin_z` [Sub-variable of OpticalImages]
(*units:m*)

`floating-point nffft_upper_margin_z` [Sub-variable of OpticalImages]
(*units:m*)

`integer nffft_back_margin_x_in_cells` [Sub-variable of OpticalImages]
(*default:* `3`)

`integer nffft_front_margin_x_in_cells` [Sub-variable of OpticalImages]
(*default:* `3`)

`integer nffft_left_margin_y_in_cells` [Sub-variable of OpticalImages]
(*default:* `3`)

`integer nffft_right_margin_y_in_cells` [Sub-variable of OpticalImages]
(*default:* `3`)

`integer nffft_lower_margin_z_in_cells` [Sub-variable of OpticalImages]
(*default:* `3`)

`integer nffft_upper_margin_z_in_cells` [Sub-variable of OpticalImages]
(*default:* `3`)

In the collection stage of optical imaging (see Figure 6.5), the far field scattered from the sample is calculated using a near-field-to-far-field transformer (NFFFT). These variables determine the surface over which the near field is collected for the calculation of the far field. For more information, see Section 6.8 [Near-Field-to-Far-Field Transformer], page 40.

`floating-point image_origin_x` (*units: m,* [Sub-variable of OpticalImages]
default: `0`)

`floating-point image_origin_y` (*units: m,* [Sub-variable of OpticalImages]
default: `0`)

`floating-point image_origin_z` (*units: m,* [Sub-variable of OpticalImages]
default: `0`)

`floating-point` [Sub-variable of OpticalImages]
`image_origin_x_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of OpticalImages]
`image_origin_y_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of OpticalImages]
`image_origin_z_in_cells` (*default:* `0`)

These variables set the coordinates of the optical conjugate of the center of the image plane. Changing these values amounts to focusing at different positions and depths in the sample using the focusing knob on a microscope. The coordinates are with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

6.9.1 Optical Image File HDF5 Content

The HDF5 file created as the output of optical imaging can be viewed and modified using freely-available tools. One of these tools is HDFView, provided by the HDF Group. MATLAB also has built-in functions and tools that handle HDF5 files. For reference, a MATLAB script named '`hdf5_read.m`' is distributed as part of the Angora package, which reads an HDF5 dataset from an HDF5 file into a MATLAB array. This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here). For example, if you want to read the dataset named `lambda` from the file '`my_file.hd5`', use

```
>> lambda = hdf5_read('my_file.hd5','lambda');
```

In MATLAB R2011a and later, there is a high-level built-in function `h5read` that could be used for the same purpose.

The HDF5 datasets in the optical image file are the following:

- '`angora_version`': Integer array of length 3 with the major version, minor version, and revision numbers of the Angora package used to create the file.
- '`wv_range`': 1-D array with the recorded free-space wavelength values (in m).
- '`k_range`': 1-D array with the recorded free-space wavenumber values (in 1/m).
- '`x_range`','`y_range`': 1-D arrays with the x and y coordinates of the image (in m).
- '`n_obj`','`n_img`': Refractive indices of the object and image spaces, respectively.
- '`magnification`': Absolute value of the lateral magnification of the imaging system.
- '`ap_half_angle`': The half-angle of the collection cone over which the scattered light is collected (in degrees).
- '`E_x_sca_r`','`E_x_sca_i`': 3-D arrays with the real and imaginary parts of the x component of the scattered electric field in the image. (if `"E_x_sca"` is included in the array `output_data`)
- '`E_x_unsca_r`','`E_x_unsca_i`': 3-D arrays with the real and imaginary parts of the x component of the unscattered electric field in the image. (if `"E_x_unsca"` is included in the array `output_data`)
- '`E_x_tot_r`','`E_x_tot_i`': 3-D arrays with the real and imaginary parts of the x component of the total electric field in the image. (if `"E_x_tot"` is included in the array `output_data`)
- '`E_y_sca_r`','`E_y_sca_i`': 3-D arrays with the real and imaginary parts of the y component of the scattered electric field in the image. (if `"E_y_sca"` is included in the array `output_data`)
- '`E_y_unsca_r`','`E_y_unsca_i`': 3-D arrays with the real and imaginary parts of the y component of the unscattered electric field in the image. (if `"E_y_unsca"` is included in the array `output_data`)
- '`E_y_tot_r`','`E_y_tot_i`': 3-D arrays with the real and imaginary parts of the y component of the total electric field in the image. (if `"E_y_tot"` is included in the array `output_data`)

- '`E_z_sca_r`','`E_z_sca_i`': 3-D arrays with the real and imaginary parts of the z component of the scattered electric field in the image. (if `"E_z_sca"` is included in the array `output_data`)
- '`E_z_unsca_r`','`E_z_unsca_i`': 3-D arrays with the real and imaginary parts of the z component of the unscattered electric field in the image. (if `"E_z_unsca"` is included in the array `output_data`)
- '`E_z_tot_r`','`E_z_tot_i`': 3-D arrays with the real and imaginary parts of the z component of the total electric field in the image. (if `"E_z_tot"` is included in the array `output_data`)
- '`intensity_sca`': 3-D array with the intensity of the scattered light at the image plane (if `"intensity_sca"` is included in the array `output_data`). See above for the definition of '`intensity_sca`'.
- '`intensity_unsca`': 3-D array with the intensity of the unscattered light at the image plane (if `"intensity_unsca"` is included in the array `output_data`). See above for the definition of '`intensity_unsca`'.
- '`intensity_tot`': 3-D array with the intensity of the total light at the image plane (if `"intensity_tot"` is included in the array `output_data`) See above for the definition of '`intensity_tot`'.
-

6.10 Incident Beams

Different types of incident beams can be sourced into the simulation grid using the `TFSF` group.

`group TFSF` [Global variable]

The `TFSF` group contains definitions for various types of incident electromagnetic beams required for scattering problems. Angora uses the *total-field/scattered-field* (TF/SF) technique to source incident beams into the simulation grid (see [TafloveHagness], page 89). In this technique, a rectangular surface surrounding the scatterer is designated the total-field/scattered-field boundary (or the *TF/SF box* in short); and the electromagnetic field on this surface is supplemented by certain terms proportional to the incident electromagnetic field. These additional terms create the incident field inside the surface (suggesting the term *injection*), while maintaining a very small electromagnetic field outside the surface. The region outside the TF/SF box only harbors the *scattered field* created by the scatterers inside the TF/SF box. The field inside the box is the *total field*, which is a sum of the incident field and the scattered field. Since the boundary divides the grid into total-field and scattered-field regions, the term "TF/SF boundary" is justified.

TF/SF incident beam injection is supported for **infinite planar layered media**. Infinite planar layers are created using the `MaterialSlabs` variable (see Section 6.5.2 [Planar Layers], page 24). Currently, only layers with different permittivities and electrical conductivities are supported. Permeability variations across layers will also be supported in the future. Angora also supports **evanescent plane waves** resulting from plane waves passing from a high-permittivity layer to a low-permittivity one at a low grazing angle. Angora supports evanescent waves only for *narrowband* plane

waves, which have appreciable frequency components only in a small band around a center frequency. A modulated Gaussian waveform with a large $f_0\tau$ can be used as a narrowband waveform in cases where evanescent waves might be present (see Section 6.6.3 [Modulated-Gaussian Waveforms], page 37).

Different types of incident beams are defined in their respective lists inside the `TFSF` group. These are explained in the following subsections.

```
TFSF:
{
    PlaneWaves:
    (
        {
            ...
            ...
        }
    );
    FocusedLaserBeams:
    (
        {
            ...
            ...
        }
    );
};
```

6.10.1 Plane Waves

A plane wave is one of the simplest solutions of Maxwell's equations; with the electric field

$$\bar{\mathbf{E}}(\bar{\mathbf{r}},t) = \hat{}\mathbf{e} E_0 \, f(t - \bar{\mathbf{r}} \cdot \hat{}\mathbf{k}^i / v_p)$$

where $\hat{}\mathbf{k}^i$ is the unit vector in the direction of propagation, $\hat{}\mathbf{e}$ is the electric-field unit vector, E_0 is the electric field amplitude, and $\bar{\mathbf{r}}$ is the distance vector. The velocity of propagation v_p is determined by the material properties in the direction from which the plane wave is incident. The time waveform $f(t)$ is arbitrary. Inserting the above expression into Maxwell's equations, it is found that the electric-field unit vector $\hat{}\mathbf{e}$ is perpendicular to the direction of propagation, as well as the magnetic-field unit vector.

In a discrete FDTD grid, a plane wave propagates at a slightly lower velocity than in continuum. Furthermore, there is an intrinsic grid velocity anisotropy that results from the inherent rotational asymmetry of the rectangular FDTD grid. These are partially alleviated in Angora by the use of the *matched numerical dispersion* (MND) technique (see [TafloveHagness], page 89).

`list PlaneWaves` [Sub-variable of TFSF]

Plane waves are defined inside a `PlaneWaves` list inside the `TFSF` group:

```
TFSF:
{
    PlaneWaves:
```

```
        (
            {
                theta = 40.0;
                phi = 90.0;
                psi = 90.0;
                waveform_tag = "waveform1";
                pw_extra_amplitude = 1.0;
                tfsf_back_margin_x_in_cells = 6;
                tfsf_front_margin_x_in_cells = 6;
                tfsf_left_margin_y_in_cells = 6;
                tfsf_right_margin_y_in_cells = 6;
                tfsf_lower_margin_z_in_cells = 6;
                tfsf_upper_margin_z_in_cells = 6;
                pw_origin_x = 0.0;
                pw_origin_y = 0.0;
                pw_origin_z = 0.0;
                display_warnings = true;
                min_cells_per_lambda = 15.0;
            },
            {
                ...
                ...
            }
        );
    };
```

`floating-point theta` (*units: degrees*) [Sub-variable of PlaneWaves]

`floating-point phi` (*units: degrees*) [Sub-variable of PlaneWaves]

The incidence angles of the plane waves are defined in terms of the traditional spherical-coordinate variables (θ, ϕ). The first of these angles is the *zenith angle*, while the second is the *azimuth angle*. In Figure 6.7, the definitions of these angles are shown schematically. The `theta` variable specifies the zenith angle in degrees. Although this angle is traditionally defined between 0 and 180deg, `theta` can be assigned any negative or positive value. The `phi` variable specifies the azimuth angle in degrees. Although this angle is traditionally defined between 0 and 360deg, `phi` can be assigned any negative or positive value.

Note that the incidence angles (θ, ϕ) specify the direction *from which the plane wave is incident*; **not** the direction in which it propagates.

`floating-point psi` (*units: degrees*) [Sub-variable of PlaneWaves]

This variable is used to specify the polarization of the electric field of the incident plane wave. Maxwell's equations dictate that the electric field is **perpendicular** to the incidence vector $\hat{}\mathbf{k}^i$. In order to define the orientation of the electric vector unambiguously, a local coordinate system (ξ, η) is defined, such that $\hat{}\xi = \hat{}\mathbf{k}^i \times \hat{}z$ and $\hat{}\eta = \hat{}\xi \times \hat{}\mathbf{k}^i$ The unit vectors $(\hat{}\xi, \hat{}\eta)$ are perpendicular to each other, and lie in the plane perpendicular to the incidence vector $\hat{}\mathbf{k}^i$.

The *polarization angle* ψ of the electric-field unit vector $\hat{\mathbf{e}}$ is defined as the left-handed (clockwise) rotation angle around the axis defined by the incidence vector $\hat{\mathbf{k}}^i$. The variable `psi` sets this angle in degrees.

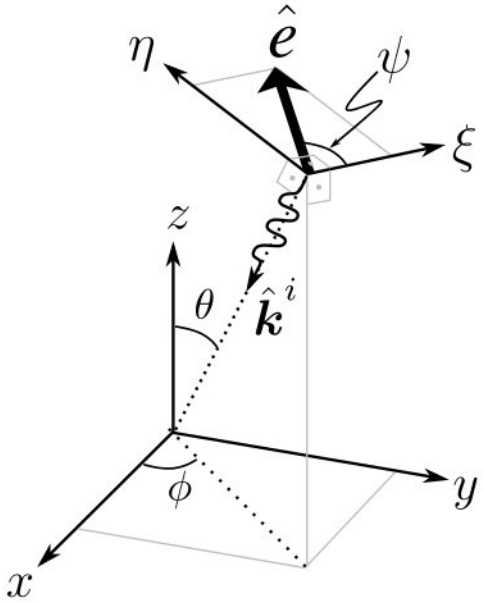

Figure 6.7: Graphical description of the incidence and polarization angles associated with a plane wave.

`string waveform_tag` [Sub-variable of PlaneWaves]
This string variable specifies the electric-field waveform $f(t)$ in [eq:pw_E_field], page 63. The waveform is interpreted in (Volts/m) units. This should match a previously-defined tag in a `Waveforms` definition (see Section 6.6 [Waveforms], page 34).

`floating-point pw_extra_amplitude` (*units:* *V/m*, *default:* `1.0`) [Sub-variable of PlaneWaves]
This variable sets the electric field amplitude E_0 in [eq:pw_E_field], page 63.

`floating-point tfsf_back_margin_x` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point tfsf_front_margin_x` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point tfsf_left_margin_y` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point tfsf_right_margin_y` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point tfsf_lower_margin_z` [Sub-variable of PlaneWaves]
(*units:* *m*)

`floating-point tfsf_upper_margin_z` [Sub-variable of PlaneWaves]
(*units:* *m*)

`integer tfsf_back_margin_x_in_cells` [Sub-variable of PlaneWaves]
(*default:* `6`)

`integer tfsf_front_margin_x_in_cells` [Sub-variable of PlaneWaves]
(*default:* `6`)

`integer tfsf_left_margin_y_in_cells` [Sub-variable of PlaneWaves]
(*default:* `6`)

`integer tfsf_right_margin_y_in_cells` [Sub-variable of PlaneWaves]
(*default:* `6`)

`integer tfsf_lower_margin_z_in_cells` [Sub-variable of PlaneWaves]
(*default:* `6`)

`integer tfsf_upper_margin_z_in_cells` [Sub-variable of PlaneWaves]
(*default:* `6`)

By default, the total-field/scattered-field (TF/SF) surface is placed 6 grid cells away from the PML boundary (see Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15). You can specify different margins to reduce the computational burden associated with the TF/SF operation. This burden is directly proportional to the area of the TF/SF surface. The margins can be specified in meters or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If given in meters, the margins are rounded to the nearest multiple of the spatial step size.

`floating-point pw_origin_x` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point pw_origin_y` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point pw_origin_z` (*units:* *m*) [Sub-variable of PlaneWaves]

`floating-point pw_origin_x_in_cells` [Sub-variable of PlaneWaves]
(*default:* `0`)

`floating-point pw_origin_y_in_cells` [Sub-variable of PlaneWaves]
(*default:* `0`)

`floating-point pw_origin_z_in_cells` [Sub-variable of PlaneWaves]
(*default:* `0`)

These variables set the coordinates of the point relative to which the distance $\bar{\mathbf{r}}$ is defined in [eq:pw_E_field], page 63. The coordinates are with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`boolean display_warnings` (*default:* `true`) [Sub-variable of PlaneWaves]

`floating-point min_cells_per_lambda` (*default:* `15`) [Sub-variable of PlaneWaves]

The boolean variable `display_warnings` enables or disables the printing of warning messages. Currently, a warning is displayed only when there are not enough grid cells per "minimum" wavelength in the excitation waveform. This minimum wavelength is defined to be the one at which the spectrum of the waveform falls to -40dB below its maximum. The number of required grid cells per the minimum wavelength is determined by the `min_cells_per_lambda` variable.

6.10.2 Focused Laser Beams

Angora can synthesize *focused laser beams* created by an aplanatic optical system (i.e., free of spherical aberration and coma) illuminated by a normally-incident *paraxial Hermite-Gaussian laser mode*. The electromagnetic formulation of the focused beam is based on the classic work of Richards and Wolf (see [Richards59], page 89). This formulation is interpreted as a sum of plane waves, and approximated by a finite sum in the FDTD implementation (see [Capoglu08], page 89). It is assumed that the Hermite-Gaussian laser mode filling the entrance pupil of the optical system has a beam width much larger than the wavelength, and is therefore in the paraxial regime. In this regime, the wavefronts are almost planar, perpendicular to the optical axis, and the electric field of the beam has a negligible longitudinal component.

The incidence geometry for the laser mode illuminating the entrance pupil of the system is shown on the upper right in Figure 6.8. A local coordinate system (ξ, η) is defined on the plane of the entrance pupil, such that $\hat{}\xi = \hat{}\mathbf{k}^i \times \hat{}z$ and $\hat{}\eta = \hat{}\xi \times \hat{}\mathbf{k}^i$ The unit vectors $(\hat{}\xi, \hat{}\eta)$ are perpendicular to each other, and lie in the plane perpendicular to the incidence vector $\hat{}\mathbf{k}^i$. The symmetry axes (x', y') of the Hermite-Gaussian beam are rotated at an angle of α with respect to the ξ axis, in a clock-wise (left-handed) sense with respect to $\hat{}\mathbf{k}^i$.

On the plane of the entrance pupil, which is assumed to coincide with the waist of the Hermite-Gaussian beam, the electric field is given by

$$\bar{\mathbf{E}}(\bar{\mathbf{r}}, t) = \hat{}\mathbf{e} E_0 \, f(t) H_m(\sqrt{2}\frac{x'}{w_0}) H_n(\sqrt{2}\frac{y'}{w_0}) \exp(-\frac{(x')^2 + (y')^2}{w_0^2})$$

where w_0 is the beam half width (or beam waist radius), $\hat{}\mathbf{e}$ is the electric-field unit vector, E_0 is the electric field amplitude, and $H_m(\cdot)$ are the (physicists') *Hermite polynomials* of order m. The first few Hermite polynomials are

$$H_0(x) = 1, \quad H_1(x) = 2x, \quad H_2(x) = 4x^2 - 2$$

The time waveform $f(t)$ is arbitrary. Since we assume that the width of the beam at the entrance pupil is much larger than the wavelengths contained in the waveform $f(t)$, it is reasonable to employ the *paraxial* approximation, in which the rays propagate parallel to the optical axis and the electric-field unit vector $\hat{}\mathbf{e}$ is perpendicular to the direction of propagation $\hat{}\mathbf{k}^i$.

The focusing optical system between the entrance pupil and the exit pupil is assumed *aplanatic*. This means that, in addition to focusing on-axis points stigmatically (without

spherical aberration), the system also correctly focuses off-axis points up to the first order in off-axis distance. The latter condition corresponds to the absence of circular coma. The *sine condition*, first derived by Ernst Abbe in 1881, is an expression of this in mathematical terms. The sine condition reads

$$h = f \sin(\theta_{\text{ill}})$$

where h is the radius of the entrance pupil, and f is the back focal length of the focusing system. The sine condition is also shown geometrically in Figure 6.8.

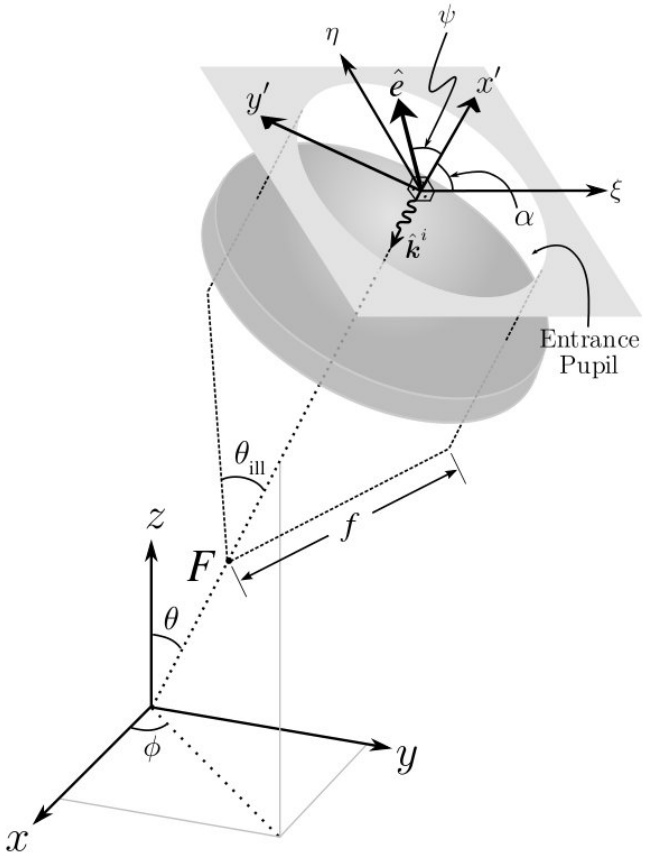

Figure 6.8: Geometry of the focused laser beam.

`list FocusedLaserBeams` [Sub-variable of TFSF]

Focused laser beams are defined inside a `FocusedLaserBeams` list inside the `TFSF` group:

```
TFSF:
{
    FocusedLaserBeams:
    (
        {
            theta = 40.0;
            phi = 90.0;
```

```
                psi = 90.0;
                alpha = 0;
                x_order = 1;
                y_order = 1;
                waveform_tag = "waveform1";
                flb_extra_amplitude = 1.0;
                ap_half_angle = 23.5782;
                back_focal_length = 0.1;
                filling_factor = 1;
                object_space_refr_index = 1.0;
                tfsf_back_margin_x_in_cells = 6;
                tfsf_front_margin_x_in_cells = 6;
                tfsf_left_margin_y_in_cells = 6;
                tfsf_right_margin_y_in_cells = 6;
                tfsf_lower_margin_z_in_cells = 6;
                tfsf_upper_margin_z_in_cells = 6;
                flb_origin_x = 0.0;
                flb_origin_y = 0.0;
                flb_origin_z = 0.0;
                display_warnings = true;
                min_cells_per_lambda = 15.0;
            },
            {
                ...
                ...
            }
        );
    };
```

`floating-point theta` (*units: degrees*) [Sub-variable of FocusedLaserBeams]

`floating-point phi` (*units: degrees*) [Sub-variable of FocusedLaserBeams]

The incidence angles for the focused laser beam are defined in reference to the paraxial beam that hits the entrance pupil (see Figure 6.8). The incidence angles are defined in terms of the traditional spherical-coordinate variables (θ, ϕ). The first of these angles is the *zenith angle*, while the second is the *azimuth angle*. The `theta` variable specifies the zenith angle in degrees. Although this angle is traditionally defined between 0 and 180deg, `theta` can be assigned any negative or positive value. The `phi` variable specifies the azimuth angle in degrees. Although this angle is traditionally defined between 0 and 360deg, `phi` can be assigned any negative or positive value.

Note that the incidence angles (θ, ϕ) specify the direction *from which the paraxial beam is incident*; **not** the direction in which it propagates.

`floating-point psi` (*units: degrees*) [Sub-variable of FocusedLaserBeams]

This variable is used to specify the polarization of the transverse electric field of the paraxial beam on the entrance pupil. The electric field is perpendicular to the incidence vector $\hat{}\mathbf{k}^i$. The *polarization angle* ψ is defined as the left-handed

(clockwise) rotation angle with respect to the incidence vector $\hat{\mathbf{k}}^i$ between the symmetry axis x' and the electric-field unit vector $\hat{\mathbf{e}}$ (see Figure 6.8). The variable `psi` sets this angle in degrees.

`floating-point alpha` (*units: degrees*) [Sub-variable of FocusedLaserBeams]
(*default:* `0`)
This angle specifies the rotation of the symmetry axes (x', y') of the Hermite-Gaussian beam with respect to the local coordinate axes (ξ, η). The rotation is clock-wise (left-handed) with respect to the incidence vector $\hat{\mathbf{k}}^i$ (see Figure 6.8).

`integer x_order` [Sub-variable of FocusedLaserBeams]

`integer y_order` [Sub-variable of FocusedLaserBeams]
These positive integers specify the orders of the Hermite polynomials $H_m(x), H_n(y)$ in the definition of the Hermite-Gaussian beam in [eq:fb_incident_E_field], page 67. `x_order` and `y_order` correspond to m and n, respectively.

`string waveform_tag` [Sub-variable of FocusedLaserBeams]
This string variable specifies the electric-field waveform $f(t)$ in [eq:fb_incident_E_field], page 67, except a time advance t_0 equal to the time of propagation from the entrance pupil to the focal point F. In other words, the actual waveform $f(t)$ on the entrance pupil is advanced in time by t_0 with respect to the waveform represented by `waveform_tag`. This is needed because we simulate the fields around the focus F; not the entrance pupil. The waveform is interpreted in (Volts/m) units. The string `waveform_tag` should match a previously-defined string tag in a `Waveforms` definition (see Section 6.6 [Waveforms], page 34).

`floating-point` [Sub-variable of FocusedLaserBeams]
`flb_extra_amplitude` (*units: V/m, default:* `1.0`)
This variable sets the electric field amplitude E_0 in [eq:fb_incident_E_field], page 67.

`floating-point ap_half_angle` [Sub-variable of FocusedLaserBeams]
(*units: degrees*)
This variable sets the half-angle θ_{ill} of the illumination cone in degrees (see Figure 6.8).

If a planar layered medium is present in the grid (see Section 6.5.2 [Planar Layers], page 24), the incidence cone bounded by the angle θ_{ill} should lie *entirely* within the upper or lower half space. In other words, the incidence cone *cannot* cut across the grazing direction ($\theta = \pi/2$) . Angora will throw an error if this is found to be the case.

`floating-point back_focal_length` [Sub-variable of FocusedLaserBeams]
(*units: m*)

`floating-point back_focal_length_in_cells` [Sub-variable of FocusedLaserBeams]
This variable specifies the back focal length f of the optical system. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`floating-point filling_factor` [Sub-variable of FocusedLaserBeams]
This dimensionless parameter represents the ratio between the half-width of the incident paraxial beam and the radius of the entrance pupil. The larger this ratio, the more overfilled the pupil, and vice versa. This parameter is defined as

$$f_0 = \frac{w_0}{f \sin(\theta_{\text{ill}})}$$

`floating-point object_space_refr_index` (*default:* `1.0`) [Sub-variable of FocusedLaserBeams]
This variable specifies the refractive index of the object space of the optical system; i.e., the space from which the paraxial beam is incident. Note that the FDTD simulation grid is in the image space of the optical system; therefore the refractive index of the image space is determined by the material filling the FDTD grid. If you want to simulate an oil immersion scenario, for example, you should fill the FDTD simulation space with the material representing the immersion oil. The object-side refractive index `object_space_refr_index` is seldom different than `1.0`, which is the default value.

`floating-point tfsf_back_margin_x` [Sub-variable of FocusedLaserBeams]
(*units: m*)

`floating-point tfsf_front_margin_x` (*units: m*) [Sub-variable of FocusedLaserBeams]

`floating-point tfsf_left_margin_y` [Sub-variable of FocusedLaserBeams]
(*units: m*)

`floating-point tfsf_right_margin_y` (*units: m*) [Sub-variable of FocusedLaserBeams]

`floating-point tfsf_lower_margin_z` (*units: m*) [Sub-variable of FocusedLaserBeams]

`floating-point tfsf_upper_margin_z` (*units: m*) [Sub-variable of FocusedLaserBeams]

`integer tfsf_back_margin_x_in_cells` (*default:* `6`) [Sub-variable of FocusedLaserBeams]

`integer tfsf_front_margin_x_in_cells` (*default:* `6`) [Sub-variable of FocusedLaserBeams]

`integer` [Sub-variable of FocusedLaserBeams]
`tfsf_left_margin_y_in_cells` (*default:* `6`)

`integer` [Sub-variable of FocusedLaserBeams]
`tfsf_right_margin_y_in_cells` (*default:* `6`)

`integer` [Sub-variable of FocusedLaserBeams]
`tfsf_lower_margin_z_in_cells` (*default:* `6`)

`integer` [Sub-variable of FocusedLaserBeams]
`tfsf_upper_margin_z_in_cells` (*default:* `6`)

By default, the total-field/scattered-field (TF/SF) surface is placed 6 grid cells away from the PML boundary (see Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15). You can specify different margins to reduce the computational burden associated with the TF/SF operation. This burden is directly proportional to the area of the TF/SF surface. Because the focused beam is actually a collection of many (often hundreds) of plane waves, the reduction of the surface area of the TF/SF box greatly helps the simulation performance.

The margins can be specified in meters or in grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If given in meters, the margins are rounded to the nearest multiple of the spatial step size.

`floating-point flb_origin_x` (*units:* *m*) [Sub-variable of FocusedLaserBeams]

`floating-point flb_origin_y` (*units:* *m*) [Sub-variable of FocusedLaserBeams]

`floating-point flb_origin_z` (*units:* *m*) [Sub-variable of FocusedLaserBeams]

`floating-point` [Sub-variable of FocusedLaserBeams]
`flb_origin_x_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of FocusedLaserBeams]
`flb_origin_y_in_cells` (*default:* `0`)

`floating-point` [Sub-variable of FocusedLaserBeams]
`flb_origin_z_in_cells` (*default:* `0`)

These variables set the coordinates of the back focal point F of the focusing lens. (see Figure 6.8). The coordinates are with respect to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name.

`boolean display_warnings` (*default:* `true`) [Sub-variable of FocusedLaserBeams]

`floating-point` [Sub-variable of FocusedLaserBeams]
`min_cells_per_lambda` (*default:* `15`)

The boolean variable `display_warnings` enables or disables the printing of warning messages. Currently, a warning is displayed only when there are not

enough grid cells per "minimum" wavelength in the excitation waveform. This minimum wavelength is defined to be the one at which the spectrum of the waveform falls to -40dB below its maximum. The number of required grid cells per the minimum wavelength is determined by the `min_cells_per_lambda` variable.

6.11 Recording

Angora can record field values computed during a simulation into a file in a variety of ways. Field values can be recorded on a cross-section of the grid, along a line through the grid, or at a given point in the grid. Currently, Angora only supports the recording of the electric or the magnetic field. Recording of other field-related quantities such as energy, flux, Poynting's vector, etc. will be implemented in the future. Please send any comments, suggestions, and requests to `help@angorafdtd.org`.

`string recorder_output_dir` (*default:* `"recorder"`) [Global variable]

This determines the subdirectory in which all the recording-related stuff will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `output_dir` (see Section 6.12 [Paths], page 85).

```
recorder_output_dir = "recorder";
Recorder:
{
        ...
        ...
};
```

`group Recorder` [Global variable]

The `Recorder` group contains the sub-variables related to different types of field recording. These are explained in the following subsections.

```
Recorder:
{
    MovieRecorders:
    (
        {
            ...
            ...
        }
    );
    LineRecorders:
    (
        {
            ...
            ...
        }
    );
    FieldValueRecorders:
    (
        {
```

```
                ...
                ...
            }
        );
    };
```

6.11.1 Movie Recording

Angora can record field components on a two-dimensional cross section of the grid into a custom movie file. The binary format used for movie recording is described in more detail in Section 6.11.1.1 [Movie File Format], page 77.

`string movie_recorder_output_dir` (*default:* `""`) [Sub-variable of Recorder]

This determines the subdirectory in which all the recorded movie files will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `recorder_output_dir` (see [recorder_output_dir], page 73).

```
Recorder:
{
    movie_recorder_output_dir = "movies";
    MovieRecorders:
    (
        ...
        ...
    );
};
```

`list MovieRecorders` [Sub-variable of Recorder]

Field values on a two-dimensional cross section of the FDTD grid can be recorded using the `MovieRecorders` list.

```
Recorder:
{
    MovieRecorders:
    (
        {
            recorded_section = "xz";
            recorded_position = 0;
            recorded_component = "Ex";
            recording_scale = "dB";
            recording_type = "uchar1";
            movie_dir = "this_movie_dir";
            movie_file_name = "MovieFile"
            movie_file_extension = "amv";
            only_records_material_info = false;
        },
        {
            ...
            ...
        }
```

```
        );
    };
```

`string recorded_section` [Sub-variable of MovieRecorders]
This determines the cross section of the grid over which the field is recorded. Currently, only xz, yz, and xy cross sections are supported. These are represented by the string values `"xz"`, `"yz"`, and `"xy"`, respectively.

`floating-point recorded_position` [Sub-variable of MovieRecorders]
(*units: m*)

`integer recorded_position_in_cells` [Sub-variable of MovieRecorders]
This value specifies the coordinate of the recorded cross section along the perpendicular direction (e.g., the z direction if `recorded_section` is `"xy"`). The coordinate is relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

`string recorded_component` [Sub-variable of MovieRecorders]
An individual movie recorder (in a group delineated by the curly brackets '`{}`') only records a single scalar value extracted from the vector-valued electromagnetic field. This could be one of the Cartesian components or the absolute value of the electric or the magnetic field. These are represented by the string values `"Ex"`, `"Ey"`, `"Ez"`, `"E"`, `"Hx"`, `"Hy"`, `"Hz"`, and `"H"`. If you would like to record multiple Cartesian components of a vector field, simply add other movie recorders (i.e., other groups, see Section 5.2.5 [Groups], page 11) to the `MovieRecorders` list with the desired `recorded_component` values.

`string recording_scale` [Sub-variable of MovieRecorders]
If `"linear"`, the raw values are recorded. If `"absolute"`, the absolute value is taken before recording. If `"dB"`, the decibel value ($20\log_{10}(|\cdot|)$) is recorded.

`string recording_type` [Sub-variable of MovieRecorders]
Movies can either be recorded either in raw floating-point format, or in a single-byte compressed format. This is specified by assigning the string values `"dbl8"` or `"uchar1"` to the `recording_type` variable, respectively. Using the single-byte format reduces the file size considerably, but results in some data loss.

If `recording_type` is `"dbl8"`, then the field values are recorded in 8-byte `double` datatype, after processed in accordance with the `recording_scale` specification above. This provides practically lossless recording, albeit with increased computational burden and file size.

With the `"uchar1"` option, the field values are reduced to 256 discrete bins within a fixed *dynamic range*. This requires only a single byte per field value; reducing the storage requirement by a factor of 8.

- If `recording_type` is `"dB"`, the maximum and minimum values in this dynamic range are determined by the global variables `max_field_value` and `dB_accuracy` (see Section 6.2.7 [Dynamic Range], page 17):

max : $20\log_{10}(|\text{max\_field\_value}|)$ **min** : $20\log_{10}(|\text{max\_field\_value}|) + (\text{dB\_accuracy})$

The `dB_accuracy` variable should always be negative; therefore the minimum value in the dynamic range is lower than the maximum.

- If `recording_type` is `"linear"` or`"absolute"`, the maximum and minimum values are determined only by the global variable `max_field_value` (see Section 6.2.7 [Dynamic Range], page 17):

  max : max_field_value **min** : (−max_field_value) or 0

`string movie_dir` (*default:* `""`) [Sub-variable of MovieRecorders]
: This determines the subdirectory in which this individual movie file will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `movie_recorder_output_dir` (see [movie_recorder_output_dir], page 74). By default, no subdirectory is created inside `movie_recorder_output_dir`.

`string movie_file_name` (*default:* `"MovieFile"`) [Sub-variable of MovieRecorders]
: This determines the base string in the full name of the movie file. Other information is appended to the name of the file to differentiate individual movie files (see the example below).

`string movie_file_extension` (*default:* `"amv"`) [Sub-variable of MovieRecorders]
: This is the extension of the movie file name. If assigned the value `""`, no extension is added to the file.

Here is an example movie file name:

```
MovieFile_Ex_0_1.amv
```

The base string in the name of the file (`"MovieFile"`) is specified by the `movie_file_name` variable. The second part of the file name, `"Ex"`, is determined by the recorded field component. The two integers that follow are the run index (see Section 6.14 [Multiple Simulation Runs], page 86) and the index of the movie inside the `MovieRecorders` list. For example, if there are two groups (two movies) in the `MovieRecorders` list, the first one will write into

```
MovieFile_Ex_0_0.amv
```

while the second will write into

```
MovieFile_Ex_0_1.amv
```

If there are two simulation runs (i.e., `number_of_runs` is equal to 2 – see Section 6.14 [Multiple Simulation Runs], page 86), then the files created in the second run will have `1` instead of `0` as the first integer in the above file names. Finally, the extension (`"amv"`) of the movie files is determined by the variable `movie_file_extension`.

`boolean only_records_material_info` (*default:* `false`) [Sub-variable of MovieRecorders]
: If set to `true`, only the material information is recorded into the file, and no field recording is performed during the simulation.

6.11.1.1 Movie File Format

Angora records movies in a custom binary format for better speed and performance. Please be aware that this format is subject to modification. The changes in the format will be documented in this manual as necessary. You may refer to the '`ChangeLog`' file in the Angora distribution for recent changes in the movie recording format.

The MATLAB script '`angora_movie.m`', distributed as part of the Angora package, reads an Angora movie file and displays it as a MATLAB movie. It can also save the movie in AVI format. This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here).

The movie file is composed of chunks of data, ordered as follows. For each chunk, a short explanation (and maybe an alias) is given, followed by a description of the datatype in parantheses.

- major package version (integer, 4 bytes)
- minor package version (integer, 4 bytes)
- package revision number (integer, 4 bytes)
- number of bytes used to record each field component (integer, 4 bytes): This is either equal to `1` or `8`, depending on the `recording_type` variable.
- spatial step size (double, 8 bytes)
- temporal step size (double, 8 bytes)
- time value that corresponds to the beginning of the simulation (double, 8 bytes): This is usually a negative value, since time waveforms frequently begin before t=0.
- maximum value in the field discretization range (double, 8 bytes): This is the maximum value in the discretization dynamic range for single-byte recording (i.e., `recording_type` is `"uchar1"`). If `recording_type` is `"dbl8"`, this value is irrelevant. Same applies to the next value in the file.
- minimum value in the field discretization range (double, 8 bytes)
- '`length_1`': length along the first dimension of the recorded array (integer, 4 bytes): If the xy section were recorded, this would be the length of the array in the x dimension. This includes the thickness of the PML sections in both directions.
- '`length_2`': length along the second dimension of the recorded array (integer, 4 bytes): If the xy section were recorded, this would be the length of the array in the y dimension. This includes the thickness of the PML sections in both directions.
- '`length_time`': number of time steps in the simulation (integer, 4 bytes)
- thickness of the PML region, in grid cells (integer, 4 bytes): See Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15 for more information on the PML. The PML sections are included in the recorded cross sectional area. They can easily be removed in post-processing.
- an array of length `length_1` with the actual physical coordinates (in m) along the first dimension of the recorded cross section (double, 8 bytes)
- an array of length `length_2` with the actual physical coordinates (in m) along the second dimension of the recorded cross section (double, 8 bytes)

- an array of length (`length_2` x `length_1`) holding the relative permittivity (if the electric field is recorded) or the relative permeability (if the magnetic field is recorded) values on the recorded cross section (double, 8 bytes): In Figure 6.9, it is assumed that the xy section is recorded, and the field positions are numbered from 0 to 11. It is seen that the **second dimension** (here, the y-dimension) is looped through first. These positions are laid out in the movie file in the same order:

```
0  1  2  3  4  5  6  7  8  9  10  11
```

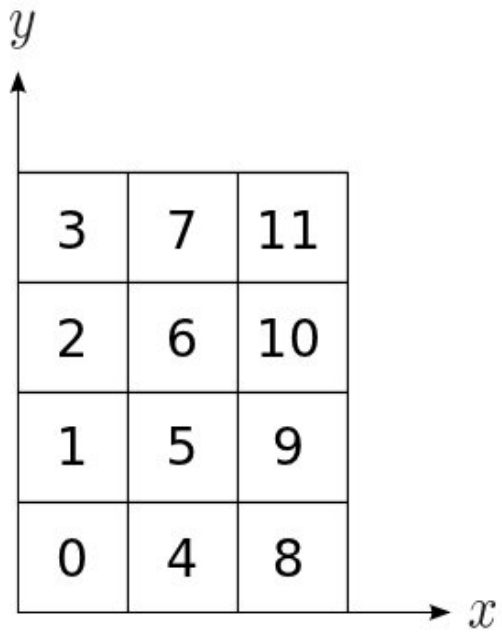

Figure 6.9: The layout of the field positions on the recorded cross section in the movie file.

- an array of length (`length_2` x `length_1`) holding the electric conductivity values (in Siemens/m, if the electric field is recorded) or the magnetic conductivity values (in Ohm/m, if the magnetic field is recorded) on the recorded cross section (double, 8 bytes): The 2D cross section is laid out in the movie file in the same way as the previous array.
- arrays (movie frames) of length (`length_2` x `length_1`) holding the field values on the recorded cross section (double, 8 bytes OR unsigned char, 1 byte – depending on `recording_type`): The total number of these movie frames is equal to `length_time`, read earlier from the binary file. Each of these frames is laid out in the movie file in the same way as the previous arrays.

6.11.2 Line Recording

Angora can record field components along a line into a file. The binary format used for line recording is described in more detail in Section 6.11.2.1 [Line File Format], page 81.

`string line_recorder_output_dir` (*default:* `""`) [Sub-variable of Recorder]
This determines the subdirectory in which all the recorded line files will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `recorder_output_dir` (see [recorder_output_dir], page 73).

```
Recorder:
{
    line_recorder_output_dir = "lines";
    LineRecorders:
    (
        ...
        ...
    );
};
```

`list LineRecorders` [Sub-variable of Recorder]

```
Recorder:
{
    LineRecorders:
    (
        {
            line_orientation = "y_directed";
            line_position_x1 = 0;
            line_position_x2 = 0;
            recorded_component = "Ex";
            recording_scale = "linear";
            line_dir = "this_line_dir";
            line_file_name = "LineFile";
            line_file_extension = "aln";
        },
        {
            ...
            ...
        }
    );
};
```

`string line_orientation` [Sub-variable of LineRecorders]
There are three possible orientations for the line over which the field values are recorded .These orientations are along the three principal axes of the grid; namely, the x,y, and z directions. These are specified by the strings `"x_directed"`, `"y_directed"`, and `"z_directed"`, respectively.

`floating-point line_position_x1` (*units:* *m*) [Sub-variable of LineRecorders]

`integer line_position_x1_in_cells` [Sub-variable of LineRecorders]
This is the first of the remaining two coordinates that specify the position of the recorded line. The coordinate is relative to the grid origin (see Section 6.2.6

[Coordinate Origin], page 16). For example, if the line is oriented in the y direction (`line_orientation` is `"y_directed"`), then `line_position_x1_in_cells` specifies the x coordinate of the line. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

`floating-point line_position_x2` (*units:* *m*) [Sub-variable of LineRecorders]

`integer line_position_x2_in_cells` [Sub-variable of LineRecorders]
This is the second of the remaining two coordinates that specify the position of the recorded line. The coordinate is relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). For example, if the line is oriented in the y direction (`line_orientation` is `"y_directed"`), then `line_position_x2_in_cells` specifies the z coordinate of the line. The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

`string recorded_component` [Sub-variable of LineRecorders]
An individual line recorder (in a group delineated by the curly brackets '`{}`') only records a single scalar value extracted from the vector-valued electromagnetic field. This could be one of the Cartesian components or the absolute value of the electric or the magnetic field. These are represented by the string values `"Ex"`, `"Ey"`, `"Ez"`, `"E"`, `"Hx"`, `"Hy"`, `"Hz"`, and `"H"`. If you would like to record multiple Cartesian components of a vector field, simply add other line recorders (i.e., other groups, see Section 5.2.5 [Groups], page 11) to the `LineRecorders` list with the desired `recorded_component` values.

`string recording_scale` [Sub-variable of LineRecorders]
If `"linear"`, the raw values are recorded. If `"absolute"`, the absolute value is taken before recording. If `"dB"`, the decibel value ($20\log_{10}(|\cdot|)$) is recorded.

`string line_dir` (*default:* `""`) [Sub-variable of LineRecorders]
This determines the subdirectory in which this individual line file will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `line_recorder_output_dir` (see [line_recorder_output_dir], page 78). By default, no subdirectory is created inside `line_recorder_output_dir`.

`string line_file_name` (*default:* `"LineFile"`) [Sub-variable of LineRecorders]
This determines the base string in the full name of the line file. Other information is appended to the name of the file to differentiate individual line files (see the example below).

`string line_file_extension` (*default:* `"aln"`) [Sub-variable of LineRecorders]
This is the extension of the line file name. If assigned the value `""`, no extension is added to the file.

Here is an example line file name:

```
LineFile_Ey_Y_0_1.aln
```

The base string in the name of the file (`"LineFile"`) is specified by the `line_file_name` variable. The second part of the file name, `"Ey"`, is determined by the recorded field component. The following string `"Y"` indicates the orientation of the line, which is y-directed for this example. The two integers that follow are the run index (see Section 6.14 [Multiple Simulation Runs], page 86) and the index of the line recorder inside the `LineRecorders` list. For example, if there are two groups (two line recorders) in the `LineRecorders` list, the first one will write into

```
LineFile_Ey_Y_0_0.aln
```

while the second will write into

```
LineFile_Ey_Y_0_1.aln
```

If there are two simulation runs (i.e., `number_of_runs` is equal to 2 – see Section 6.14 [Multiple Simulation Runs], page 86), then the files created in the second run will have `1` instead of `0` as the first integer in the above file names. Finally, the extension (`"aln"`) of the line files is determined by the variable `line_file_extension`.

6.11.2.1 Line File Format

As with movies, Angora records the line files in a custom binary format for better speed and performance. Please be aware that this format is subject to modification. The changes in the format will be documented in this manual as necessary. You may refer to the '`ChangeLog`' file in the Angora distribution for recent changes in the line recording format.

The MATLAB script '`angora_line.m`', distributed as part of the Angora package, reads an Angora line file and displays it as a MATLAB movie. This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here).

The line file is composed of chunks of data, ordered as follows. For each chunk, a short explanation (and maybe an alias) is given, followed by a description of the datatype in parantheses.

- major package version (integer, 4 bytes)
- minor package version (integer, 4 bytes)
- package revision number (integer, 4 bytes)
- temporal step size (double, 8 bytes)
- time value that corresponds to the beginning of the simulation (double, 8 bytes): This is usually a negative value, since time waveforms frequently begin before t=0.
- '`total_length`': the number of recorded elements on each line snapshot (integer, 4 bytes)
- '`length_time`': number of time steps in the simulation (integer, 4 bytes)
- thickness of the PML region, in grid cells (integer, 4 bytes): See Section 6.2.4 [Perfectly-Matched Layer (PML)], page 15 for more information on the PML. The recorded line includes two PML sections on opposite ends, each with this length. These sections can easily be removed in post-processing.

- arrays (line snapshots) of length `total_length` holding the field values on the recorded line (double, 8 bytes): The total number of these line snapshots is equal to `length_time`, read earlier from the binary file.

6.11.3 Field-Value Recording

Angora can record the time history of the field at a given position in the simulation grid. The format used for this sort of recording is **HDF5** (Hierarchical Data Format) (`http://www.hdfgroup.org/HDF5/`). The HDF5 format was chosen for its standard interface, and the availability of free software tools for inspecting and modifying HDF5 output. The HDF5 output created by the field-value recorder is explained in more detail in Section 6.11.3.1 [Field-Value File HDF5 Content], page 84.

`string field_value_recorder_output_dir` (*default:* `""`) [Sub-variable of Recorder]

This determines the subdirectory in which all the recorded field-value files will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `recorder_output_dir` (see [recorder_output_dir], page 73).

```
Recorder:
{
    field_value_recorder_output_dir = "fieldvalues";
    FieldValueRecorders:
    (
        ...
        ...
    );
};
```

`list FieldValueRecorders` [Sub-variable of Recorder]

```
Recorder:
{
    FieldValueRecorders:
    (
        {
            coord_x = 0;
            coord_y = 0;
            coord_z = 0;
            recorded_component = "Ex";
            recording_scale = "linear";
            field_value_dir = "this_field_value_dir";
            field_value_file_name = "FieldValueFile";
            field_value_file_extension = "hd5";
        },
        {
            ...
            ...
        }
    );
```

```
};
```

`floating-point coord_x` (*units: m*) [Sub-variable of FieldValueRecorders]

`integer coord_x_in_cells` [Sub-variable of FieldValueRecorders]
This is the x coordinate of the recorded position in the simulation grid. It is relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

`floating-point coord_y` (*units: m*) [Sub-variable of FieldValueRecorders]

`integer coord_y_in_cells` [Sub-variable of FieldValueRecorders]
This is the y coordinate of the recorded position in the simulation grid. It is relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

`floating-point coord_z` (*units: m*) [Sub-variable of FieldValueRecorders]

`integer coord_z_in_cells` [Sub-variable of FieldValueRecorders]
This is the z coordinate of the recorded position in the simulation grid. It is relative to the grid origin (see Section 6.2.6 [Coordinate Origin], page 16). The units are either in meters or grid cells. For the latter, the `_in_cells` suffix should be appended to the variable name. If the coordinate corresponds to a non-integer cell position, the closest integer position is chosen.

`string recorded_component` [Sub-variable of FieldValueRecorders]
An individual field-value recorder (in a group delineated by the curly brackets '`{}`') only records a single scalar value extracted from the vector-valued electromagnetic field. This could be one of the Cartesian components or the absolute value of the electric or the magnetic field. These are represented by the string values `"Ex"`, `"Ey"`, `"Ez"`, `"E"`, `"Hx"`, `"Hy"`, `"Hz"`, and `"H"`. If you would like to record multiple Cartesian components of a vector field, simply add other field-value recorders (i.e., other groups, see Section 5.2.5 [Groups], page 11) to the `FieldValueRecorders` list with the desired `recorded_component` values.

`string recording_scale` [Sub-variable of FieldValueRecorders]
If `"linear"`, the raw values are recorded. If `"absolute"`, the absolute value is taken before recording. If `"dB"`, the decibel value ($20\log_{10}(|\cdot|)$) is recorded.

`string field_value_dir` (*default:* `""`) [Sub-variable of FieldValueRecorders]
This determines the subdirectory in which this individual field-value file will be placed. Unless it has a slash '`/`' up front; this path is interpreted as being relative to `field_value_recorder_output_dir` (see [field_value_recorder_output_dir], page 82). By default, no subdirectory is created inside `field_value_recorder_output_dir`.

`string field_value_file_name` [Sub-variable of FieldValueRecorders]
(*default:* `"FieldValueFile"`)

This determines the base string in the full name of the field-value file. Other information is appended to the name of the file to differentiate individual field-value files (see the example below).

`string field_value_file_extension` [Sub-variable of FieldValueRecorders]
(*default:* `"hd5"`)

This is the extension of the field-value file name. If assigned the value `""`, no extension is added to the file. The HDF5 extension `"hd5"` is applied by default.

Here is an example field-value file name:

```
FieldValueFile_Ex_0_1.hd5
```

The base string in the name of the file (`"FieldValueFile"`) is specified by the `field_value_file_name` variable. The second part of the file name, `"Ex"`, is determined by the recorded field component. The two integers that follow are the run index (see Section 6.14 [Multiple Simulation Runs], page 86) and the index of the field-value recorder inside the `FieldValueRecorders` list. For example, if there are two groups (two field-value recorders) in the `FieldValueRecorders` list, the first one will write into

```
FieldValueFile_Ex_0_0.hd5
```

while the second will write into

```
FieldValueFile_Ex_0_1.hd5
```

If there are two simulation runs (i.e., `number_of_runs` is equal to 2 – see Section 6.14 [Multiple Simulation Runs], page 86), then the files created in the second run will have `1` instead of `0` as the first integer in the above file names. Finally, the extension (`"hd5"`) of the line files is determined by the variable `field_value_file_extension`.

6.11.3.1 Field-Value File HDF5 Content

The HDF5 file created as the output of the field-value recorder can be viewed and modified using freely-available tools. One of these tools is HDFView, provided by the HDF Group. MATLAB also has built-in functions and tools that handle HDF5 files. For reference, a MATLAB script named '`angora_fieldvalue.m`' is distributed as part of the Angora package, which reads an Angora field-value file and plots the recorded waveform. This script is installed in the directory '`$(prefix)/share/angora/`' (see Chapter 3 [Compilation and Installation], page 7). If Angora was installed without any `$(prefix)` configuration option, the default location is '`/usr/local/share/angora/`'. This script can also be downloaded directly from the Angora website (link here).

The HDF5 datasets in the field-value file are the following:

- '`angora_version`': An integer array of length 3 with the major version, minor version, and revision numbers of the Angora package used to create the file.
- '`time_step`': A floating-point value specifying the temporal step in the simulation (in sec).
- '`initial_time_value`': A floating-point value specifying the time value corresponding to the beginning of the simulation (in sec). This is usually a negative value, since time waveforms frequently begin before t=0.

- '`field_values`': A 1-D floating-point array with the recorded field values.

6.12 Paths

`string angora_basepath` (*default:* `"."`) [Global variable]

This variable specifies the base directory for all the input-output operations in Angora. If there is no slash '`/`' in front of the path, it is interpreted as a relative path starting from the working directory (i.e, the one from which Angora is launched.)

Any other input or output directory will be assumed *relative* to `angora_basepath`. An overarching exception is when a directory is specified with a slash '`/`' up front; in which case that directory will be taken as an *absolute path*, and not relative to `angora_basepath`.

`string output_dir` (*default:* `"output"`) [Global variable]

This is the base directory for all the output that will result from Angora. It is interpreted as being relative to `angora_basepath`, unless it is preceded by a slash '`/`'. All other output directories are created as subdirectories of this directory.

Example:

```
angora_basepath = "angora_stuff";
output_dir = "data";
```

With these variable assignments, all the output will be written into subdirectories within '`./angora_stuff/data/`'.

`string input_dir` [Global variable]

This is the base directory for all the input that will be read by Angora. It is interpreted as being relative to `angora_basepath`, unless it is preceded by a slash '`/`'. Unless the path to an input file is absolute (i.e., preceded by a slash '`/`'), it is interpreted as being relative to `input_dir`.

Example:

```
angora_basepath = "angora_stuff";
input_dir = "input_data";
```

With these variable assignments, the input base directory becomes '`./angora_stuff/input_data/`'.

6.13 Logging

You can keep a log of the simulations run by Angora in a log file, which contains several lines of information for each simulation. First, an estimate of the finishing time and duration of the simulation is written into the log entry. The actual finishing time and duration is added to the log entry upon completion of the simulation.

Here is an example entry for a simulation in the log file:

```
johndoe started Angora on 02/22/12 11:54:36AM
    Estimated to finish on 02/22/12 11:54:42AM
    Estimated duration : 6 seconds.
    Simulation finished on 02/22/12 11:54:42AM
    Elapsed time : 6 seconds.
```

`boolean enable_logging` (*default:* `"true"`) [Global variable]
: If set to '`true`', Angora will keep a record of the simulations that it runs in a log file. The name of this log file is specified by the `log_file_name` variable, and the directory in which this file resides is specified by the `log_output_dir` variable.

`string log_file_name` (*default:* `"angora.log"`) [Global variable]
: This is the name of the Angora log file. It resides in the directory specified by the `log_output_dir` variable.

`string log_output_dir` (*default:* `"log"`) [Global variable]
: This is the directory in which the Angora log file is kept. Unless it is preceded by a slash '`/`', it is taken as relative to the base output directory `output_dir` (see Section 6.12 [Paths], page 85).

6.14 Multiple Simulation Runs

A number of consecutive Angora simulations can be set up in a single configuration file.

`integer number_of_runs` (*default:* `1`) [Global variable]
: The number of simulation runs is specified by the `number_of_runs` variable. The simulation runs (or *runs* for short) are indexed from `0` to `number_of_runs-1`. You can refer to these indices later in the configuration file for enabling or disabling certain configuration variables for certain runs (see [enabled_for_runs], page 86).

`integer-array disabled_runs` (*default: none*) [Global variable]
: This array of integers (see Section 5.2.6 [Arrays], page 12) lists the run indices for simulations that will be skipped.

  Example:

  ```
  disabled_runs = [1,2,3,4,5];
  ```

  If `number_of_runs` was `7`, the above variable will cause only the simulations with indices `0` and `6` to be run.

`integer-array disabled_run_range` (*default: none*) [Global variable]
: If you would like to disable simulations that correspond to a *range* of run indices, you can use this variable. This has to be an array of integers with only two elements (see Section 5.2.6 [Arrays], page 12). Simulations with run indices between (and including) these two integers will be skipped.

  The following variable assignment has the same effect as the one in the previous example:

  ```
  disabled_run_range = [1,5];
  ```

`integer-array enabled_for_runs` (*default: all*) [Sub-variable of any group]

`integer-array enabled_for_run_range` (*default: all*) [Sub-variable of any group]
: Certain variables can be *enabled* or *disabled* for any of the simulation runs using the `enabled_for_runs` or `enabled_for_run_range` arrays. These arrays can be used inside any group structure (see Section 5.2.5 [Groups], page 11) to specify the run indices for which that group is enabled.

The `enabled_for_runs` array simply lists the run indices for which the specific group is enabled. For example,

```
number_of_runs = 4;
PointSources:
(
    {
        //point source #1
        enabled_for_runs = [0,1,2];
        coord_x_in_cells = 0;
        coord_y_in_cells = 0;
        coord_z_in_cells = 0;
        source_orientation = "x_directed";
        waveform_tag = "waveform1";
    },
    {
        //point source #2
        enabled_for_runs = [3];
        coord_x_in_cells = 0;
        coord_y_in_cells = 0;
        coord_z_in_cells = 0;
        source_orientation = "x_directed";
        waveform_tag = "waveform2";
    }
);
```

In this example, each group represents a collection of variable assignments that characterize an individual point source. With the `enabled_for_runs` variables set as shown, simulations `0`, `1`, and `2` will be run with the first point source; whereas simulation `3` will be run with the second point source.

Alternatively, the `enabled_for_run_range` array can be used to specify a range of run indices for which the group is enabled. This should be an integer array of length two. It specifies the lower and upper limits of the range of run indices for the specific group. For example,

```
number_of_runs = 10;
PointSources:
(
    {
        enabled_for_run_range = [0,5];
        coord_x_in_cells = 0;
        coord_y_in_cells = 0;
        coord_z_in_cells = 0;
        source_orientation = "x_directed";
        waveform_tag = "waveform1";
    }
);
```

This point source will only be enabled for run indices `0`, `1`, `2`, `3`, `4`, and `5`.

6.15 Miscellaneous

6.15.1 Auto-Saving the Configuration

Angora can automatically save a record of every simulation configuration that it processes.

`boolean auto_save_cfg` (*default:* `"false"`) [Global variable]
: If set to '`true`', Angora will automatically write every simulation configuration it runs into another configuration file, and save it in the directory specified by `cfg_output_dir`. A time/date string is appended to the name of the saved file to differentiate between subsequent executions of the same configuration file.

`string cfg_output_dir` (*default:* `"cfg"`) [Global variable]
: This is the directory in which the auto-saved configuration files are placed. Unless it is preceded by a slash '`/`', it is taken as relative to the base output directory `output_dir` (see Section 6.12 [Paths], page 85).

7 References

[Matzler02] C. Matzler, "MATLAB Functions for Mie Scattering and Absorption Version 2," [Online]. Available: http://www.iap.unibe.ch/publications/download/199/en/. [Accessed April 2012].

[Roden00] J. A. Roden and S. D. Gedney, "Convolution PML (CPML): an efficient FDTD implementation of the CFD-PML for arbitrary media," Microw. Opt. Technol. Lett., vol. 27, pp. 334-9, Dec. 2000.

[Kuzuoglu96] M. Kuzuoglu and R. Mittra, "Frequency dependence of the constitutive parameters of causal perfectly matched absorbers," IEEE Microwave Guided Wave Lett., vol. 6, pp. 447-449, Dec. 1996.

[Berenger02] J.-P. Berenger,"Numerical reflection from FDTD-PMLs: a comparison of the split PML with the unsplit and CFS PMLs," IEEE Trans. Antennas Propag, vol. 50, pp.258-265, Mar 2002.

[Rogers09] J. D. Rogers, I. R. Capoglu, V. Backman, "Nonscalar elastic light scattering from continuous media in the Born approximation", Optics Letters, vol. 34, no. 12, pp. 1891-1893, 2009.

[Richards59] B. Richards and E. Wolf, "Electromagnetic diffraction in optical systems. II. Structure of the image field in an aplanatic system," Proc. Roy. Soc. A, vol. 253, no. 1274, pp.358379, Dec. 1959.

[Capoglu08] I. R. Capoglu, A. Taflove, and V. Backman, "Generation of an incident focused light pulse in FDTD," Optics Express, vol. 16, no. 23, pp. 19,20819,220, Nov. 2008.

[Capoglu12] I. R. Capoglu, A. Taflove, V. Backman, "A frequency-domain near-field-to-far-field transform for planar layered media", IEEE Trans. Antennas Propag, vol. 60 No. 04, Apr. 2012.

[Capoglu12b] I. R. Capoglu, J. D. Rogers, A. Taflove, V. Backman, "The microscope in a computer: Image synthesis from three-dimensional full-vector solutions of Maxwells equations at the nanometer scale", Progress in Optics, (to appear in vol. 57, 2012.)

[TafloveHagness] A. Taflove and S. C. Hagness (2005). Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd ed.. Artech House Publishers. ISBN 1-58053-832-0.

[Hwang01] K.-P. Hwang and A. C. Cangellaris, Effective permittivities for second-order accurate FDTD equations at dielectric interfaces, IEEE Microw. Wireless Compon. Lett., vol. 11, no. 4, pp. 15860, Apr. 2001.

List of Figures

Figure 1.1: Scattering from a sphere illuminated by a plane wave incident from the -z direction. 2
Figure 1.2: The absolute scattered electric field phasor amplitude on the xz plane at 509.1 nm. 5
Figure 6.1: The location of the coordinate origin in the FDTD grid for (`origin_x_in_cells`,`origin_y_in_cells`,`origin_z_in_cells`)=(2,3,2). 17
Figure 6.2: A 2D slice of an example zero-mean sample distribution. This distribution can be assigned to different constitutive parameters of a material. 29
Figure 6.3: The illustration of the column-major ordering of a three-dimensional array. The values indicated by numbers should be laid out in the file in the same order. 30
Figure 6.4: The angles and unit vectors for spherical coordinates. 46
Figure 6.5: A simplified depiction of the optical imaging geometry. 54
Figure 6.6: Collection of the scattered light in the upper half space (left figure), or the lower half space (right figure) for the calculation of the optical image. 59
Figure 6.7: Graphical description of the incidence and polarization angles associated with a plane wave. 65
Figure 6.8: Geometry of the focused laser beam. 68
Figure 6.9: The layout of the field positions on the recorded cross section in the movie file. 78

Indices

Configuration Variable Index

A

alpha 70
amplitude 35, 36, 38
anchor 32
angora_basepath 85
ap_half_angle 58, 70
append_run_index_to_name 31
auto_save_cfg 88

B

back_coord_x 19
back_coord_x_in_cells 19
back_focal_length 71
back_focal_length_in_cells 71

C

center_coord_x 21
center_coord_x_in_cells 21
center_coord_y 21
center_coord_y_in_cells 21
center_coord_z 21
center_coord_z_in_cells 21
cfg_output_dir 88
coll_half_space 59
constitutive_param_type 26, 31
coord 34
coord_in_cells 34
coord_x 33, 39, 83
coord_x_in_cells 33, 39, 83
coord_y 33, 39, 83
coord_y_in_cells 33, 39, 83
coord_z 33, 39, 83
coord_z_in_cells 33, 39, 83
corr_len 28
corr_len_in_cells 28
courant 15
cpml_feature_size 16
cpml_feature_size_in_cells 16

D

datatype 33
dB_accuracy 18
delay 35, 36, 38
differentiated 38
DifferentiatedGaussianWaveforms 36
dir1_max 49
dir1_min 49
dir2_max 49
dir2_min 49
direction_spec 48
disabled_run_range 86
disabled_runs 86
display_warnings 67, 72
do_not_include_first_lambda 48, 57
do_not_include_last_lambda 48, 57
dx 15

E

electric_conductivity 22
enable_logging 86
enabled_for_run_range 86
enabled_for_runs 86

F

f_0 38
far_field_dir 43, 51
far_field_file_extension 43, 51
far_field_file_name 43, 51
far_field_origin_x 43, 51
far_field_origin_x_in_cells 43, 51
far_field_origin_y 43, 51
far_field_origin_y_in_cells 43, 51
far_field_origin_z 43, 51
far_field_origin_z_in_cells 43, 51
field_value_dir 83
field_value_file_extension 84
field_value_file_name 84
field_value_recorder_output_dir 82
FieldValueRecorders 82
file_extension 31
file_name 31
filling_factor 71
flb_extra_amplitude 70
flb_origin_x 72
flb_origin_x_in_cells 72
flb_origin_y 72
flb_origin_y_in_cells 72
flb_origin_z 72
flb_origin_z_in_cells 72
FocusedLaserBeams 68
front_coord_x 19
front_coord_x_in_cells 20

G

GaussianWaveforms 35
grid_dimension_x 15

grid_dimension_x_in_cells 15
grid_dimension_y 15
grid_dimension_y_in_cells 15
grid_dimension_z 15
grid_dimension_z_in_cells 15
GroundPlanes 33

I

image_expansion_factor_x 58
image_expansion_factor_y 58
image_origin_x 60
image_origin_x_in_cells 60
image_origin_y 60
image_origin_y_in_cells 60
image_origin_z 60
image_origin_z_in_cells 60
image_oversampling_rate_x 58
image_oversampling_rate_y 58
image_space_refr_index 58
imaging_output_dir 54
input_dir 85

J

j_0 39

L

lambda_max 47, 57
lambda_max_in_cells 47, 57
lambda_min 47, 56
lambda_min_in_cells 47, 56
lambda_spacing_type 47, 57
left_coord_y 19
left_coord_y_in_cells 20
limit_to_s 49
line_dir 80
line_file_extension 80
line_file_name 80
line_orientation 79
line_position_x1 79
line_position_x1_in_cells 79
line_position_x2 80
line_position_x2_in_cells 80
line_recorder_output_dir 79
LineRecorders 79
log_file_name 86
log_output_dir 86
lower_coord_z 19
lower_coord_z_in_cells 20

M

m 28
magnetic_conductivity 22
magnification 58
material_tag 22, 23, 24
Materials 21
MaterialsFromFiles 29
MaterialSlabs 24
max_coord 24
max_coord_in_cells 24
max_field_value 18
max_new_materials 33
mean 27
min_cells_per_lambda 67, 72
min_coord 24
min_coord_in_cells 24
ModulatedGaussianWaveforms 37
modulation_type 38
movie_dir 76
movie_file_extension 76
movie_file_name 76
movie_recorder_output_dir 74
MovieRecorders 74

N

n_diff 37
nffft_back_margin_x 42, 50, 59
nffft_back_margin_x_in_cells 42, 50, 60
nffft_front_margin_x 42, 50, 59
nffft_front_margin_x_in_cells 42, 50, 60
nffft_left_margin_y 42, 50, 59
nffft_left_margin_y_in_cells 42, 50, 60
nffft_lower_margin_z 42, 50, 60
nffft_lower_margin_z_in_cells 42, 50, 60
nffft_right_margin_y 42, 50, 59
nffft_right_margin_y_in_cells 42, 50, 60
nffft_upper_margin_z 42, 50, 60
nffft_upper_margin_z_in_cells 42, 50, 60
num_of_dirs_1 49
num_of_dirs_2 49
num_of_lambdas 47, 56
num_of_time_steps 16
number_of_runs 86

O

object_space_refr_index 71
Objects 23
only_records_material_info 76
OpticalImages 54
origin_x 16
origin_x_in_cells 17
origin_y 16
origin_y_in_cells 17
origin_z 17
origin_z_in_cells 17
output_data 55
output_dir 85

P

pd_nffft_output_dir 46

phase 38
PhasorDomainNFFFT 46
phi 41, 64, 69
PlaneWaves 63
pml_thickness 15
pml_thickness_in_cells 15
PointSources 38
psi 64, 69
pw_extra_amplitude 65
pw_origin_x 66
pw_origin_x_in_cells 66
pw_origin_y 66
pw_origin_y_in_cells 66
pw_origin_z 66
pw_origin_z_in_cells 66

R

radius 21
radius_in_cells 21
random_seed 28
RandomMaterials 25
recorded_component 75, 80, 83
recorded_position 75
recorded_position_in_cells 75
recorded_section 75
Recorder 73
recorder_output_dir 73
recording_scale 75, 80, 83
recording_type 75
RectangularBoxes 19
rel_permeability 22
rel_permittivity 22
right_coord_y 19
right_coord_y_in_cells 20

S

shape_tag 19, 20, 24, 28
Shapes 18
SimulationSpace 22
source_orientation 39
Spheres 20
std_dev 27

T

tau 35, 36, 38
td_nffft_output_dir 40
TFSF 62
tfsf_back_margin_x 65, 71
tfsf_back_margin_x_in_cells 66, 71
tfsf_front_margin_x 65, 71
tfsf_front_margin_x_in_cells 66, 71
tfsf_left_margin_y 65, 71
tfsf_left_margin_y_in_cells 66, 72
tfsf_lower_margin_z 66, 71
tfsf_lower_margin_z_in_cells 66, 72
tfsf_right_margin_y 65, 71
tfsf_right_margin_y_in_cells 66, 72
tfsf_upper_margin_z 66, 71
tfsf_upper_margin_z_in_cells 66, 72
theta 41, 64, 69
TimeDomainNFFFT 41
transparent 22

U

upper_coord_z 19
upper_coord_z_in_cells 20

W

waveform_tag 35, 36, 38, 39, 65, 70
Waveforms 34
WhittleMaternCorrelated 25
write_hertzian_dipole_far_field 41, 49

X

x_order 70

Y

y_order 70

Concept Index

A

Absorbing layers 15
Arrays 12

B

Boolean values 11

C

Check mode 9
Comments, inserting 12
Compiling Angora 7
‘config_all.cfg’ 14
Configuration file, checking for errors 9
Configuration file, template 14
Configuration format 10
Configuration variables 14
Configuration variables, assigning 10

Configuration variables, value types ... 10
Configuration, auto-saving ... 88
Configuring Angora simulations ... 14
Courant factor ... 15
CPML ... 15

D

Dimensions, grid ... 15
Directories, input ... 85
Directories, output ... 85
Directories, specifying ... 85
Documentation, building ... 8
Downloading Angora ... 6
Dynamic range ... 17

E

Executing Angora ... 9

F

Floating-point values ... 10
Focused beams ... 67
Focused Hermite-Gaussian beams ... 67
Focused laser beams ... 67
Focused laser modes ... 67

G

Gaussian waveforms (differentiated), defining ... 36
Gaussian waveforms (modulated), defining ... 37
Gaussian waveforms, defining ... 35
Grid size ... 15
Grid spacing ... 15
Grid termination ... 15
Ground planes, placing ... 33
Groups ... 11

H

Hertzian sources, placing ... 38

I

Imaging ... 53
Imaging, HDF5 file content ... 61
Incident beams ... 62
Including other configuration files ... 13
Infinitesimal sources, placing ... 38
Installing Angora ... 7
Integer values ... 10

L

libconfig ... 10
Lists ... 12
Log, keeping ... 85
Logging ... 85

M

Material files, reading from ... 29
Materials, defining ... 21
Maximum field value ... 17
MPI support ... 9
MPI support, enabling ... 8
Multiple runs ... 86
Multiple simulations ... 86

N

Near-field-to-far-field transformer (NFFFT) ... 40
Near-field-to-far-field transformer (NFFFT), phasor domain ... 45
Near-field-to-far-field transformer (NFFFT), phasor domain, HDF5 file content ... 52
Near-field-to-far-field transformer (NFFFT), time domain ... 40
Near-field-to-far-field transformer (NFFFT), time domain, HDF5 file content ... 44

O

Objects, placing ... 23
Optical imaging ... 53
Optical imaging, HDF5 file content ... 61
Origin, global ... 16
Origin, grid ... 16

P

Parallelization ... 9
Parallelization, enabling ... 8
Paths, input ... 85
Paths, output ... 85
Paths, specifying ... 85
PEC planes, placing ... 33
Perfectly-matched layers ... 15
Planar layers, placing ... 24
Plane waves ... 63
PML ... 15
Point sources, placing ... 38

R

Random materials, placing ... 25
Recording ... 73
Recording, cross-section ... 74
Recording, cross-section, file format ... 77
Recording, field-value ... 82
Recording, field-value, HDF5 file format ... 84
Recording, line ... 78
Recording, line, file format ... 81

Recording, movie 74
Recording, movie, file format 77
Rectangular boxes, defining 19
Running Angora 9

S

Shapes, defining 18
Simulations, configuring 14
Simulations, parallelizing 9
Spheres, defining 20
Stability 15
Step size, spatial 15
Step size, temporal 15
Stratification, defining 24
String values 11

T

Time steps, number of 16
Total-field/scattered-field (TF/SF) boundary ... 62
Total-field/scattered-field (TF/SF) boundary, focused laser beams 67
Total-field/scattered-field (TF/SF) boundary, plane waves 63

W

Waveforms, defining 34