Computer Physics Communications ••• (••••) •••–•••

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method ☆

Ardavan F. Oskooi [a,c,*], David Roundy [b], Mihai Ibanescu [a,c,d], Peter Bermel [c], J.D. Joannopoulos [a,c,d], Steven G. Johnson [a,c,e,**]

[a] Center for Materials Science & Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, United States
[b] Department of Physics, Oregon State University, Corvallis, OR 97331, United States
[c] Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139, United States
[d] Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, United States
[e] Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, United States

ARTICLE INFO

Article history:
Received 2 October 2009
Received in revised form 13 November 2009
Accepted 17 November 2009
Available online xxxx

PACS:
02.70.Bf
82.20.Wt
03.50.De
87.64.Aa

Keywords:
Computational electromagnetism
FDTD
Maxwell solver

ABSTRACT

This paper describes Meep, a popular free implementation of the finite-difference time-domain (FDTD) method for simulating electromagnetism. In particular, we focus on aspects of implementing a full-featured FDTD package that go beyond standard textbook descriptions of the algorithm, or ways in which Meep differs from typical FDTD implementations. These include pervasive interpolation and accurate modeling of subpixel features, advanced signal processing, support for nonlinear materials via Padé approximants, and flexible scripting capabilities.

Program summary

Program title: Meep
Catalogue identifier: AEFU_v1_0
Program summary URL:: http://cpc.cs.qub.ac.uk/summaries/AEFU_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: GNU GPL
No. of lines in distributed program, including test data, etc.: 151 821
No. of bytes in distributed program, including test data, etc.: 1 925 774
Distribution format: tar.gz
Programming language: C++
Computer: Any computer with a Unix-like system and a C++ compiler; optionally exploits additional free software packages: GNU Guile [1], libctl interface library [2], HDF5 [3], MPI message-passing interface [4], and Harminv filter-diagonalization [5]. Developed on 2.8 GHz Intel Core 2 Duo.
Operating system: Any Unix-like system; developed under Debian GNU/Linux 5.0.2.
RAM: Problem dependent (roughly 100 bytes per pixel/voxel)
Classification: 10
External routines: Optionally exploits additional free software packages: GNU Guile [1], libctl interface library [2], HDF5 [3], MPI message-passing interface [4], and Harminv filter-diagonalization [5] (which requires LAPACK and BLAS linear-algebra software [6]).
Nature of problem: Classical electrodynamics
Solution method: Finite-difference time-domain (FDTD) method
Running time: Problem dependent (typically about 10 ns per pixel per timestep)
References:
[1] GNU Guile, http://www.gnu.org/software/guile
[2] Libctl, http://ab-initio.mit.edu/libctl
[3] M. Folk, R.E. McGrath, N. Yeager, HDF: An update and future directions, in: Proc. 1999 Geoscience and Remote Sensing Symposium (IGARSS), Hamburg, Germany, vol. 1, IEEE Press, 1999, pp. 273–275.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author at: Center for Materials Science & Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, United States.
** Principal corresponding author.
E-mail addresses: ardavan@mit.edu (A.F. Oskooi), roundyd@physics.oregonstate.edu (D. Roundy), michel@alum.mit.edu (M. Ibanescu), bermel@mit.edu (P. Bermel), joannop@mit.edu (J.D. Joannopoulos), stevenj@math.mit.edu (S.G. Johnson).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2009.11.008

[4] T.M. Forum, MPI: A Message Passing Interface, in: Supercomputing 93, Portland, OR, 1993, pp. 878–883.
[5] Harminv, http://ab-initio.mit.edu/harminv.
[6] LAPACK, http://www.netlib.org/lapack/lug.

1. Introduction

One of the most common computational tools in classical electromagnetism is the finite-difference time-domain (FDTD) algorithm, which divides space and time into a regular grid and simulates the time evolution of Maxwell's equations [1–5]. This paper describes our free, open-source implementation of the FDTD algorithm: *Meep* (an acronym for *MIT Electromagnetic Equation Propagation*), available online at http://ab-initio.mit.edu/meep. Meep is full-featured, including, for example: arbitrary anisotropic, nonlinear, and dispersive electric and magnetic media; a variety of boundary conditions including symmetries and perfectly matched layers (PML); distributed-memory parallelism; Cartesian (1d/2d/3d) and cylindrical coordinates; and flexible output and field computations. It also includes some unusual features, such as advanced signal processing to analyze resonant modes, accurate subpixel averaging, a frequency-domain solver that exploits the time-domain code, complete scriptability, and integrated optimization facilities. Here, rather than review the well-known FDTD algorithm itself (which is thoroughly covered elsewhere), we focus on the particular design decisions that went into the development of Meep whose motivation may not be apparent from textbook FDTD descriptions, the tension between abstraction and performance in FDTD implementations, and the unique or unusual features of our software.

Why implement yet another FDTD program? Literally dozens of commercial FDTD software packages are available for purchase, but the needs of research often demand the flexibility provided by access to the source code (and relaxed licensing constraints to speed porting to new clusters and supercomputers). Our interactions with other photonics researchers suggest that many groups end up developing their own FDTD code to serve their needs (our own groups have used at least three distinct in-house FDTD implementations over the past 15 years), a duplication of effort that seems wasteful. Most of these are not released to the public, and the handful of other free-software FDTD programs that could be downloaded when Meep was first released in 2006 were not nearly full-featured enough for our purposes. Since then, Meep has been cited in over 100 journal publications and has been downloaded over 10,000 times, reaffirming the demand for such a package.

FDTD algorithms are, of course, only one of many numerical tools that have been developed in computational electromagnetism, and may perhaps seem primitive in light of other sophisticated techniques, such as finite-element methods (FEMs) with high-order accuracy and/or adaptive unstructured meshes [6–8], or even radically different approaches such as boundary-element methods (BEMs) that discretize only interfaces between homogeneous materials rather than volumes [9–12]. Each tool, of course, has its strengths and weaknesses, and we do not believe that any single one is a panacea. The nonuniform unstructured grids of FEMs, for example, have compelling advantages for metallic structures where micrometer wavelengths may be paired with nanometer skin depths. On the other hand, this flexibility comes at a price of substantial software complexity, which may not be worthwhile for dielectric devices at infrared wavelengths (such as in integrated optics or fibers) where the refractive index (and hence the typical resolution required) varies by less than a factor of four between materials, while small features such as surface roughness can be accurately handled by perturbative techniques [13]. BEMs, based on integral-equation formulations of electromagnetism, are especially powerful for scattering problems involving small objects in a large volume, since the volume need not be discretized and no artificial "absorbing boundaries" are needed. On the other hand, BEMs have a number of limitations: they may still require artificial absorbers for interfaces extending to infinity (such as input/output waveguides) [14]; any change to the Green's function (such as introduction of anisotropic materials, imposition of periodic or symmetry boundary conditions, or a switch from three to two dimensions) requires re-implementation of large portions of the software (e.g. singular panel integrations and fast solvers) rather than purely local changes as in FDTD or FEM; continuously varying (as opposed to piecewise-constant) materials are inefficient; and solution in the time domain (rather than frequency domain, which is inadequate for nonlinear or active systems in which frequency is not conserved) with BEM requires an expensive solver that is nonlocal in time as well as in space [11]. And then, of course, there are specialized tools that solve only a particular type of electromagnetic problem, such as our own MPB software that only computes eigenmodes (e.g. waveguide modes) [15], which are powerful and robust within their domain but are not a substitute for a general-purpose Maxwell simulation. FDTD has the advantages of simplicity, generality, and robustness: it is straightforward to implement the full time-dependent Maxwell equations for nearly arbitrary materials (including nonlinear, anisotropic, dispersive, and time-varying materials) and a wide variety of boundary conditions, one can quickly experiment with new physics coupled to Maxwell's equations (such as populations of excited atoms for lasing [16–20]), and the algorithm is easily parallelized to run on clusters or supercomputers. This simplicity is especially attractive to researchers whose primary concern is investigating new interactions of physical processes, and for whom programmer time and the training of new students is far more expensive than computer time.

The starting point for any FDTD solver is the time-derivative parts of Maxwell's equations, which in their simplest form can be written:

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E} - \mathbf{J}_B, \tag{1}$$

$$\frac{\partial \mathbf{D}}{\partial t} = +\nabla \times \mathbf{H} - \mathbf{J}, \tag{2}$$

where (respectively) $\mathbf{E}$ and $\mathbf{H}$ are the macroscopic electric and magnetic fields, $\mathbf{D}$ and $\mathbf{B}$ are the electric displacement and magnetic induction fields [21], $\mathbf{J}$ is the electric-charge current density, and $\mathbf{J}_B$ is a fictitious magnetic-charge current density (sometimes convenient in calculations, e.g. for magnetic-dipole sources). In time-domain calculations, one typically solves the initial-value problem where the fields and currents are zero for $t < 0$, and then nonzero values evolve in response to some currents $\mathbf{J}(\mathbf{x}, t)$ and/or $\mathbf{J}_B(\mathbf{x}, t)$. (In contrast, a *frequency-domain* solver assumes a time dependence of $e^{-i\omega t}$ for all currents and fields, and solves the resulting linear equations for the steady-state response or eigenmodes [22, Appendix D].) We prefer to use dimensionless units $\varepsilon_0 = \mu_0 = c = 1$. From our perspective, this choice emphasizes both the scale invariance of Maxwell's equations [22, Chapter 2] and also the fact that the most meaningful quantities to calculate are almost always dimensionless ratios (such as scattered

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

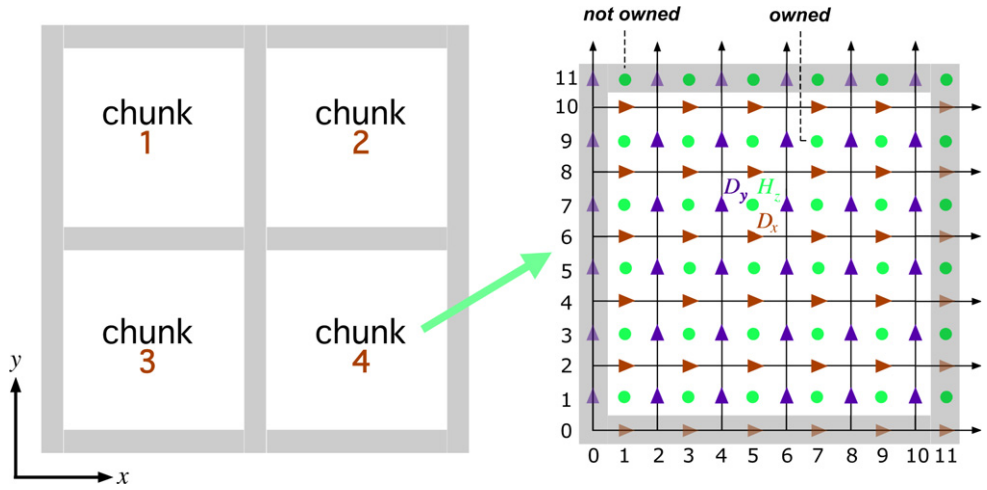

Fig. 1. The computational cell is divided into chunks (left) that have a one-pixel overlap (gray regions). Each chunk (right) represents a portion of the Yee grid, partitioned into *owned* points (chunk interior) and *not-owned* points (gray regions around the chunk edges) that are determined from other chunks and/or via boundary conditions. Every point in the interior of the computational cell is owned by exactly one chunk, the chunk responsible for timestepping that point.

power over incident power, or wavelength over some characteristic lengthscale). The user can pick any desired unit of distance a (either an SI unit such as $a = 1$ μm or some typical lengthscale of a given problem), and all distances are given in units of a, all times in units of a/c, and all frequencies in units of c/a. In a linear dispersionless medium, the constituent relations are $\mathbf{D} = \varepsilon\mathbf{E}$ and $\mathbf{B} = \mu\mathbf{H}$, where ε and μ are the relative permittivity and permeability (possibly tensors); the case of nonlinear and/or dispersive media (including conductivities) is discussed further in Section 4.

The remaining paper is organized as follows. In Section 2, we discuss the discretization and coordinate system; in addition to the standard Yee discretization [1], this raises the question of how exactly the grid is described and divided into "chunks" for parallelization, PML, and other purposes. Section 3 describes a central principle of Meep's design, *pervasive interpolation* providing (as much as possible) the illusion of continuity in the specification of sources, materials, outputs, and so on. This led to the development of several techniques unique to Meep, such as a scheme for subpixel material averaging designed to eliminate the first-order error usually associated with averaging techniques or stairstepping of interfaces. In Section 4, we describe and motivate our techniques for implementing nonlinear and dispersive materials, including a slightly unusual method to implement nonlinear materials using a Padé approximant that eliminates the need to solve cubic equations for every pixel. Section 5 describes how typical computations are performed in Meep, such as memory-efficient transmission spectra or sophisticated analysis of resonant modes via harmonic inversion. This section also describes how we have adapted the time-domain code, almost without modification, to solve frequency-domain problems with much faster convergence to the steady-state response than merely time-stepping. The user interface of Meep is discussed in Section 6, explaining the considerations that led us to a scripting interface (rather than a GUI or CAD interface). Section 7 describes some of the tradeoffs between performance and generality in this type of code and the specific compromises chosen in Meep. Finally, we make some concluding remarks in Section 8.

2. Grids and boundary conditions

The starting point for the FDTD algorithm is the discretization of space and time into a grid. In particular, Meep uses the standard *Yee grid* discretization which staggers the electric and magnetic fields in time and in space, with each field component sampled at different spatial locations offset by half a pixel, allowing the time and space derivatives to be formulated as center-difference approximations [23]. This much is common to nearly every FDTD implementation and is described in detail elsewhere [1]. In order to parallelize Meep, efficiently support simulations with symmetries, and to efficiently store auxiliary fields only in certain regions (for PML absorbing layers), Meep further divides the grid into *chunks* that are joined together into an arbitrary topology via boundary conditions. (In the future, different chunks may have different resolutions to implement a nonuniform grid [24–27].) Furthermore, we distinguish two coordinate systems: one consisting of integer coordinates on the Yee grid, and one of continuous coordinates in "physical" space that are interpolated as necessary onto the grid (see Section 3). This section describes those concepts as they are implemented in Meep, as they form a foundation for the remaining sections and the overall design of the Meep software.

2.1. Coordinates and grids

The two spatial coordinate systems in Meep are described by the `vec`, a continuous vector in $\mathbb{R}^d$ (in d dimensions), and the `ivec`, an integer-valued vector in $\mathbb{Z}^d$ describing locations on the Yee grid. If $\mathbf{n}$ is an `ivec`, the corresponding `vec` is given by $0.5\Delta x\mathbf{n}$, where Δx is the spatial resolution (the same along x, y, and z)—that is, the integer coordinates in an `ivec` correspond to *half*-pixels, as shown in the right panel of Fig. 1. This is to represent locations on the spatial Yee grid, which offsets different field components in space by half a pixel as shown (in 2d) in the right panel of Fig. 1. In 3d, the E_x and D_x components are sampled at `ivecs` $(2\ell+1, 2m, 2n)$, E_y and D_y are sampled at `ivecs` $(2\ell, 2m+1, 2n)$, and so on; H_x and B_x are sampled at `ivecs` $(2\ell, 2m+1, 2n+1)$, H_y and B_y are sampled at `ivecs` $(2\ell+1, 2m, 2n+1)$, and so on. In addition to these grids for the different field components, we also occasionally refer to the *centered* grid, at odd `ivecs` $(2\ell+1, 2m+1, 2n+1)$ corresponding to the "center" of each pixel. (The origin of the coordinate systems is an arbitrary `ivec` that can be set by the user, but is typically the center of the computational volume.) The philosophy of Meep, as described in Section 3, is that as much as possible the user should be concerned only with continuous physical coordinates (`vecs`), and the interpolation/discretization onto `ivecs` occurs internally as transparently as possible.

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

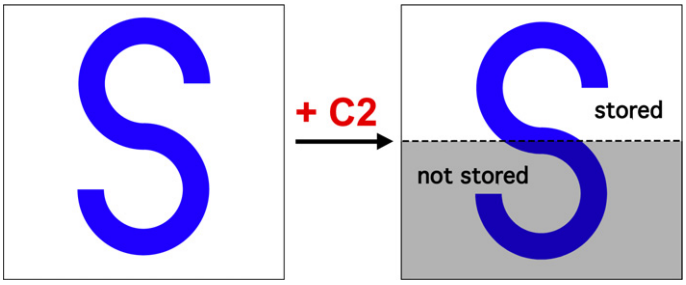

Fig. 2. Meep can exploit mirror and rotational symmetries, such as the 180-degree (C_2) rotational symmetry of the S-shaped structure in this schematic example. Although Meep maintains the illusion that the entire structure is stored and simulated, internally only half of the structure is stored (as shown at right), and the other half is inferred by rotation. The rotation gives a boundary condition for the not-owned grid points along the dashed line.

2.2. Grid chunks and owned points

An FDTD simulation must occur within a finite volume of space, the *computational cell*, terminated with some boundary conditions and possibly by absorbing PML regions as described below. This (rectilinear) computational cell, however, is further subdivided into convex rectilinear *chunks*. On a parallel computer, for example, different chunks may be stored at different processors. In order to simplify the calculations for each chunk, we employ the common technique of padding each chunk with extra "boundary" pixels that store the boundary values [28] (shown as gray regions in Fig. 1)—this means that the chunks are *overlapping* in the interior of the computational cell, where the overlaps require communication to synchronize the values.

More precisely, the grid points in each chunk are partitioned into *owned* and *not-owned* points. The *not-owned* points are determined by communication with other chunks and/or by boundary conditions. The *owned* points are time-stepped within the chunk, independently of the other chunks (and possibly in parallel), and *every grid point inside the computational cell is owned by exactly one chunk*.

The question then arises: how do we decide which points within the chunk are owned? In order for a grid point to be owned, the chunk must contain all the information necessary for timestepping that point (once the not-owned points have been communicated). For example, for a D_y point $(2\ell, 2m+1, 2n)$ to be owned, the H_z points at $(2\ell \pm 1, 2m+1, 2n)$ must both be in the chunk in order to compute $\nabla \times \mathbf{H}$ for timestepping $\mathbf{D}$ at that point. This means that the D_y points along the left (minimum-x) edge of the chunk (as shown in the right panel of Fig. 1) *cannot* be owned: there is no H_z point to the left of it. An additional dependency is imposed by the case of anisotropic media: if there is an ε_{xy} coupling E_x to D_y, then updating E_x at $(2\ell+1, 2m, 2n)$ requires the four D_y values at $(2\ell+1\pm1, 2m\pm1, 2n)$ (these are the surrounding D_y values, as seen in the right panel of Fig. 1). This means that the E_x (and D_x) points along the *right* (maximum-x) edge of the chunk (as shown in the right panel of Fig. 1) cannot be owned either: there is no D_y point to the right of it. Similarly for $\nabla \times \mathbf{D}$ and anisotropic μ.

All of these considerations result in the shaded-gray region of Fig. 1 (right) being not-owned. That is, if the chunk intersects $k+1$ pixels along a given direction starting at an `ivec` coordinate of 0 (e.g. $k=5$ in Fig. 1), the endpoint `ivec` coordinates 0 and $2k+1$ are not-owned and the interior coordinates from 1 to $2k$ (inclusive) are owned.

2.3. Boundary conditions and symmetries

All of the not-owned points in a chunk must be determined by boundary conditions of some sort. The simplest boundary conditions are when the not-owned points are owned by some other chunk, in which case the values are simply copied from that chunk (possibly requiring communication on a multiprocessor system) each time they are updated. In order to minimize communications overhead, all communications between two chunks are batched into a single message (by copying the relevant not-owned points to/from a contiguous buffer) rather than sending one message per point to be copied.

At the edges of the computational cell, some user-selected boundary condition must be imposed. For example, one can use perfect electric or magnetic conductors where the relevant electric/magnetic-field components are set to zero at the boundaries. One can also use Bloch-periodic boundary conditions, where the fields on one side of the computational cell are copied from the other side of the computational cell, optionally multiplied by a complex phase factor $e^{ik_i\Lambda_i}$ where k_i is the propagation constant in the ith direction, and Λ_i is the length of the computational cell in the same direction. Meep does *not* implement any absorbing boundary conditions—absorbing boundaries are, instead, handled by an artificial material, perfectly matched layers (PML), placed adjacent to the boundaries [1].

Bloch-periodic boundary conditions are useful in periodic systems [22], but this is only one example of a useful symmetry that may be exploited via boundary conditions. One may also have mirror and rotational symmetries. For example, if the materials and the field sources have a mirror symmetry, one can cut the computational costs in two by storing chunks only in half the computational cell and applying mirror boundary conditions to obtain the not-owned pixels adjacent to the mirror plane. As a more unusual example, consider an S-shaped structure as in Fig. 2, which has no mirror symmetry but is symmetric under 180-degree rotation, called C_2 symmetry [29]. Meep can exploit this case as well (assuming the current sources have the same symmetry), storing only half of the computational cell as in Fig. 2 and inferring the not-owned values along the dashed line by a 180-degree rotation. (In the simple case where the stored region is a single chunk, this means that the not-owned points are determined by owned points in the same chunk, requiring copies, possibly with sign flips.) Depending on the sources, of course, the fields can be even or odd under mirror flips or C_2 rotations [22], so the user can specify an additional sign flip for the transformation of the vector fields (and pseudovector $\mathbf{H}$ and $\mathbf{B}$ fields, which incur an additional sign flip under mirror reflections [21,22]). Meep also supports four-fold rotation symmetry (C_4), where the field can be multiplied by factors of 1, i, -1, or $-i$ under each 90-degree rotation [29]. (Other rotations, such as three-fold or six-fold, are not supported because they do not preserve the Cartesian Yee grid.) In 2d, the xy-plane is itself a mirror plane (unless in the presence of anisotropic materials) and the symmetry decouples TE modes (with fields E_x, E_y, and H_z) from TM modes (H_x, H_y, and E_z) [22]; in this case Meep only allocates those fields for which the corresponding sources are present.

A central principle of Meep is that symmetry optimizations be transparent to the user once the desired symmetries are speci-

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

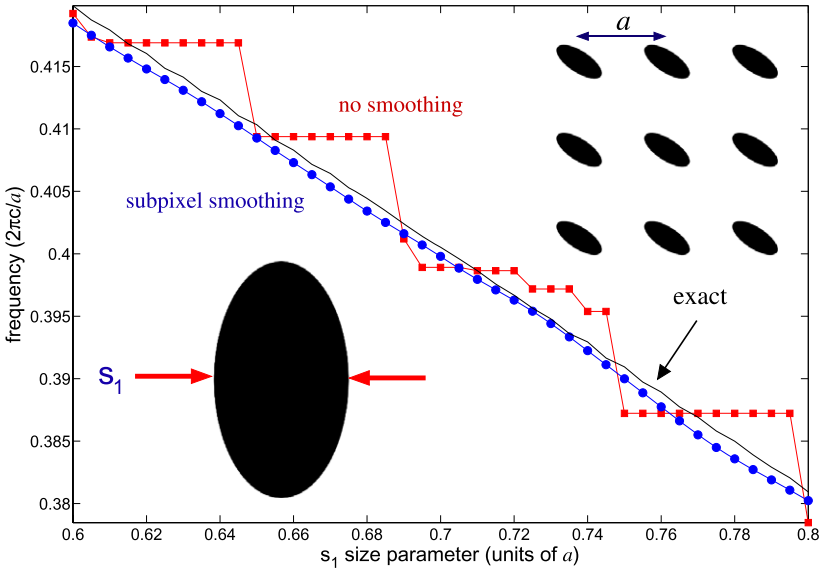

Fig. 3. A key principle of Meep is that continuously varying inputs yield continuously varying outputs. Here, an eigenfrequency of a photonic crystal varies continuously with the eccentricity of a dielectric rod, accomplished by subpixel smoothing of the material parameters, whereas the nonsmoothed result is "stairstepped". Specifically, the plot shows a TE eigenfrequency of 2d square lattice (period a) of dielectric ellipses ($\varepsilon = 12$) in air versus one semi-axis diameter of the ellipse (in gradations of $0.005a$) for no smoothing (red squares, resolution of 20 pixels/a), subpixel smoothing (blue circles, resolution of 20 pixels/a) and "exact" results (black line, no smoothing at resolution of 200 pixels/a). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

fied. Meep maintains the illusion that the entire computational cell is computed—for example, the fields in the entire computational cell can still be queried or exported to a file, flux planes and similar computations can still extend anywhere within the computational cell, and so on. The fields in the nonstored regions are simply computed behind the scenes (without ever allocating memory for them) by transforming the stored chunks as needed. A key enabling factor for maintaining this illusion efficiently is the *loop-in-chunks* abstraction employed by the Meep code, described in Section 7.

Meep also supports continuous rotational symmetry around a given axis, where the structure is invariant under rotations and the fields transform as $e^{im\phi}$ for some m [22], but this is implemented separately by providing the option to simulate Maxwell's equations in the (r, z)-plane with cylindrical coordinates, for which operators like $\nabla\times$ change form.

3. Interpolation and the illusion of continuity

A core design philosophy of Meep is to provide the illusion of continuous space and time, masking the underlying discretization from the user as much as possible. There are two components to this approach: the input and the outputs. Continuously varying inputs, such as the geometry, materials, and the source currents, lead to continuously varying outputs, as in the example of Fig. 3. Similarly, the value of any field (or any function of the fields) can be output at any point in space or integrated over any region. Furthermore, the effects of these inputs and the resulting outputs must converge as quickly as possible to the exact solution as the resolution increases. In this section, we discuss how this illusion of continuity is implemented for field outputs, current inputs, and geometry/materials.

Any field component (or any combinations such as flux, energy, and user-defined functions) can be evaluated at any point in space. In general, this requires interpolation from the Yee grid. Since the underlying FDTD center-difference algorithm has second-order accuracy, we linearly interpolate fields as needed (which also has second-order accuracy for smooth functions). Similarly, we provide an interface to integrate any function of the fields over any convex rectilinear region (boxes, planes, or lines), and the integral is computed by integrating the linear interpolation of the fields within the integration region. This is straightforward, but there are two subtleties due to the staggered Yee grid. First, computation of quantities like $\mathbf{E}\times\mathbf{H}$ that mix different field components requires an additional interpolation: first, the fields are interpolated onto the centered grid (Section 2), then the integrand is computed, and then the linear interpolation of the integrand is integrated over the specified region. Second, the computation of quantities like $\mathbf{E}\times\mathbf{H}$ mixes two fields that are stored at different *times*: $\mathbf{H}$ is stored at times $(n-0.5)\Delta t$, while $\mathbf{E}$ is stored at times $n\Delta t$ [1]. Simply using these time-offset fields together is only first-order accurate. If second-order accuracy is desired, Meep provides the option to temporarily synchronize the electric and magnetic fields: the magnetic fields are saved to a backup array, stepped by Δt, and they are averaged with the backup array to obtain the magnetic fields at $n\Delta t$ with $O(\Delta t^2)$ accuracy. (The fields are restored from backup before resuming timestepping.) This restores second-order accuracy at the expense of an extra half a timestep's computation, which is usually negligible because such field computations are rarely required at every timestep of a simulation—see Section 5 for how Meep performs typical transmission simulations and other calculations efficiently.

The conceptually reversed process is required for specifying sources: the current density is specified at some point (for dipole sources) or in some region (for distributed current sources) in continuous space, and then must be *restricted* to a corresponding current source on the Yee grid. Meep performs this restriction using exactly the same code (the loop-in-chunks abstraction of Section 7) and the same weights as the interpolation procedure above. Mathematically, we are exploiting a well-known concept (originating in multigrid methods) that restriction can be defined as the *transpose* of interpolation [30]. This is illustrated by a 2d example in Fig. 4. Suppose that the bilinear interpolation f (blue) of four grid points (red) is $f = 0.32f_1 + 0.48f_2 + 0.08f_3 + 0.12f_4$, which can be viewed as multiplying a vector of those fields by the row-vector $[0.32, 0.48, 0.08, 0.12]$. Conversely, if we place a point-dipole current source J (blue) at the same point, it is restricted on the grid (red) to values $J_1 = 0.32J$, $J_2 = 0.48J$, $J_3 = 0.08J$, and $J_4 = 0.12J$ as shown in Fig. 4, corresponding to multiplying J by

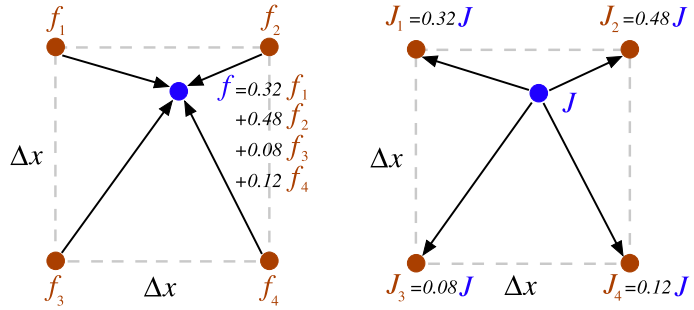

Fig. 4. Left: a bilinear interpolation of values $f_{1,2,3,4}$ on the grid (red) to the value f at an arbitrary point. Right: the reverse process is *restriction*, taking a value J at an arbitrary point (e.g. a current source) and converting into values on the grid. Restriction can be viewed as the transpose of interpolation and uses the same coefficients. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

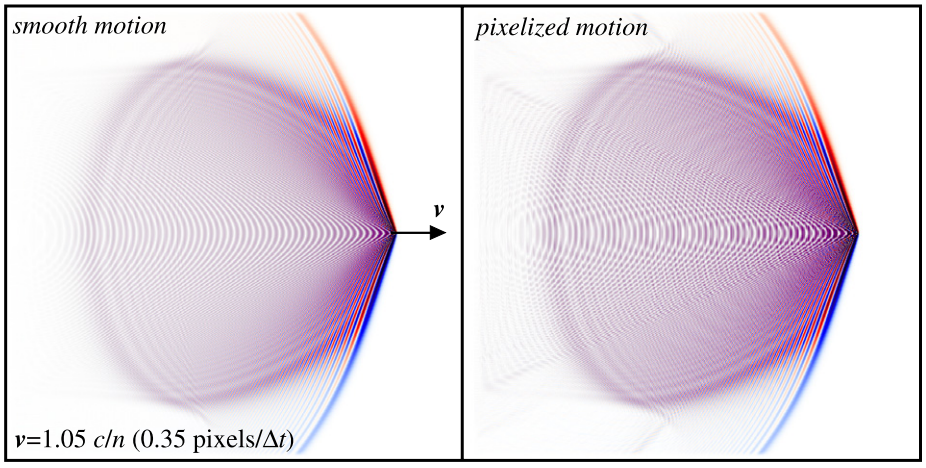

Fig. 5. Cerenkov radiation emitted by a point charge moving at a speed $v = 1.05c/n$ exceeding the phase velocity of light in a homogeneous medium of index $n = 1.5$. Thanks to Meep's interpolation (or technically *restriction*), the smooth motion of the source current (left panel) can be expressed as continuously varying currents on the grid, whereas the nonsmooth pixelized motion (no interpolation) (right panel) reveals high-frequency numerical artifacts of the discretization (counter-propagating wavefronts behind the moving charge).

the column vector $[0.32, 0.48, 0.08, 0.12]^T$.[1] Such a restriction has the property of preserving the sum (integral) of the currents, and typically leads to second-order convergence of the resulting fields as the resolution increases (see below). An example of the utility of this continuous restriction process is shown in Fig. 5 via the phenomenon of Cerenkov radiation [31]: a point charge q moving at a constant velocity $\mathbf{v}$ with a magnitude $1.05c/n$ exceeding the phase velocity c/n in the medium emits a shockwave-like radiation pattern, and this can be directly modeled in Meep by a continuously moving current source $\mathbf{J} = -\mathbf{v}q\delta(\mathbf{x} - \mathbf{v}t)$ [32]. In contrast, pixelizing the motion into discrete jumps to the nearest grid point leads to visible numerical artifacts in the radiation, as seen in the right panel of Fig. 5.

All of the second-order accuracy of FDTD and the above interpolations is generally spoiled to only first-order, however, if one directly discretizes a discontinuous material boundary [33,35]. Moreover, directly discretizing a discontinuity in ε or μ leads to "stairstepped" interfaces that can only be varied in discrete jumps of one pixel at a time. Both of these problems are solved in Meep by using an appropriate subpixel smoothing of ε and μ: before discretizing, discontinuities are smoothed into continuous transitions over a distance of one pixel Δx, using a carefully designed averaging procedure. Any subpixel smoothing technique will achieve the goal of continuously varying results as the geometry is continuously varied. In the case of Meep this is illustrated by Fig. 3: in a 2d photonic crystal (square lattice of dielectric rods), the lowest TE-polarization eigenfrequency (computed as in Section 5) varies continuously with the eccentricity of the elliptical rods for subpixel averaging, whereas the nonaveraged discontinuous discretization produces a stairstepped discontinuous eigenfrequency. On the other hand, most subpixel smoothing techniques will not increase the accuracy of FDTD—on the contrary, smoothing discontinuous interfaces changes the structure, and generally introduces *additional* error into the simulation [33]. In order to design an accurate smoothing technique, we exploited recent results in perturbation theory that show how a particular subpixel smoothing can be chosen to yield zero first-order error [13,33,34,36]. The results are shown in Figs. 6 and 7: for both computation of the eigenfrequencies (of an anisotropic photonic crystal) in Fig. 6 and the scattering loss from a bump on a strip waveguide in Fig. 7, the errors in Meep's results decrease quadratically $[O(\Delta x^2)]$, whereas doing no averaging leads to erratic linear convergence $[O(\Delta x)]$. Furthermore, Fig. 6 compares to other subpixel-averaging schemes, including the obvious strategy of simply averaging ε within each pixel [37], and shows that they lead to first-order convergence no better than no averaging at all.

The subpixel averaging is discussed in more detail elsewhere [33,34,36], so we only briefly summarize it here. In order for the smoothing to yield zero first-order perturbation, the smoothing scheme must be anisotropic. Even if the initial interface is between isotropic materials, one obtains a tensor ε (or μ) which uses the mean ε for fields parallel to the interface and the harmonic mean (inverse of mean of ε^{-1}) for fields perpendicular to the interface—this was initially proposed heuristically [38] and later shown to be

[1] Technically, for a dipole-current source given by a delta function with amplitude I, the corresponding current density is $J = I/\Delta x^d$ in d dimensions.

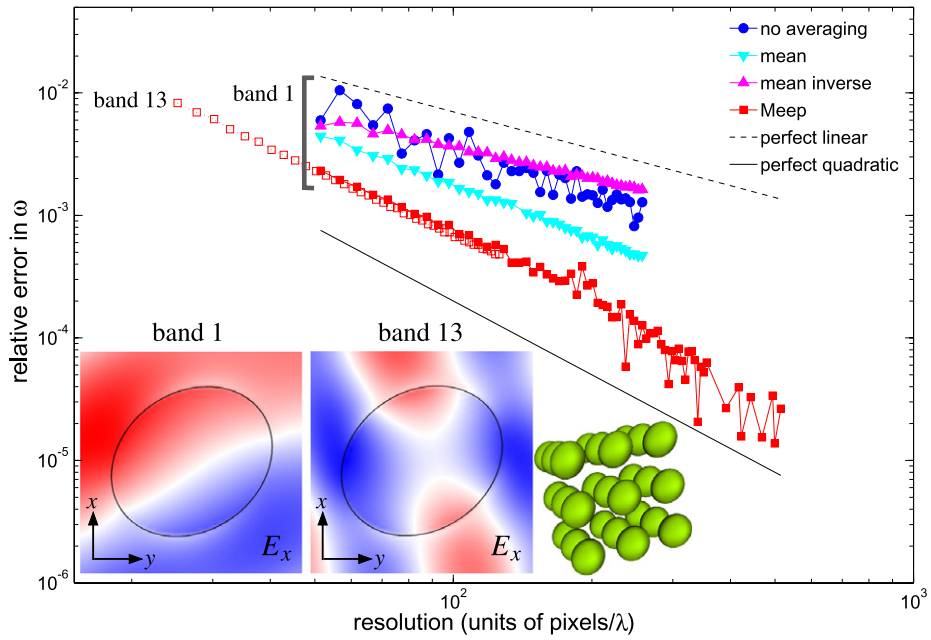

Fig. 6. Appropriate subpixel averaging can *increase* the accuracy of FDTD with discontinuous materials [33,34]. Here, relative error $\Delta\omega/\omega$ (comparing to the "exact" ω_0 from a planewave calculation [15]) for an eigenmode calculation (as in Section 5) for a cubic lattice (period a) of 3d anisotropic-ε ellipsoids (right inset) versus spatial resolution (units of pixels per vacuum wavelength λ), for a variety of subpixel smoothing techniques. Straight lines for perfect linear (black dashed) and perfect quadratic (black solid) convergence are shown for reference. Most curves are for the first eigenvalue band (left inset shows E_x in xy cross-section of unit cell), with vacuum wavelength $\lambda = 5.15a$. Hollow squares show Meep's method for band 13 (middle inset), with $\lambda = 2.52a$. Meep's method for bands 1 and 13 is shown for resolutions up to 100 pixels/a.

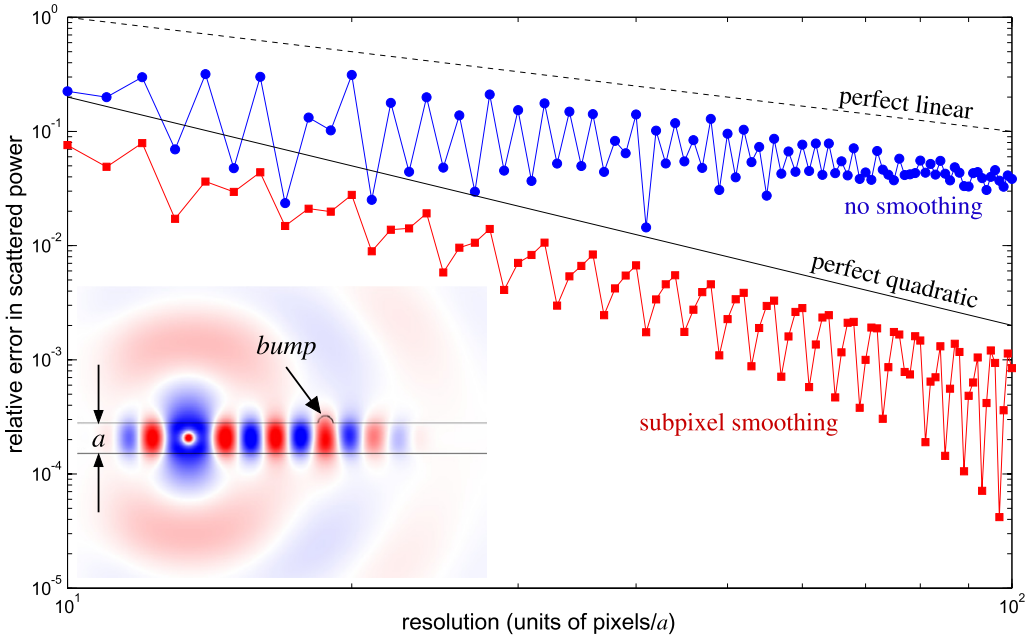

Fig. 7. The relative error in the scattered power from a small semicircular bump in a dielectric waveguide ($\varepsilon = 12$), excited by a point-dipole source in the waveguide (geometry and fields shown in inset), as a function of the computational resolution. Appropriate subpixel smoothing of the dielectric interfaces leads to roughly second-order [$O(\Delta x^2)$] convergence (red squares), whereas the unsmoothed structure has only first-order convergence (blue circles). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

justified via perturbation theory [13,33]. (If the initial materials are anisotropic, a more complicated formula is needed [34,36].) The key point is that, even if the physical structure consists entirely of isotropic materials, the discretized structure will use anisotropic materials. Stable simulation of anisotropic media requires an FDTD variant recently proposed in Ref. [39].

There are a few limitations to this subpixel averaging. First, the case of perfect metals requires a different approach [40,41] that is not yet implemented in Meep. Although Meep does not yet implement subpixel averaging for dispersive materials, there is numerical evidence that similar accuracy improvements are obtained in that case by the same technique [42], and we suspect that a similar derivation can be applied (using the unconjugated form of perturbation theory for the complex-symmetric Maxwell equations in reciprocal media with losses [43]). Second, once the smoothing eliminates the first-order error, the presence of sharp corners (associated with field singularities) introduce an error intermediate between first- and second-order [33], which we hope to address in

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

future work. Third, the fields directly *on* the interface are still at best first-order accurate even with subpixel smoothing—however, these localized errors are equivalent to currents that radiate zero power to first order [36,44]. The improved accuracy from smoothing is therefore obtained for fields evaluated off of the interface as in scattered flux integrated over a surface away from the interface (Fig. 7), for nonlocal properties like resonant frequencies and eigenfrequencies (Fig. 6), and for overall integrals of fields and energies [to which the interface contributes only $O(\Delta x)$ of the integration domain and hence first-order errors on the interface have a second-order effect].

4. Materials

Time-dependent methods for electromagnetism, given their generality, allow for the simulation of a broad range of material systems. Certain classes of materials, particularly active and nonlinear materials which do not conserve frequency, are ideally suited for modeling by such methods. Materials are represented in Maxwell's equations (1) and (2) via the relative permittivity $\varepsilon(\mathbf{x})$ and permeability $\mu(\mathbf{x})$ which in general depend on position, frequency (material dispersion) and the fields themselves (nonlinearities). Meep currently supports arbitrary anisotropic material tensors, anisotropic dispersive materials (Lorentz–Drude models and conductivities, both magnetic and electric), and nonlinear materials (both second- and third-order nonlinearities), which taken together permit investigations of a wide range of physical phenomena. The implementation of these materials in Meep is mostly based on standard techniques [1], so we will focus here on two places where Meep differs from the usual approach. For nonlinearities, we use a Padé approximant to avoid solving cubic equations at each step. For PML absorbing media in cylindrical coordinates, we only use a "quasi-PML" [46] based on a Cartesian PML, but explain why its performance is comparable to a true PML while requiring less computational effort.

4.1. Nonlinear materials

Optical nonlinearities arise when large field intensities induce changes in the local ε or μ to produce a number of interesting effects: temporal and spatial soliton propagation, optical bistability, self-focusing of optical beams, second- and third-harmonic generation, and many other effects [47,48]. Such materials are usually described by a power-series expansion of $\mathbf{D}$ in terms of $\mathbf{E}$ and various susceptibilities. In many common materials, or when considering phenomena in a sufficiently narrow bandwidth (such as the resonantly enhanced nonlinear effects [49] well-suited to FDTD calculations), these nonlinear susceptibilities can be accurately approximated via nondispersive (instantaneous) effects [50]. Meep supports instantaneous isotropic (or diagonal anisotropic) nonlinear effects of the form:

$$D_i - P_i = \varepsilon^{(1)} E_i + \chi_i^{(2)} E_i^2 + \chi_i^{(3)} |\mathbf{E}|^2 E_i, \tag{3}$$

where $\varepsilon^{(1)}$ represents all the linear nondispersive terms and P_i is a dispersive polarization $\mathbf{P} = \chi^{(1)}_{\text{dispersive}}(\omega)\mathbf{E}$ from dispersive materials such as Lorentz media [1]. (A similar equation relates $\mathbf{B}$ and $\mathbf{H}$.) Implementing this equation directly, however, would require one to solve a cubic equation at each time step [1, Section 9.6], since $\mathbf{D}$ is updated from $\nabla \times \mathbf{H}$ before updating $\mathbf{E}$ from $\mathbf{D}$.

However, Eq. (3) is merely a power-series approximation for the true material response, valid for sufficiently small field intensities, so it is not necessary to insist that it be solved exactly. Instead, we approximate the solution of Eq. (3) by a Padé approximant [51], which matches the "exact" cubic solution to high-order accuracy by the rational function:

$$E_i = \left[\frac{1 + (\frac{\chi^{(2)}}{[\varepsilon^{(1)}]^2}\tilde{D}_i) + 2(\frac{\chi^{(3)}}{[\varepsilon^{(1)}]^3}\|\tilde{\mathbf{D}}\|^2)}{1 + 2(\frac{\chi^{(2)}}{[\varepsilon^{(1)}]^2}\tilde{D}_i) + 3(\frac{\chi^{(3)}}{[\varepsilon^{(1)}]^3}\|\tilde{\mathbf{D}}\|^2)}\right]\left[\varepsilon^{(1)}\right]^{-1}\tilde{D}_i, \tag{4}$$

where $\tilde{D}_i = D_i - P_i$. For the case of isotropic $\varepsilon^{(1)}$ and $\chi^{(2)} = 0$, so that we have a purely Kerr ($\chi^{(3)}$) material, this matches the "exact" cubic E to $O(D^7)$ error. With $\chi^{(2)} \neq 0$, the error is $O(D^4)$.

For more complicated dispersive nonlinear media or for arbitrary anisotropy in $\chi^{(2)}$ or $\chi^{(3)}$, one approach that Meep may implement in the future is to incorporate the nonlinear terms in the auxiliary differential equations for a Lorentz medium [1].

4.2. Absorbing boundary layers: PML, pseudo-PML, and quasi-PML

A perfectly matched layer (PML) is an artificial absorbing medium that is commonly used to truncate computational grids for simulating wave equations (e.g. Maxwell's equations), and is designed to have the property that interfaces between the PML and adjacent media are reflectionless in the exact wave equation [1]. There are various interchangeable formulations of PML for FDTD methods [1], which are all equivalent to a coordinate stretching of Maxwell's equations into complex spatial coordinates; Meep implements a version of the uniaxial PML (UPML), expressing the PML as an effective dispersive anisotropic ε and μ [1]. Meep provides support for arbitrary user-specified PML absorption profiles (which have an important influence on reflections due to discretization error and other effects) for a given round-trip reflection (describing the strength of the PML in terms of the amplitude of light passing through the PML, reflecting off the edge of the computational cell, and propagating back) [45]. For the case of periodic media such as photonic crystals, the medium is not analytic and the premise of PML's reflectionless property is violated; in this case, a "PML" material overlapped with the photonic crystal is only a "pseudo-PML" that is reflectionless only in the limit of a sufficiently thick and gradual absorber, and control over the absorption profile is important [45].

For the radial direction in cylindrical coordinates, a true PML can be derived by coordinate-stretching, but it requires more storage and computational effort than the Cartesian UPML [52,53], as well as increasing code complexity. Instead, we chose to implement a *quasi-PML* [46], which simply consists of using the Cartesian UPML materials as an approximation for the true radial PML. This approximation becomes more and more accurate as the outer radius of the computational cell increases, because the implicit curvature of the PML region decreases with radius and approaches the Cartesian case. Furthermore, one must recall that *every* PML has reflections once space is discretized [1], which can be mitigated by gradually turning on the PML absorption over a finite-thickness PML layer. The quasi-PML approximation is likewise mitigated by the same gradual absorption profile, and the only question is that of the constant factor in the reflection convergence: how thick does the quasi-PML need to be to achieve low reflections, compared to a true PML? Fig. 8 shows that, for a typical calculation, the performance of the quasi-PML in cylindrical coordinates (left) is comparable to that of a true PML in Cartesian coordinates (right). Here, we plot a measure of the reflection from the PML as a function of the PML absorber length L, for a fixed round-trip reflection [45], using as a measure of the reflection the "field convergence" factor: the difference between the $\mathbf{E}$ field at a given point for simulations with PML absorber lengths L and $L+1$. The PML conductivity $\sigma(x)$ is turned on gradually as $(x/L)^d$ for $d = 1, 2, 3$, and it can be shown that this leads to reflections that decrease as $1/L^{2d+2}$ and field-convergence factors that decrease as $1/L^{2d+4}$ [45]. Precisely these decay rates are observed in Fig. 8, with similar constant coefficients. As the resolution is increased (approaching the exact wave equations), the constant coefficient in

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

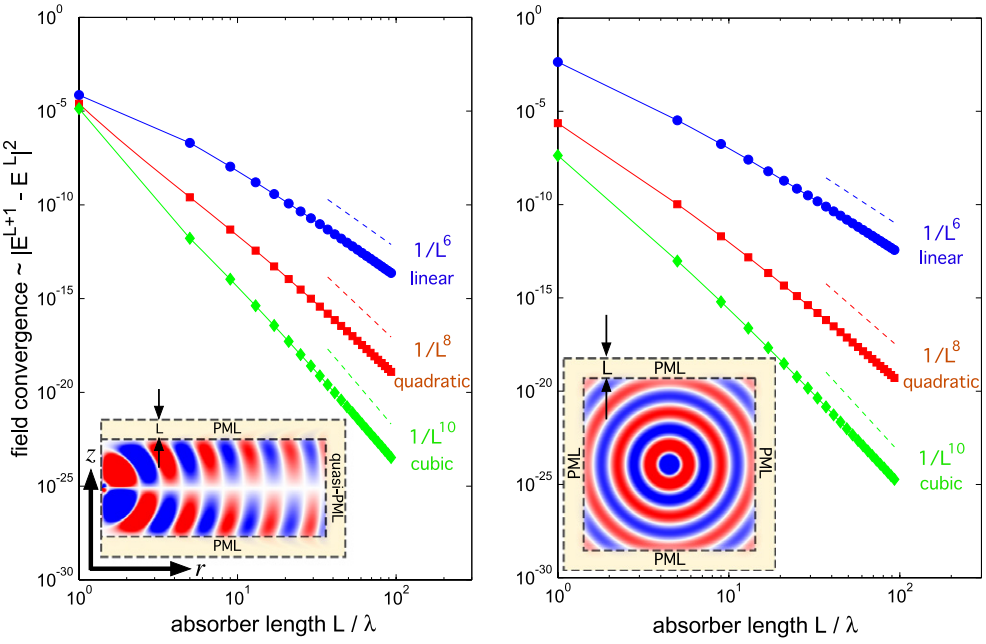

Fig. 8. The performance of a quasi-PML in the radial direction (cylindrical coordinates, left panel) at a resolution 20 pixels/λ is nearly equivalent to that of a true PML (in Cartesian coordinates, right panel). The plot shows the difference in the electric field E_z (insets) from a point source between simulations with PML thickness L and $L+1$, which is a simple proxy for the PML reflections [45]. The different curves are for PML conductivities that turn on as $(x/L)^d$ for $d = 1, 2, 3$ in the PML, leading to different rates of convergence of the reflection [45].

the Cartesian PML plot will decrease (approaching zero reflection), while the quasi-PML's constant coefficient will saturate at some minimum (corresponding to its finite reflectivity in the exact wave equation for a fixed L). This difference seems of little practical concern, however, because the reflection from a one-wavelength thick quasi-PML at a moderate resolution (20 pixels/λ) is already so low.

5. Enabling typical computations

Simulating Maxwell's equations in the time domain enables the investigation of problems inherently involving multiple frequencies, such as nonlinearities and active media. However, it is also well adapted to solving frequency domain problems since it can solve large bandwidths at once, for example analyzing resonant modes or computing transmission/reflection spectra. In this section, we describe techniques Meep uses to efficiently compute scattering spectra and resonant modes in the time domain. Furthermore, we describe how the time domain method can be adapted to a purely frequency domain solver while sharing almost all of the underlying code.

5.1. Computing flux spectra

A principle task of computational time-domain tools are investigations of transmission or scattering spectra from arbitrary structures, where one wants to compute the transmitted or scattered power in a particular direction as a function of the frequency of incident light. One can solve for the power at many frequencies in a single time-domain simulation by Fourier transforming the response to a short pulse. Specifically, for a given surface S, one wishes to compute the integral of the Poynting flux:

$$P(\omega) = \Re \iint_S \mathbf{E}_\omega(\mathbf{x})^* \times \mathbf{H}_\omega(\mathbf{x})\, d\mathbf{A}, \tag{5}$$

where $\mathbf{E}_\omega$ and $\mathbf{H}_\omega$ are the fields produced by a source at frequency ω, and $\Re$ denotes the real part of the expression. The basic idea, in time-domain, is to use a short-pulse source (covering a wide bandwidth including all frequencies of interest), and compute $\mathbf{E}_\omega$ and $\mathbf{H}_\omega$ from the Fourier transforms of $\mathbf{E}(t)$ and $\mathbf{H}(t)$. There are several different ways to compute these Fourier transforms. For example, one could store the electric and magnetic fields throughout S over all times and at the end of the simulation perform a discrete-time Fourier transform (DTFT) of the fields:

$$\mathbf{E}_\omega = \sum_n e^{i\omega n \Delta t} \mathbf{E}(n\Delta t)\Delta t, \tag{6}$$

for all frequencies (ω) of interest, possibly exploiting a fast Fourier transform (FFT) algorithm. Such an approach has the following computational cost: for a simulation having T timesteps, $F \ll T$ frequencies to compute, N_S fields in the flux region and N pixels in the entire computational cell this approach requires $\Theta(N + N_S T)$ storage and $\Theta(NT + T \log T)$ time (using a FFT-based chirp-z algorithm [54]).[2] The difficulty with this approach is that if a long simulation (large T) is required to obtain a high frequency resolution by the usual uncertainty relation [56], then the $\Theta(N_S T)$ storage requirements for the fields $\mathbf{E}(t)$ and $\mathbf{H}(t)$ at each point in S become excessive. Instead, Meep accumulates the DTFT summation of the fields at every point in S as the simulation progresses; once the time stepping has terminated, Eq. (5) can be evaluated using these Fourier-transformed fields.[3] The computational cost of this approach is $\Theta(N + N_S F)$ storage [much less than $\Theta(N_S T)$ if $F \ll T$] and $\Theta(NT + N_S FT)$ time. Although our current approach works well, another possible approach that we have been considering is to use Padé approximation: one stores the fields at every timestep on S, but instead of using the DTFT one constructs a Padé approximant to extrapolate the infinite-time DTFT from a short time series [57]. This requires $\Theta(N + N_S T)$ storage (but T is potentially much smaller) and $O(NT + T \log^2 T)$ time [58].

[2] Here, Θ has the usual meaning of an asymptotic tight bound [55].

[3] It is tempting to instead accumulate the Fourier transform of the Poynting flux at each time, but this is not correct since the flux is not a linear function of the fields.

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

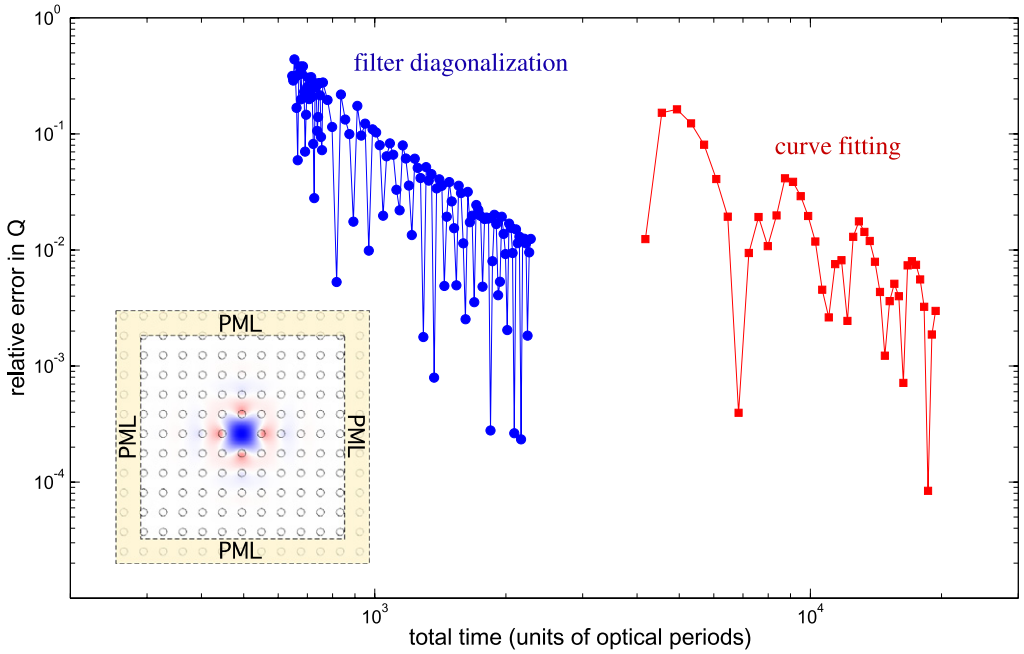

Fig. 9. Relative error in the quality factor Q for a photonic-crystal resonant cavity (inset, period a) with $Q \sim 10^6$, versus simulation time in units of optical periods of the resonance. Blue circles: filter-diagonalization method. Red squares: least-squares fit of energy in cavity to a decaying exponential. Filter-diagonalization requires many fewer optical periods than the decay time Q, whereas curve fitting requires a simulation long enough for the fields to decay significantly. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2. Analyzing resonant modes

Another major goal of time-domain simulations is analysis of resonant phenomena, specifically by determining the resonant frequency ω_0 and the quality factors Q (i.e., the number of optical cycles $2\pi/\omega_0$ for the field to decay by $e^{-2\pi}$) of one or more resonant modes. One straightforward and common approach to compute ω_0 and Q is by computing the DTFT of the field at some point in the cavity in response to a short pulse [1]: ω_0 is then the center of a peak in the DTFT and $1/Q$ is the fractional width of the peak at half maximum. The problem with this approach is that the Fourier uncertainty relation (equivalently, spectral leakage from the finite time window [56]) means that resolving the peak in this way requires a simulation much longer than Q/ω_0 (problematic for structures that may have very high Q, even 10^9 or higher [59]). Alternatively, one can perform a least squares fit of the field time-series within the cavity to an exponentially decaying sinusoid, but this leads to an ill-conditioned, nonconvex, nonlinear fitting problem (and is especially difficult if more than one resonant mode may be present). If only a single resonant mode is present, one can perform a least-squares fit of the energy in the cavity to a decaying exponential in order to determine Q, but a long simulation is still required to accurately resolve a large Q (as shown below). A more accurate and efficient approach, requiring only a short simulation even for very large Q values, is the technique of *filter diagonalization* originally developed for NMR spectroscopy, which transforms the time-series data into a small eigenproblem that is solved for all resonant frequencies and quality factors at once (even for multiple overlapping resonances) [60]. Chapter 16 of Ref. [1] compared the DFT peak-finding method with filter-diagonalization by attempting to resolve two near-degenerate modes in a microcavity, and demonstrated the latter's ability to accurately resolve closely-spaced peaks with as much as a factor of five times fewer timesteps. In our own work, we have used filter diagonalization to compute quality factors of 10^8 or more using simulations only a few hundred optical cycles in length [59]. We quantify the ability of filter diagonalization to resolve a large $Q \sim 10^6$ in Fig. 9, comparing the relative error in Q versus simulation time for filter diagonalization and the least-squares energy-fit method above. (The specific cavity is formed by a missing rod in a two-dimensional photonic crystal consisting of a square lattice of dielectric rods in air with period a, radius $0.2a$, and $\varepsilon = 12$ [22].) Fig. 9 demonstrates that filter diagonalization is able to identify the quality factor using almost an order of magnitude fewer time steps than the curve fitting method. (Another possible technique to identify resonant modes uses Padé approximants, which can also achieve high accuracy from a short simulation [57,61].)

5.3. Frequency-domain solver

A common electromagnetic problem is to find the fields that are produced in a geometry in response to a source at a single frequency ω. In principle, the solution of such problems need not involve time at all, but involve solving a linear equation purely in the frequency domain [22, Appendix D]; this can be achieved by many methods, such as finite-element methods [6–8], boundary-element methods [9–12], or finite-difference frequency-domain methods [62]. However, if one already has a full-featured parallel FDTD solver, it is attractive to exploit that solver for frequency-domain problems when they arise. The most straightforward approach is to simply run a simulation with a constant-frequency source—after a long time, when all transient effects from the source turn-on have disappeared, the result is the desired frequency-domain response. The difficulty with this approach is that a very long simulation may be required, especially if long-lived resonant modes are present at nearby frequencies (in which case a time $\gg Q/\omega$ is required to reach steady state). Instead, we show how the FDTD timestep can be used to directly plug a frequency-domain problem into an iterative linear solver, finding the frequency-domain response in the equivalent of many fewer timesteps while exploiting the FDTD code almost without modification.

The central component of any FDTD algorithm is the time step: an operation that advances the field by Δt in time. In order to extract a frequency-domain problem from this operation, we first

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

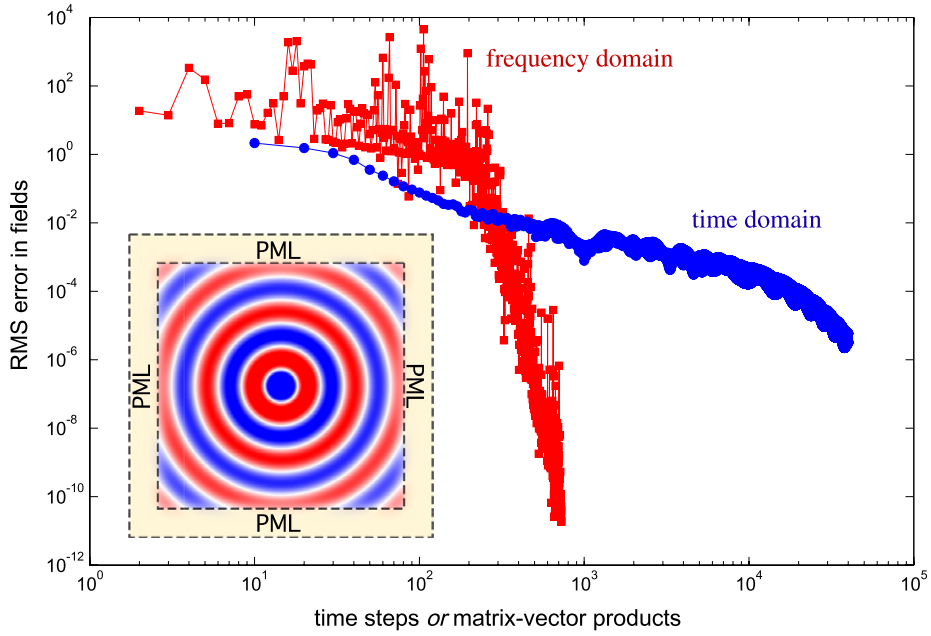

Fig. 10. Root-mean-square error in fields in response to a constant-frequency point source in vacuum (inset), for frequency-domain solver (red squares, adapted from Meep time-stepping code) vs. time-domain method (blue circles, running until transients decay away). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

express the timestep as an abstract linear operation: if $\mathbf{f}^n$ represents all of the fields (electric and magnetic) at time step n, then (in a linear time-invariant structure) the time step operation can be expressed in the form:

$$\mathbf{f}^{n+1} = \hat{T}_0 \mathbf{f}^n + \mathbf{s}^n, \tag{7}$$

where $\hat{T}_0$ is the timestep operator with no sources and $\mathbf{s}^n$ are the source terms (currents) from that time step. Now, suppose that one has a time-harmonic source $\mathbf{s}^n = e^{-i\omega n\Delta t}\mathbf{s}$ and wish to solve for the resulting time-harmonic (steady state) fields $\mathbf{f}^n = e^{-i\omega n\Delta t}\mathbf{f}$. Substituting these into Eq. (7), we obtain the following linear equation for the field amplitudes $\mathbf{f}$:

$$\left(\hat{T}_0 - e^{-i\omega\Delta t}\right)\mathbf{f} = -\mathbf{s}. \tag{8}$$

This can then be solved by an iterative method, and the key fact is that iterative methods for $Ax = b$ only require one to supply a function that multiplies the linear operator A by a vector [63]. Here, A is represented by $\hat{T}_0 - e^{-i\omega\Delta t}$ and hence one can simply use a standard iterative method by calling the unmodified timestep function from FDTD to provide the linear operator. To obtain the proper right-hand side $\mathbf{s}$, one merely needs to execute a single timestep (7), with sources, starting from zero field $\mathbf{f} = 0$. Since in general this linear operator is not Hermitian (especially in the presence of PML absorbing regions), we employ the BiCGSTAB-L algorithm (a generalization of the stabilized biconjugate gradient algorithm, where increasing the integer parameter L trades off increased storage for faster convergence) [64,65].

This technique means that all of the features implemented in our time-domain solver (not only arbitrary materials, subpixel averaging, and other physical features, but also parallelization, visualization, and user-interface features) are immediately available as a frequency-domain solver. To demonstrate the performance of this frequency-domain solver over the straightforward approach of simply running a long simulation until transients have disappeared, we computed the root-mean-square error in the field as a function of the number of time steps (or evaluations of $\hat{T}_0$ by BiCGSTAB-L) for two typical simulations. The first simulation, shown in Fig. 10, consists of a point source in vacuum surrounded by PML (inset). The frequency-domain solver (red squares) shows rapid, near-exponential convergence, while the error in the time-domain method (blue circles) decreases far more gradually (in fact, only polynomially). A much more challenging problem is to obtain the frequency-domain response of a cavity (ring resonator) with multiple long-lived resonant modes: in the time domain, these modes require a long simulation ($\sim Q$) to reach steady state, whereas in the frequency domain the resonances correspond to poles (near-zero eigenvalues of A) that increase the condition number and hence slow convergence [63]. Fig. 11 shows the results for a ring resonator cavity with multiple closely-spaced resonant modes, excited at one of the resonant frequencies (inset)—although both frequency- and time-domain methods take longer to converge than for the non-resonant case of Fig. 10, the advantage of the frequency-domain's exponential convergence is even more clear. The convergence is accelerated in frequency domain by using $L = 10$ (green diamonds) rather than $L = 2$ (at the expense of more storage). In time domain, the convergence is limited by the decay of high-Q modes at other frequencies, and the impact of these modes can be reduced by turning on the constant-frequency source more gradually (magenta triangles, hyperbolic-tangent turn-on of the source over 175 optical periods).

This is by no means the most sophisticated possible frequency-domain solver. For example, we currently do not use any preconditioner for the iterative scheme [63]. In two dimensions, a sparse-direct solver may be far more efficient than an iterative scheme [63]. The key point, however, is that programmer time is much more expensive than computer time, and this technique allows us to obtain substantial improvements in solving frequency-domain problems with only minimal changes to an existing FDTD program.

6. User interface and scripting

In designing the style of user interaction in Meep, we were guided by two principles. First, in research or design one hardly ever needs just *one* simulation—one almost always performs a whole series of simulations for a class of related problems, exploring the parameter dependencies of the results, optimizing some

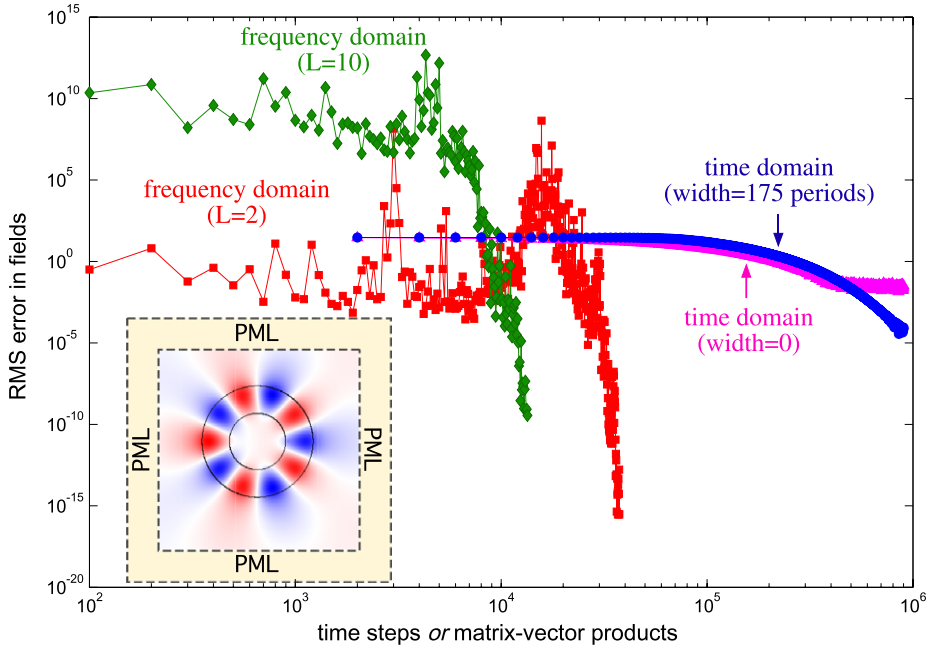

Fig. 11. Root-mean-square error in fields in response to a constant-frequency point source exciting one of several resonant modes of a dielectric ring resonator (inset, $\varepsilon = 11.56$), for frequency-domain solver (red squares, adapted from Meep time-stepping code) vs. time-domain method (magenta triangles, running until transients decay away). Green diamonds show frequency-domain BiCGSTAB-L solver for five times more storage, accelerating convergence. Blue circles show time-domain method for a more gradual turn-on of source, which avoids exciting long-lived resonances at other frequencies. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

output as a function of the input parameters, or looking at the same geometry under a sequence of different stimuli. Second, there is the Unix philosophy: "Write programs that do one thing and do it well" [66]—Meep should perform electromagnetic simulations, while for additional functionality it should be combined with other programs and libraries via standard interfaces like files and scripts.

Both of these principles argue against the graphical CAD-style interface common in commercial FDTD software. First, while graphical interfaces provide a quick and attractive route to setting up a single simulation, they are not so convenient for a series of related simulations. One commonly encounters problems where the size/position of certain objects is determined by the size/position of other objects, where the number of objects is itself a parameter (such as a photonic-crystal cavity surrounded by a variable number of periods [22]), where the length of the simulation is controlled by a complicated function of the fields, where one output is optimized as a function of some parameter, and many other situations that become increasingly cumbersome to express via a set of graphical tools and dialog boxes. Second, we don't *want* to write a mediocre CAD program—if we wanted to use a CAD program, we would use a professional-quality one, export the design to a standard interchange format, and write a conversion program to turn this format into what Meep expects. The most flexible and self-contained interface is, instead, to allow the user to control the simulation via an arbitrary program. Meep allows this style of interaction at two levels: via a low-level C++ interface, and via a standard high-level scripting language (Scheme) implemented by an external library (GNU Guile). The potential slowness of the scripting language is irrelevant because all of the expensive parts of the FDTD calculation are implemented in C/C++.

The high-level scripting interface to Meep is documented in detail, with several tutorials, on the Meep web page (http://ab-initio.mit.edu/meep), so we restrict ourselves to a single short example in order to convey the basic flavor. This example, in Fig. 12, computes the (2d) fields in response to a point source located within a dielectric waveguide. We first set the size of the computational cell to 16×8 (via `geometry-lattice`, so-called because it determines the lattice vectors in the periodic case)—recall that the interpretation of the unit of distance is arbitrary and up to the user (it could be 16 μm × 8 μm, in which case the frequency units are c/μm, or 16 mm × 8 mm with frequency units of c/mm, or any other convenient distance unit). Let us call this arbitrary unit of distance a. Then we specify the geometry within the cell as a list of geometric objects like blocks, cylinders, etc.—in this case by a single block defining the waveguide with $\varepsilon = 12$—or optionally by an arbitrary user-defined function $\varepsilon(x, y)$ (and μ, etc.). A layer of PML is then specified around the boundaries with thickness 1; this layer lies *inside* the computational cell and overlaps the waveguide, which is necessary in order to absorb waveguide modes when they reach the edge of the cell. We add a point source, in this case an electric-current source **J** in the z-direction (sources of arbitrary spatial profile can also be specified). The time-dependence of the source is a sharp turn-on to a continuous-wave source $\cos(\omega t)$ at the beginning of the simulation; gradual turn-ons, Gaussian pulses, or arbitrary user-specified functions of time can also be specified. The frequency is 0.15 in units of c/a, corresponding to a vacuum wavelength $\lambda = a/0.15$ (e.g. $\lambda \approx 6.67$ μm if $a = 1$ μm). We set the resolution to 10 pixels per unit distance (10 pixels/a), so that the entire computational cell is 160×80 pixels, and then run for 200 time units (units of a/c), corresponding to $200 \times 0.15 = 30$ optical periods. We output the dielectric function at the beginning, and the E_z field at the end.

In keeping with the Unix philosophy, Meep is not a plotting program; instead, it outputs fields and related data to the standard HDF5 format for scientific datasets [67], which can be read by many other programs and visualized in various ways. (We also provide a way to effectively "pipe" the HDF5 output to an external program within Meep: for example, to output the HDF5 file, convert it immediately to an image with a plotting program, and then delete the HDF5 file; this is especially useful for producing animations consisting of hundreds of frames.)

Another important technique to maintain flexibility is that of higher-order functions [68]: wherever it is practical, our functions

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

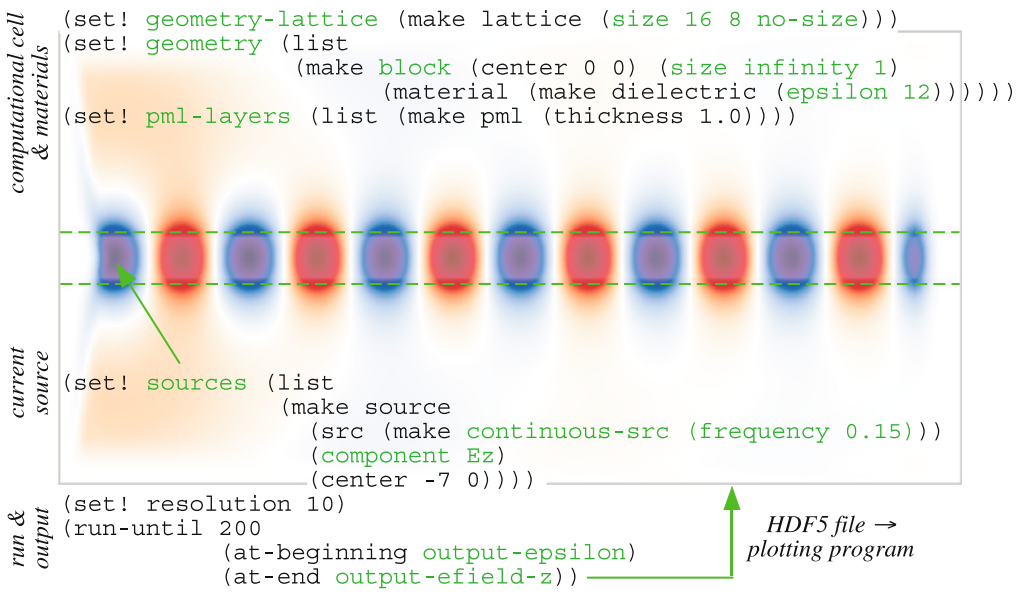

Fig. 12. A simple Meep example showing the E_z field in a dielectric waveguide ($\varepsilon = 12$) from a point source at a given frequency. A plot of the resulting field (blue/white/red = positive/zero/negative) is in the background, and in the foreground is the input file in the high-level scripting interface (the Scheme language). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

take functions as arguments instead of (or in addition to) numbers. Thus, for example, instead of specifying special input codes for all possible source distributions in space and time, we simply allow a user-defined function to be used. More subtly, the arguments `output-epsilon` and `output-efield-z` to the `run-until` function in Fig. 12 are actually functions themselves: we allow the user to pass arbitrary "step functions" to `run-until` that are called after every FDTD timestep and which can perform arbitrary computations on the fields as desired (or halt the computation if a desired condition is reached). The `output-efield-z` is simply a predefined step function that outputs E_z. These step-functions can be modified by transformation functions like `at-end`, which take step functions as arguments and return a new step function that only calls the original step functions at specified times (at the end of the simulation, or the beginning, or at certain intervals, for example). In this way, great flexibility in the output and computations is achieved. One can, for example, output a given field component only at certain time intervals after a given time, and only within a certain subvolume or slice of the computational cell, simply by composing several of these transformations. One can even output an arbitrary user-defined function of the fields instead of predetermined components.

There is an additional subtlety when it comes to field output, because of the Yee lattice in which different field components are stored at different points; presented in this way to the user, it would be difficult to perform post-processing involving multiple field components, or even to compare plots of different field components. As mentioned in Section 3 and again in Section 7.2, therefore, the field components are automatically interpolated from the Yee grid onto a fixed "centered" grid in each pixel when exported to a file.

Although at a simplistic level the input format can just be considered as a file format with a lot of parentheses, because Scheme is a full-fledged programming language one can control the simulation in essentially arbitrary ways. Not only can one write loops and use arithmetic to define the geometry and the relationships between the objects or perform parameter sweeps, but we also expose external libraries for multivariable optimization, integration, root-finding, and other tasks in order that they can be coupled with simulations.

Parallelism is completely transparent to the user: exactly the same input script is fed to the parallel version of Meep (written with the MPI message-passing standard for distributed-memory parallelism [69]) as to the serial version, and the distribution of the data across processors and the collection of results is handled automatically.

7. Abstraction versus performance

In an FDTD simulation, essentially just one thing has to be fast: inner loops over all the grid points or some large fraction thereof. Everything else is negligible in terms of computation time (but not programmer time!), so it can use high-level abstractions without penalty—for example, the use of a Scheme interpreter as the user interface has no performance consequences for a typical computation, because the inner loops are not written in Scheme.[4] For these inner loops, however, there is a distinct tension between abstraction (or simplicity) and performance, and in this section we discuss some of the tradeoffs that result from this tension and the choices that have been made in Meep.

The primacy of inner loops means that some popular principles of abstraction must be discarded. A few years ago, a colleague of ours attempted to write a new FDTD program in textbook object-oriented C++ style: every pixel in the grid would be an object, and every type of material would be a subclass overriding the necessary timestepping and field-access operations. Timestepping would consist of looping over the grid, calling some "step" method of each object, so that objects of different materials (magnetic, dielectric, nonlinear, etc.) would dynamically apply the corresponding field-update procedures. The result of this noble experiment was a working program but a performance failure, many times slower than the aging Fortran software it was intended to replace: the performance overhead of object dereferencing, virtual method dis-

[4] The exception to this rule is when the user supplies a Scheme function and asks that it be evaluated for every grid point, for example to integrate some function of the fields. If this is done frequently during the simulation, it is slow; in these circumstances, however, the user can replace the Scheme function with one written in C/C++ if needed. This is rare because most such functions that might be used frequently during a simulation, such as energy or flux, are already supplied in C/C++ within Meep.

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

patch, and function calls in the inner loop overwhelmed all other considerations. In Meep, each field's components are stored as simple linear arrays of floating-point numbers in row-major (C) order (parallel-array data structures worthy of Fortran 66), and there are separate inner loops for each type of material (more on this below). In a simple experiment on a 2.8 GHz Intel Core 2 CPU, merely moving the `if` statements for the different material types into these inner loops decreased Meep's performance by a factor of two in a typical 3d calculation and by a factor of six in 2d (where the calculations are simpler and hence the overhead of the conditionals is more significant). The cost of the conditionals, including the cost of mispredicted branches and subsequent pipeline stalls [70] along with the frustration of compiler unrolling and vectorization, easily overwhelmed the small cost of computing, e.g., $\nabla \times \mathbf{H}$ at a single point.

7.1. Timestepping and cache tradeoffs

One of the dominant factors in performance on modern computer systems is not arithmetic, but memory: random memory access is far slower than arithmetic, and the organization of memory into a hierarchy of caches is designed to favor locality of access [70]. That is, one should organize the computation so that as much work as possible is done with a given datum once it is fetched (temporal locality) and so that subsequent data that are read or written are located nearby in memory (spatial locality). The optimal strategies to exploit both kinds of locality, however, appear to lead to sacrifices of abstraction and code simplicity so severe that we have chosen instead to sacrifice some potential performance in the name of simplicity.

As it is typically described, the FDTD algorithm has very little temporal locality: the field at each point is advanced in time by Δt, and then is not modified again until *all* the fields at *every* other point in the computational cell have been advanced. In order to gain temporal locality, one must employ *asynchronous timestepping*: essentially, points in small regions of space are advanced several steps in time before advancing points far away, since over a short time interval the effects of far-away points cannot be felt. A detailed analysis of the characteristics of this problem, as well as a beautiful "cache-oblivious" algorithm that automatically exploits a cache of any size for grids of any dimensionality, is described in Ref. [71]. On the other hand, an important part of Meep's usability is the abstraction that the user can perform arbitrary computations or output using the fields in any spatial region at any time, which seems incompatible with the fields at different points in space being out-of-sync until a predetermined end of the computation. The bookkeeping difficulty of reconciling these two viewpoints led us to reject the asynchronous approach, despite its potential benefits.

However, there may appear to be at least a small amount of temporal locality in the synchronous FDTD algorithm: first **B** is advanced from $\nabla \times \mathbf{E}$, then **H** is computed from **B** and μ, then **D** is advanced from $\nabla \times \mathbf{H}$, then **E** is computed from **D** and ε. Since most fields are used at least once after they are advanced, surely the updates of the different fields can be merged into a single loop, for example advancing **D** at a point and then immediately computing **E** at the same point—the **D** field need not even be stored. Furthermore, since by merging the updates one is accessing several fields around the same point at the same time, perhaps one can gain *spatial* locality by interleaving the data, say by storing an array of $(\mathbf{E}, \mathbf{H}, \varepsilon, \mu)$ tuples instead of separate arrays. Meep does not do either of these things, however, for two reasons, the first of which is more fundamental. As is well known, one cannot easily merge the **B** and **H** updates with the **D** and **E** updates at the same point, because the discretized $\nabla \times$ operation is nonlocal (involves multiple grid points)—this is why one normally updates **H** everywhere in space before updating **D** from $\nabla \times \mathbf{H}$, because in computing $\nabla \times \mathbf{H}$ one uses the values of **H** at different grid points and all of them must be in sync. A similar reasoning, however, applies to updating **E** from **D** and **H** from **B**, once the possibility of anisotropic materials is included—because the Yee grid stores different field components at different locations, any accurate handling of off-diagonal susceptibilities must also inevitably involve fields at multiple points (as in Ref. [39]). To handle this, **D** must be stored explicitly and the update of **E** from **D** must take place after **D** has been updated everywhere, in a separate loop. And since each field is updated in a separate loop, the spatial-locality motivation to merge the field data structures rather than using parallel arrays is largely removed.

Of course, not all simulations involve anisotropic materials—although they appear even in many simulations with nominally isotropic materials thanks to the subpixel averaging discussed in Section 3—but this leads to the second practical problem with merging the **E** and **D** (or **H** and **B**) update loops: the combinatorial explosion of the possible material cases. The update of **D** from $\nabla \times \mathbf{H}$ must handle 16 possible cases, each of which is a separate loop (see above for the cost of putting conditionals inside the loops): with or without PML (4 cases, depending upon the number of PML conductivities and their orientation relative to the field), with or without conductivity, and with the derivative of two **H** components (3d) or only one **H** component (2d TE polarization). The update of **E** from **D** involves 12 cases: with or without PML (2 cases, distinct from those in the **D** update), the number of off-diagonal ε^{-1} components (3 cases: 0, 1, or 2), and with or without nonlinearity (2 cases). If we attempted to join these into a single loop, we would have $16 \times 12 = 192$ cases, a code-maintenance headache. (Note that the multiplicity of PML cases comes from the fact that, including the corners of the computational cell, we might have 0 to 3 directions of PML, and the orientation of the PML directions relative to a given field component matters greatly.)

The performance penalty of separate **E** and **D** (or **H** and **B**) updates appears to be modest. Even if, by somehow merging the loops, one assumes that the time to compute $\mathbf{E} = \varepsilon^{-1}\mathbf{D}$ could become *zero*, benchmarking the relative time spent in this operation indicates that a typical 3d transmission calculation would be accelerated by only around 30% (and less in 2d).

7.2. The loop-in-chunks abstraction

Finally, let us briefly mention a central abstraction that, while not directly visible to end-users of Meep, is key to the efficiency and maintainability of large portions of the software (field output, current sources, flux/energy computations and other field integrals, and so on). The purpose of this abstraction is to mask the complexity of the partitioning of the computational cell into overlapping chunks connected by symmetries, communication, and other boundary conditions as described in Section 2.

Consider the output of the fields at a given timestep to an HDF5 datafile. Meep provides a routine `get-field-pt` that, given a point in space, interpolates it onto the Yee grid and returns a desired field component at that point. In addition to interpolation, this routine must also transform the point onto a chunk that is actually stored (using rotations, periodicity, etc.) and communicate the data from another processor if necessary. If the point is on a boundary between two chunks, the interpolation process may involve multiple chunks, multiple rotations, etc., and communications from multiple processors. Because this process involves only a single point, it is not easily parallelizable. Now, to output the fields everywhere in some region to a file, one approach is to simply call `get-field-pt` for every point in a grid for that region and output the results, but this turns out to be tremendously slow because of the repeated transformations and communications for every single point. We nevertheless want to interpolate fields for

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008

output rather than dumping the raw Yee grid, because it is much easier for post-processing if the different field components are interpolated onto the same grid; also, to maintain transparency of features like symmetry one would like to be able to output the whole computational cell (or an arbitrary subset) even if only a part of it is stored. Almost exactly the same problems arise for integrating things like flux $\mathbf{E} \times \mathbf{H}$ or energy or user-defined functions of the fields (noting that functions combining multiple field components require interpolation), and also for implementing volume (or line, or surface) sources which must be projected onto the grid in some arbitrary volume.

One key to solving this difficulty is to realize that, when the field in some volume V is needed (for output, integration, and so on), the rotations, communications, etc. for points in V are identical for all the points in the intersection of V with some chunk (or one of its rotations/translations). The second is to realize that, when interpolation is needed, there is a particular grid for which interpolation is easy: for *owned* points of the *centered* grid (Section 2) lying at the center of each pixel, it is always possible to interpolate from fields on any Yee grid without any inter-chunk communication and by a simple equal-weight averaging of at most 2^d points in d dimensions.

The combination of these two observations leads to the *loop-in-chunks* abstraction. Given a (convex rectilinear) volume V and a given grid (either centered, or one of the Yee-field grids), it computes the intersection of all the chunks and their rotations/translations with V. For each intersection it invokes a caller-specified function, passing the portion of the chunk, the necessary rotations (etc.) of the fields, and interpolation weights (if needed, for the boundary of V). That function then processes the specified portion of the chunk (for example, outputting it to the corresponding portion of a file, or integrating the desired fields). All of this can proceed in parallel (with each processor considering only those chunks stored locally). This is (relatively) fast because the rotations, interpolations, and so on are computed only once per chunk intersection, while the inner loop over all grid points in each chunk can be as tight as necessary. Moreover, all of the rather complicated and error-prone logic involved in computing V's intersection with the chunks (e.g., special care is required to ensure that each conceptual grid point is processed exactly once despite chunk overlaps and symmetries) is localized to one place in the source code; field output, integration, sources, and other functions of the fields are isolated from this complexity.

8. Concluding remarks

We have reviewed in this paper a number of the unusual implementation details of Meep that distinguish our software package from standard textbook FDTD methods. Beginning with a discussion of the fundamental structural unit of chunks that constitute the Yee grid and enable parallelization: we provided an overview of Meep's core design philosophy of creating an illusion of continuous space and time for inputs and outputs; we explained and motivated the somewhat unusual design intricacies of nonlinear materials and PMLs; we discussed important aspects of Meep's computational methods for flux spectra and resonant modes; we demonstrated the formulation of a frequency-domain solver requiring only minimal modifications to the underlying time-stepping algorithm. In addition to the inner workings of Meep's internal structure, we reviewed how such features are accessible to users via an external scripting interface.

We believe that a free/open-source, full-featured FDTD package like Meep can play a vital role in enabling new research in electromagnetic phenomena. Not only does it provide a low barrier to entry for standard FDTD simulations, but the simplicity of the FDTD algorithm combined with access to the source code offers an easy route to investigate new physical phenomena coupled with electromagnetism. For example, we have colleagues working on coupling multi-level atoms to electromagnetism within Meep for modeling lasing and saturable absorption, adapting published techniques from our and other groups [16–20], but also including new physics such as diffusion of excited gases. Other colleagues have modified Meep for modeling gyromagnetic media in order to design new classes of "one-way" waveguides [72]. Meep is even being used to simulate the quantum phenomena of Casimir forces (from quantum vacuum fluctuations, which can be computed from classical Green's functions) [73,74]—in fact, this was possible without any modifications of the Meep code due to the flexibility of Meep's scripting interface. We hope that other researchers, with the help of the understanding of Meep's architecture that this paper provides, will be able to adapt Meep to future phenomena that we have not yet envisioned.

Acknowledgements

This work was supported in part by the Materials Research Science and Engineering Center program of the National Science Foundation under Grant Nos. DMR-9400334 and DMR-0819762, by the Army Research Office through the Institute for Soldier Nanotechnologies under contract DAAD-19-02-D0002, and also by Dr. Dennis Healy of DARPA MTO under award N00014-05-1-0700 administered by the Office of Naval Research. We are also grateful to A.W. Rodriguez and A.P. McCauley for their efforts to generalize Meep for quantum-Casimir problems, S.L. Chua for his work on dispersive and multi-level materials, and Y. Chong for early support in Meep for gyrotropic media.

References

[1] A. Taflove, S.C. Hagness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd ed., Artech, Norwood, MA, 2005.
[2] K.S. Kunz, R.J. Luebbers, The Finite-Difference Time-Domain Method for Electromagnetics, CRC Press, Boca Raton, 1993.
[3] D.M. Sullivan, Electromagnetic Simulation Using the FDTD Method, Wiley–IEEE Press, New York, 2000.
[4] A. Elsherbeni, V. Demir, The Finite Difference Time Domain Method for Electromagnetics: With MATLAB Simulations, SciTech, Rayleigh, NC, 2009.
[5] W. Yu, R. Mittra, T. Su, Y. Liu, X. Yang, Parallel Finite-Difference Time-Domain Method, Artech House, Norwood, MA, 2006.
[6] M.N. Sadiku, Numerical Techniques in Electromagnetics, 2nd ed., CRC, 2000.
[7] J. Jin, The Finite Element Method in Electromagnetics, 2nd ed., Wiley–IEEE Press, 2002.
[8] K. Yasumoto (Ed.), Electromagnetic Theory and Applications for Photonic Crystals, CRC, 2005.
[9] S. Rao, D. Wilton, A. Glisson, Electromagnetic scattering by surfaces of arbitrary shape, IEEE Trans. Antennas and Propagation 30 (3) (1982) 409–418.
[10] K. Umashankar, A. Taflove, S. Rao, Electromagnetic scattering by arbitrary shaped three-dimensional homogeneous lossy dielectric objects, IEEE Trans. Antennas and Propagation 34 (6) (1986) 758–766.
[11] M. Bonnet, Boundary Integral Equation Methods for Solids and Fluids, Wiley, 1999.
[12] W.C. Chew, J.-M. Jin, E. Michielssen, J. Song (Eds.), Fast and Efficient Algorithms in Computational Electromagnetics, Artech House, 2000.
[13] S.G. Johnson, M. Ibanescu, M.A. Skorobogatiy, O. Weisberg, J.D. Joannopoulos, Y. Fink, Perturbation theory for Maxwell's equations with shifting material boundaries, Phys. Rev. E 65 (2002) 066611.
[14] L. Zhang, J. Lee, A. Farjadpour, J. White, S. Johnson, A novel boundary element method with surface conductive absorbers for 3-D analysis of nanophotonics, in: Microwave Symposium Digest, 2008 IEEE MTT-S International, 2008, pp. 523–526.
[15] S.G. Johnson, J.D. Joannopoulos, Block-iterative frequency-domain methods for Maxwell's equations in a planewave basis, Opt. Express 8 (3) (2001) 173–190.
[16] R.W. Ziolkowski, J.M. Arnold, D.M. Gogny, Ultrafast pulse interactions with two-level atoms, Phys. Rev. A 52 (4) (1995) 3082–3094.
[17] A.S. Nagra, R.A. York, FDTD analysis of wave propagation in nonlinear absorbing and gain media, IEEE Trans. Antennas and Propagation 46 (3) (1998) 334–340.
[18] S.-H. Chang, A. Taflove, Finite-difference time-domain model of lasing action in a four-level two-electron atomic system, Opt. Express 12 (16) (2004) 3827–3833.

[19] Y. Huang, S.-T. Ho, Computational model of solid-state, molecular, or atomic media for FDTD simulation based on a multi-level multi-electron system governed by Pauli exclusion and Fermi–Dirac thermalization with application to semiconductor photonics, Opt. Express 14 (8) (2006) 3569–3587.
[20] P. Bermel, E. Lidorikis, Y. Fink, J.D. Joannopoulos, Active materials embedded in photonic crystals and coupled to electromagnetic radiation, Phys. Rev. B 73 (2006) 165125.
[21] J.D. Jackson, Classical Electrodynamics, 3rd ed., Wiley, New York, 1998.
[22] J.D. Joannopoulos, S.G. Johnson, R.D. Meade, J.N. Winn, Photonic Crystals: Molding the Flow of Light, 2nd ed., Princeton Univ. Press, 2008.
[23] K.S. Yee, Numerical solution of initial boundary value problems involving Maxwells Equations in isotropic media, IEEE Trans. Antennas and Propagation 14 (3) (1966) 302–307.
[24] M.J. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (1984) 484–512.
[25] I.S. Kim, W.J.R. Hoefer, A local mesh refinement algorithm for the time domain-finite difference method using Maxwell's curl equations, IEEE Trans. Microwave Theory Tech. 38 (6) (1990) 812–815.
[26] S.S. Zivanovic, K.S. Yee, K.K. Mei, A. Subgridding, Method for the time-domain finite-difference method to solve Maxwell's equations, IEEE Trans. Microwave Theory Tech. 39 (3) (1991) 471–479.
[27] M. Okoniewski, E. Okoniewska, M.A. Stuchly, Three-dimensional subgridding algorithm for FDTD, IEEE Trans. Antennas and Propagation 45 (3) (1997) 422–429.
[28] C. Lin, L. Snyder, Principles of Parallel Programming, Addison–Wesley, 2008.
[29] T. Inui, Y. Tanabe, Y. Onodera, Group Theory and Its Applications in Physics, Springer-Verlag, Terlos, 1996.
[30] U. Trottenberg, C.W. Oosterlee, A. Schuller, Multigrid, Academic Press, 2000.
[31] L. Landau, L. Pitaevskii, E. Lifshitz, Electrodynamics of Continuous Media, 2nd ed., Butterworth–Heinemann, 1984.
[32] C. Luo, M. Ibanescu, S.G. Johnson, J.D. Joannopoulos, Cerenkov radiation in photonic crystals, Science 299 (2003) 368–371.
[33] A. Farjadpour, D. Roundy, A. Rodriguez, M. Ibanescu, P. Bermel, J. Joannopoulos, S. Johnson, G. Burr, Improving accuracy by sub-pixel smoothing in the finite-difference time domain, Opt. Lett. 31 (2006) 2972–2974.
[34] A.F. Oskooi, C. Kottke, S.G. Johnson, Accurate finite-difference time-domain simulation of anisotropic media by subpixel smoothing, Opt. Lett. 34 (18) (2009) 2778–2780.
[35] A. Ditkowski, K. Dridi, J.S. Hesthaven, Convergent Cartesian grid methods for Maxwell's equations in complex geometries, J. Comput. Phys. 170 (2001) 39–80.
[36] C. Kottke, A. Farjadpour, S.G. Johnson, Perturbation theory for anisotropic dielectric interfaces, and application to subpixel smoothing of discretized numerical methods, Phys. Rev. E 77 (2008) 036611.
[37] S. Dey, R. Mittra, A conformal finite-difference time-domain technique for modeling cylindrical dielectric resonators, IEEE Trans. Microwave Theory Tech. 47 (9) (1999) 1737–1739.
[38] R.D. Meade, A.M. Rappe, K.D. Brommer, J.D. Joannopoulos, O.L. Alerhand, Accurate theoretical analysis of photonic band-gap materials, Phys. Rev. B 48 (1993) 8434–8437;
S.G. Johnson, Phys. Rev. B 55 (1997) 15942, Erratum.
[39] G. Werner, J. Cary, A stable FDTD algorithm for non-diagonal anisotropic dielectrics, J. Comput. Phys. 226 (2007) 1085–1101.
[40] P. Mezzanotte, L. Roselli, R. Sorrentino, A. Simple, Way to model curved metal boundaries in FDTD algorithm avoiding staircase approximation, IEEE Microwave Guided Wave Lett. 5 (8) (1995) 267–269.
[41] J. Anderson, M. Okoniewski, S.S. Stuchly, Practical 3-D contour/staircase treatment of metals in FDTD, IEEE Microwave Guided Wave Lett. 6 (3) (1996) 146–148.
[42] A. Deinega, I. Valuev, Subpixel smoothing for conductive and dispersive media in the finite-difference time-domain method, Opt. Lett. 32 (23) (2007) 3429–3431.
[43] P. Leung, S. Liu, K. Young, Completeness and time-independent perturbation of the quasinormal modes of an absorptive and leaky cavity, Phys. Rev. A 49 (1994) 3982–3989.
[44] S.G. Johnson, M.L. Povinelli, M. Soljačić, A. Karalis, S. Jacobs, J.D. Joannopoulos, Roughness losses and volume-current methods in photonic-crystal waveguides, Appl. Phys. B 81 (2005) 283–293.
[45] A.F. Oskooi, L. Zhang, Y. Avniel, S.G. Johnson, The failure of perfectly matched layers, and towards their redemption by adiabatic absorbers, Opt. Express 16 (15) (2008) 11376–11392.
[46] Q.H. Liu, J.Q. He, Quasi-PML for waves in cylindrical coordinates, Microwave and Optical Tech. Lett. 19 (2) (1998) 107–111.
[47] N. Bloembergen, Nonlinear Optics, W.A. Benjamin, New York, 1965.
[48] G.P. Agrawal, Nonlinear Fiber Optics, 3rd ed., Academic Press, San Diego, 2001.
[49] A. Rodriguez, M. Soljačić, J.D. Joannopoulos, S.G. Johnson, $\chi^{(2)}$ and $\chi^{(3)}$ harmonic generation at a critical power in inhomogeneous doubly resonant cavities, Opt. Express 15 (12) (2007) 7303–7318.
[50] R.W. Boyd, Nonlinear Optics, Academic Press, London, UK, 1992.
[51] J. Baker, A. George, P. Graves-Morris, Padé Approximants, 2nd ed., Cambridge University Press, 1996.
[52] F.L. Teixeira, W.C. Chew, Systematic derivation of anisotropic PML absorbing media in cylindrical and spherical coordinates, IEEE Microwave Guided Wave Lett. 7 (11) (1997) 371–373.
[53] J.-Q. He, Q.-H. Liu, A nonuniform cylindrical FDTD algorithm with improved PML and quasi-PML absorbing boundary conditions, IEEE Trans. Geoscience Remote Sensing 37 (2) (1999) 1066–1072.
[54] D.H. Bailey, P.N. Swartztrauber, A. Fast, Method for the numerical evaluation of continuous Fourier and Laplace transforms, SIAM J. Sci. Comput. 15 (5) (1994) 1105–1110.
[55] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd ed., MIT Press, 2009.
[56] A.V. Oppenheim, R.W. Schafer, Discrete-Time Signal Processing, 3rd ed., Prentice-Hall, 2009.
[57] W.-H. Guo, W.-J. Li, Y.-Z. Huang, Computation of resonant frequencies and quality factors of cavities by FDTD technique and Padé approximation, IEEE Microwave and Wireless Comp. Lett. 11 (5) (2001) 223–225.
[58] S. Cabay, D.-K. Choi, Algebraic computations of scaled Padé fractions, SIAM J. Comput. 15 (1) (1986) 243–270.
[59] A. Rodriguez, M. Ibanescu, J.D. Joannopoulos, S.G. Johnson, Disorder-immune confinement of light in photonic-crystal cavities, Opt. Lett. 30 (2005) 3192–3194.
[60] V.A. Mandelshtam, H.S. Taylor, Harmonic inversion of time signals and its applications, J. Chem. Phys. 107 (17) (1997) 6756–6769.
[61] S. Dey, R. Mittra, Efficient computation of resonant frequencies and quality factors of cavities via a combination of the finite-difference time-domain technique and the Padé approximation, IEEE Microwave Guided Wave Lett. 8 (12) (1998) 415–417.
[62] A. Christ, H.L. Hartnagel, Three-dimensional finite-difference method for the analysis of microwave-device embedding, IEEE Trans. Microwave Theory Tech. 35 (8) (1987) 688–696.
[63] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H.V. der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, PA, 1994.
[64] G.L.G. Sleijpen, D.R. Fokkema, BiCGSTAB(L) for linear equations involving unsymmetric matrices with complex spectrum, Electron. Trans. Numer. Anal. 1 (1993) 11–32.
[65] G.L.G. Sleijpen, H.A. van der Vorst, D.R. Fokkema, BiCGstab(L) and other hybrid Bi-CG methods, Numer. Algorithms 7 (1994) 75–109.
[66] P.H. Salus, A. Quarter, Century of UNIX, Addison–Wesley, Reading, MA, 1994.
[67] M. Folk, R.E. McGrath, N. Yeager, HDF: An update and future directions, in: Proc. 1999 Geoscience and Remote Sensing Symposium (IGARSS), vol. 1, IEEE Press, Hamburg, Germany, 1999, pp. 273–275.
[68] H. Abelson, G.J. Sussman, Structure and Interpretation of Computer Programs, MIT Press, Cambridge, MA, 1985.
[69] T.M. Forum, MPI: A Message Passing Interface, in: Supercomputing '93, Portland, OR, 1993, pp. 878–883.
[70] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, 3rd ed., Elsevier, San Francisco, CA, 2003.
[71] M. Frigo, V. Strumpen, The memory behavior of cache oblivious stencil computations, J. Supercomputing 39 (2) (2007) 93–112.
[72] Z. Wang, Y.D. Chong, J.D. Joannopoulos, M. Soljačić, Reflection-free one-way edge modes in a gyromagnetic photonic crystal, Phys. Rev. Lett. 100 (2008) 013905.
[73] A.W. Rodriguez, A.P. McCauley, J.D. Joannopoulos, S.G. Johnson, Casimir forces in the time domain: Theory, Phys. Rev. A 80 (2009) 012115.
[74] A.P. McCauley, A.W. Rodriguez, J.D. Joannopoulos, S.G. Johnson, Casimir forces in the time domain: II. Applications, arXiv.org e-Print archive, arXiv:0906.5170, 2009.

Please cite this article in press as: A.F. Oskooi et al., MEEP: A flexible free-software package for electromagnetic simulations by the FDTD method, Computer Physics Communications (2009), doi:10.1016/j.cpc.2009.11.008