Cgmx User Guide: An Overture Solver for Maxwell's Equations on Composite Grids

William D. Henshaw
Department of Mathematical Sciences,
Rensselaer Polytechnic Institute,
Troy, NY, USA, 12180. May 18, 2015

Abstract:
Cgmx is a program that can be used to solve the time-dependent Maxwell's equations of electromagnetics on composite overlapping grids in two and three space dimensions. This document explains how to run Cgmx. Numerous examples are given. The boundary conditions, initial conditions and forcing functions are described. Cgmx solves Maxwell's equations in second-order form. Second-order accurate and fourth-order accurate approximations are available. Cgmx can accurately handle material interfaces. Cgmx also implements a version of the Yee scheme for the first-order system form of Maxwell's equations. The Yee scheme can be run on a single Cartesian grid with material boundaries treated with a stair-step approximation.

Contents

1 Introduction

Cgmx is a program that can be used to solve Maxwell's equations of electromagnetics on composite overlapping grids [5]. It is built upon the **Overture** framework [1],[4],[2].

More information about **Overture** can be found on the **Overture** home page, `overtureFramework.org`.

Cgmx features:

- solve the time-domain Maxwell equations on overlapping grids to second- and fourth-order accuracy in two and three space dimensions.
- solve material interface problems to second- and fourth-order accuracy in two and three space dimensions.
- solve the time-domain Maxwell equations on a Cartesian grid with variable μ and ϵ using the Yee scheme.

Installation: To install cgmx you should follow the instructions for installing Overture and cg from the Overture web page. Note that cgmx is NOT built by default when building the cg solvers so that after installing cg you must go into the `cg/mx` directory and type 'make'. If you only wish to build cgmx and not any of the other cg solvers, then after building Overture and unpacking cg, go to the `cg/mx` and type make.
The cgmx solver is found in the `mx` directory in the **cg** distribution and has sub-directories

`bin` : contains the executable, cgmx. You may want to put this directory in your path.

`check` : contains regression tests.

`cmd` : sample command files for running cgmx, see section (2).

`doc` : documentation.

`lib` : contains the cgmx library, `libCgmx.a`.

`src` : source files

1.1 Basic steps

Here are the basic steps to solve a problem with cgmx.

1. Generate an overlapping grid with ogen.
2. Run cgmx (found in the `bin/cgmx` directory).
3. Assign the boundary conditions and initial conditions.
4. Choose the parameters for the PDE (such as material properties such as μ, ϵ))
5. Choose run time parameters, time to integrate to, time stepping method etc.
6. Compute the solution (optionally plotting the results as the code runs).
7. When the code is finished you can look at the results (provided you saved a 'show file') using `plotStuff`.

The commands that you enter to run cgmx can be saved in a command file (by default they are saved in the file 'cgmx.cmd'). This command file can be used to re-run the same problem by typing 'cgmx file.cmd'. The command file can be edited to change parameters.

To get started you can run one of the demo's that come with cgmx, these are explained in section (2). For more information on the algorithms and approximations used in Cgmx see

1. The *Cgmx Reference Manual* [6].
2. *A High-Order Accurate Parallel Solver for Maxwell's Equations on Overlapping Grids* [5].

2 Sample command files for running cgmx

Command files are supported throughout the Overture. They are files that contain lists of commands. These commands can initially be saved when the user is interactively choosing options. The command files can then be used to re-run the job. Command files can be edited and changed.

In this section we present a number of command files that can be used to run cgmx.

2.1 Running a command file

Given a command file for cgmx such as `cyleigen.cmd`, found in `cmd/cyleigen.cmd`, one can type '`cgmx cyleigen.cmd`' to run this command file . You can also just type '`cgmx cyleigen`, leaving off the `.cmd` suffix. Typing '`cgmx -noplot cyleigen`' will run without interactive graphics (unless the command file turns on graphics). Note that here I assume that the `bin` directory is in your path so that the `cgmx` command is found when you type it's name. The Cgmx sample command files will automatically look for an overlapping grid in the `Overture/sampleGrids` directory, unless the grid is first found in the location specified in the command file.

When you run a command file a graphics screen will appear and after some processing the run-time dialog should appear and the initial conditions will be plotted. The program will also print out some information about the problem being solved. At this point choose `continue` or `movie mode`. Section (6.2) describes the options available in the run time dialog.

When running in parallel it is convenient to define a shell variable for the parallel version of cgmx. For example in the tcsh shell you might use

```
set cgmxp  = ${CGBUILDPREFIX}/mx/bin/cgmx
```

An example of running the parallel version is then

```
mpirun -np 2 $cgmxp cic.planeWaveBC -g=cice2.order4.hdf
```

2.2 Scattering of a plane wave from a two-dimensional conducting cylinder

The command file `cic.planeWaveBC.cmd` can be used to compute the scattering of a plane wave from a PEC (perfectly electric conducting) cylinder. The exact solution is known for this case and the errors in the numerical solution are determined while the solution is being computed.
(1) Generate the grid using the command file `cicArg.cmd` in `Overture/sampleGrids`:

```
ogen -noplot cicArg -order=4 -interp=e -factor=2
```

(2) Run cgmx:

```
cgmx cic.planeWaveBC -g=cice2.order4.hdf
```

Parallel: If you have built the parallel version then use

```
mpirun -np 2 $cgmxp cic.planeWaveBC -g=cice2.order4.hdf
```

where `$cgmxp` is a variable that points to the parallel version of cgmx (see Section 2.1).

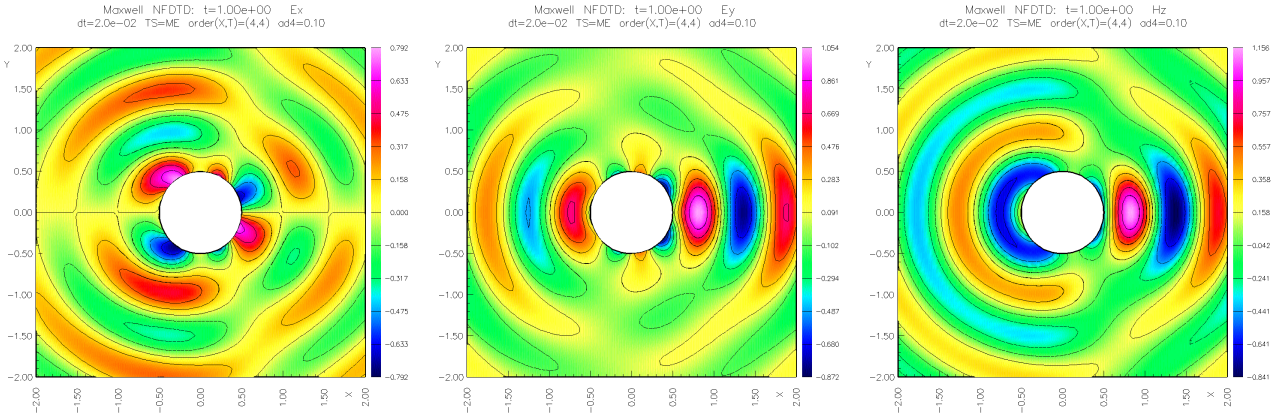

Figure 1: Scattering of a plane wave by a PEC cylinder. Computed solution at $t = 1.0$, E_x, E_y and H_z.

Notes:

1. See the comments at the top of the command file for command line arguments and further examples.
2. This case was run with fourth-order accuracy.

2.3 Scattering from a three-dimensional conducting sphere

The command file `sib.planeWaveBC.cmd` can be used to compute the scattering of a plane wave from a PEC sphere. The exact solution is known for this case and the errors in the numerical solution are determined while the solution is being computed.
(1) Generate the grid using the command file `sibArg.cmd` in `Overture/sampleGrids`:

```
ogen -noplot sibArg -order=4 -interp=e -factor=4
```

(2) Run cgmx:

```
cgmx sib.planeWaveBC -g=sibe4.order4.hdf
```

Parallel: If you have built the parallel version then use

```
mpirun -np 2 $cgmxp sib.planeWaveBC -g=sibe4.order4.hdf
```

where `$cgmxp` is a variable that points to the parallel version of cgmx, e.g.,

```
set cgmxp  = ${CGBUILDPREFIX\}/mx/bin/cgmx}
```

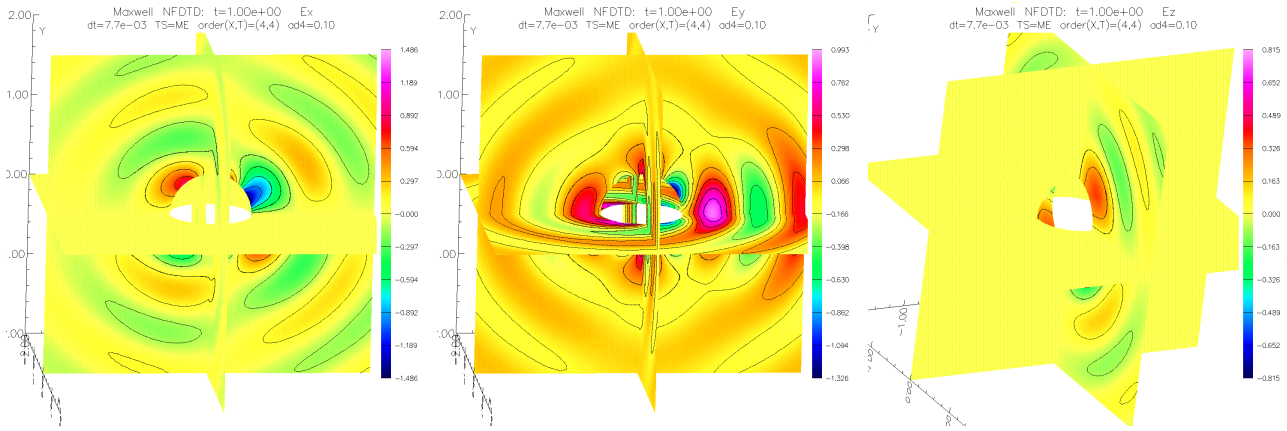

Figure 2: Scattering of a plane wave by a PEC sphere. Computed solution at $t = 1.0$, E_x, E_y and E_z.

Notes:

1. See the comments at the top of the command file for command line arguments and further examples.
2. This case was run with fourth-order accuracy.

2.4 Scattering of a plane wave from a two-dimensional dielectric cylinder

The command file `dielectricCyl.cmd` can be used to compute the scattering of a plane wave from a dielectric cylinder. The exact solution is known for this case and the errors in the numerical solution are determined while the solution is being computed.
(1) Generate the grid using the command file `innerOuter.cmd` in `Overture/sampleGrids`:

```
ogen -noplot innerOuter -factor=4 -order=4 -deltaRad=.5 -interp=e -name="innerOutere4.order4.hdf"
```

(2) Run cgmx:

```
cgmx dielectricCyl -g=innerOutere4.order4.hdf -kx=2 -eps1=.25 -eps2=1. -go=halt
```

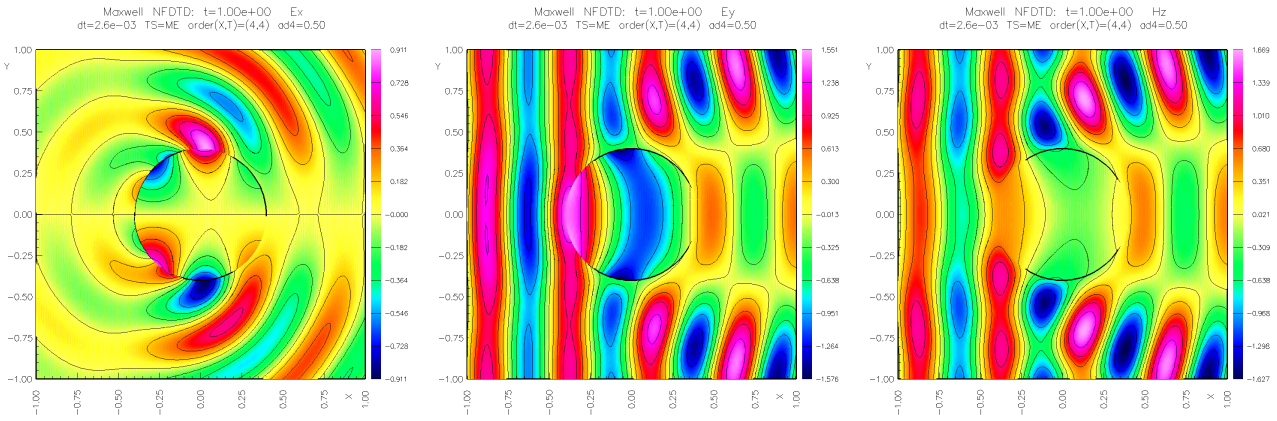

Figure 3: Scattering of a plane wave from a dielectric cylinder. Computed solution at $t = 1.0$, E_x, E_y and H_z.

Notes:

1. See the comments at the top of the command file for command line arguments and further examples.
2. This case was run with fourth-order accuracy.

2.5 Scattering of a plane wave from a two-dimensional dielectric cylinder - Yee scheme

The command file `dielectricCyl.cmd` can be used to compute the scattering of a plane wave from a dielectric cylinder using the Yee scheme. The exact solution is known for this case and the errors in the numerical solution are determined while the solution is being computed.
(1) Generate the grid using the command file `bigSquare.cmd` in `Overture/sampleGrids`:

```
ogen noplot bigSquare -factor=8 -xa=-1. -xb=1. -ya=-1. -yb=1. -name="bigSquareSize1f8.hdf"
```

(2) Run cgmx:

```
cgmx dielectricCyl -g=bigSquareSize1f8.hdf -kx=2 -eps1=.25 -eps2=1. -method=Yee -errorNorm=2 -go=halt
```

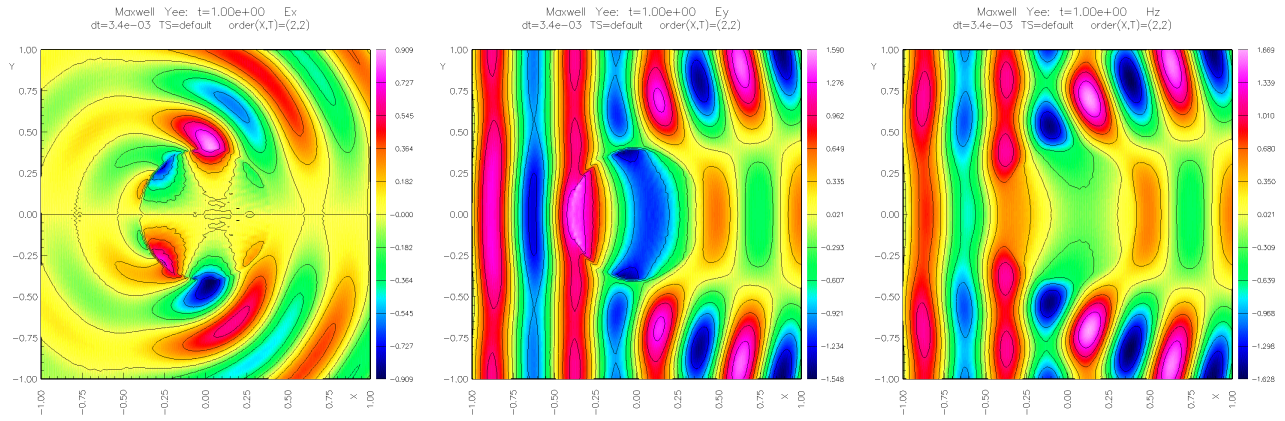

Figure 4: Scattering of a plane wave from a dielectric cylinder using the Yee scheme. The solution is computed on a single Cartesian grid and the cylinder is represented in a stair-step fashion. Computed solution at $t = 1.0$, E_x, E_y and H_z.

Notes:

1. The material parameters are chosen using ...

2.6 Scattering of a plane wave from a three-dimensional dielectric sphere

The command file `dielectricCyl.cmd` (yes, the same file as for the 2d dielectric cylinder) can be used to compute the scattering of a plane wave from a dielectric cylinder. The exact solution is known for this case and the errors in the numerical solution are determined while the solution is being computed.
(1) Generate the grid using the command file `solidSphereInABox.cmd` in `Overture/sampleGrids`:

```
ogen noplot solidSphereInABox -order=4 -interp=e -factor=2
```

(2) Run cgmx:

```
cgmx dielectricCyl -cyl=0 -g=solidSphereInABoxe2.order4 -kx=1 -eps1=.25 -eps2=1. -go=halt -tp=.01
```

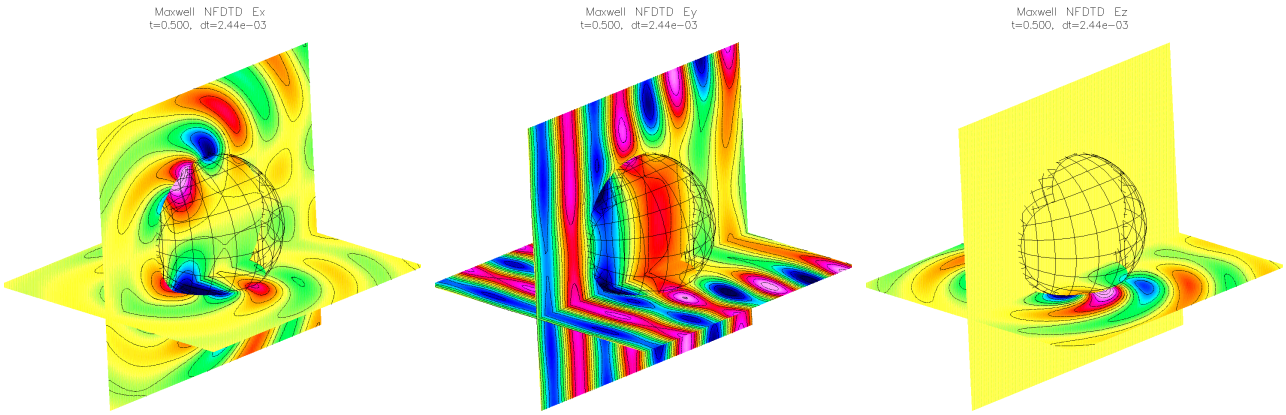

Figure 5: Scattering of a plane wave from a dielectric sphere. Computed solution at $t = 1.0$, E_x, E_y and H_z.

Notes:

1. See the comments at the top of the command file for command line arguments and further examples.
2. This case was run with fourth-order accuracy.

2.7 Diffraction from a two-dimensional knife edge slit

The command file `knifeEdgel.cmd` can be used to compute the scattering of a plane wave from narrow slit.
(1) Generate the grid using the command file `innerOuter.cmd` in `Overture/sampleGrids`:

```
ogen -noplot knifeEdge -interp=e -factor=2 -order=4 -yTop=1.1 -name="knifeSlit2.order4.hdf"
```

(2) Run cgmx:

```
cgmx knifeEdge -g=knifeSlit2.order4 -kx=8 -tp=.05 -tf=2. -plotIntensity=1 -go=halt
```

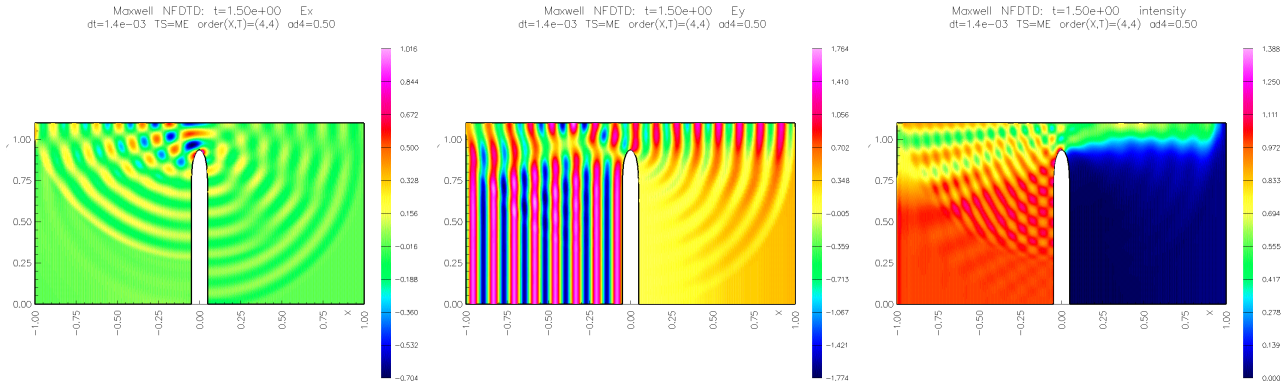

Figure 6: Scattering of a plane wave from a slit. Computed solution at $t = 1.5$, E_x, E_y and intensity.

Notes:

1. Comment on initial conditions and initial conditions bounding box ...
2. Adjust boundaries for incident field ...

2.8 Scattering of a plane wave by a two-dimensional triangular body

The command file `scat.cmd` can be used to compute the scattering of a plane wave from a body.
(1) Generate the grid using the command file `triangleArg.cmd` in `Overture/sampleGrids`:

```
ogen noplot triangleArg -factor=8 -order=4 -interp=e
```

(2) Run cgmx:

```
cgmx scat -g=trianglee8.order4.hdf -bg=backGround
```

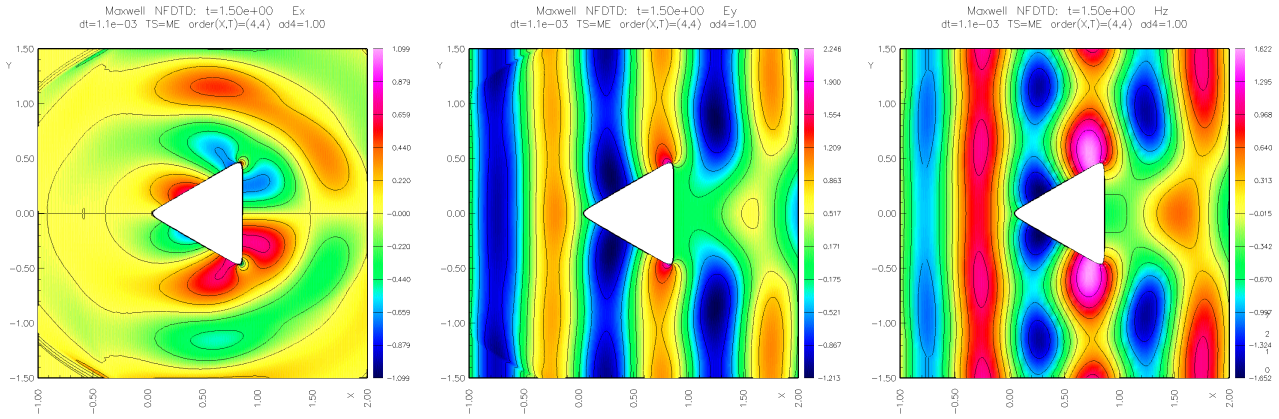

Figure 7: Scattering of a plane wave from a triangular body. Computed solution at $t = 1.5$, E_x, E_y and H_z. The initial startup wave-front can be seen leaving the domain in the upper left and lower left.

Notes:

1. Computes the scattered field directly using the `planeWaveBoundaryForcing` option.
2. Boundary conditions on the outer square use the `abcEM2` far-field conditions.

2.9 Comparing far-field boundary conditions

The command file `rbc.cmd` can be used to compare far-field boundary conditions. A Gaussian source 5.1 is placed in the middle of square or box and far field boundary conditions are applied to all boundaries. For comparison we also show results when the outer boundary is taken as a perfect electrical conductor (which results in large reflections).
(1) Generate the grids (command files in Overture/sampleGrids)

```
ogen -noplot squareArg -order=4 -nx=128
ogen noplot boxArg -order=4 -xa=-1. -xb=1. -ya=-1. -yb=1. -za=-1. -zb=1. ...
                   -factor=4 -name="boxLx2Ly2Lz2Factor4.order4.hdf"
```

(2a) Run cgmx with the square grid and different boundary conditions:

```
cgmx rbc -g=square128.order4.hdf -x0=.5 -y0=.5 -rbc=abcEM2 -go=halt
cgmx rbc -g=square128.order4.hdf -x0=.5 -y0=.5 -rbc=abcPML -pmlWidth=21 -pmlStrength=50. -go=halt
cgmx rbc -g=square128.order4.hdf -x0=.5 -y0=.5 -rbc=perfectElectricalConductor -go=halt
```

(2b) Run cgmx with the box grid and different boundary conditions:

```
cgmx rbc -g=boxLx2Ly2Lz2Factor4.order4.hdf -rbc=abcEM2 -x0=0. -y0=0. -z0=0. -go=halt
cgmx rbc -g=boxLx2Ly2Lz2Factor4.order4.hdf -rbc=abcPML -x0=0. -y0=0. -z0=0. -pmlWidth=11 -go=halt
cgmx rbc -g=boxLx2Ly2Lz2Factor4.order4.hdf -rbc=perfectElectricalConductor -x0=0. -y0=0. -z0=0. -go=halt
```

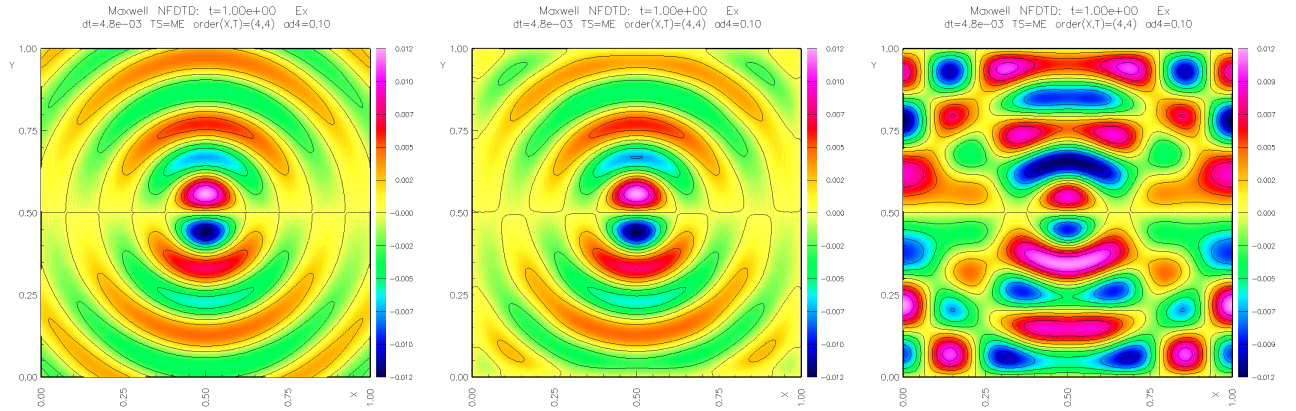

Figure 8: Gaussian source in a square with different boundary conditions. E_x at time $t = 1.0$. Left bc=`abcEM2`, middle: bc=`abcPML` and right bc=`perfectElectricalConductor`.

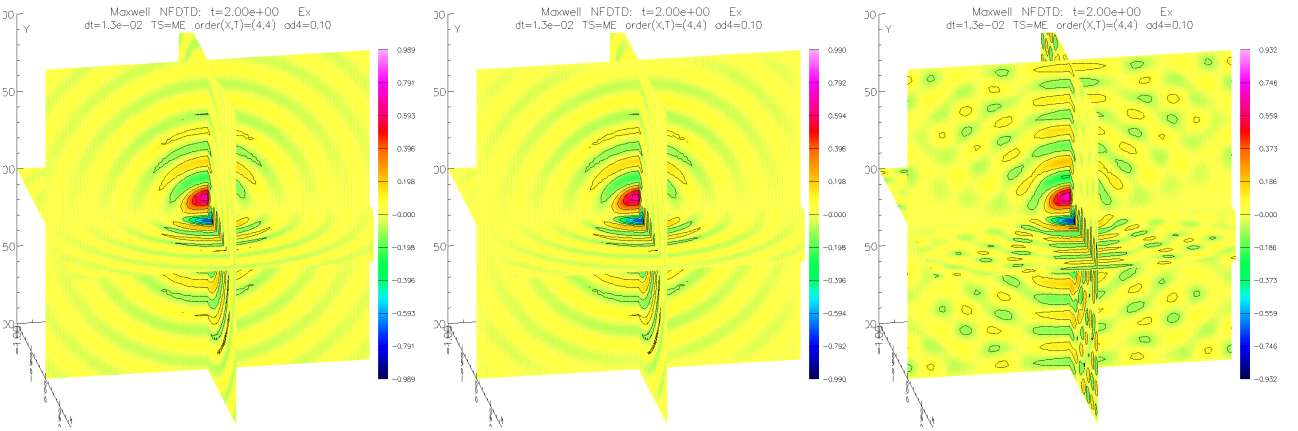

Figure 9: Gaussian source in a box with different boundary conditions. E_x at time $t = 1.0$. Left bc=`abcEM2`, middle: bc=`abcPML` and right bc=`perfectElectricalConductor`.

Notes:

1. Note that for the PML boundary condition the solution is damped in a region next to the boundary (set by the option -pmlWidth=*number-of-lines*). This explains why the solution decays near the boundaries.

2.10 Transmission of a plane wave through a bumpy glass-air interface

The command file `afm.cmd` can be used compute the propagation of a plane wave through the two-dimensional interface between two materials (air and glass).
(1) Generate the grids (command files in Overture/sampleGrids)

```
ogen noplot afm -interp=e -order=4 -factor=4
```

(2) Run cgmx (specifying ϵ in the two domains and the wave number k_y of the incident field),

```
cgmx afm -g=afme4.order4.hdf -eps1=2.25 -eps2=1. -ky=20 -diss=4. -tf=1.4 -tp=.2 -go=halt
```

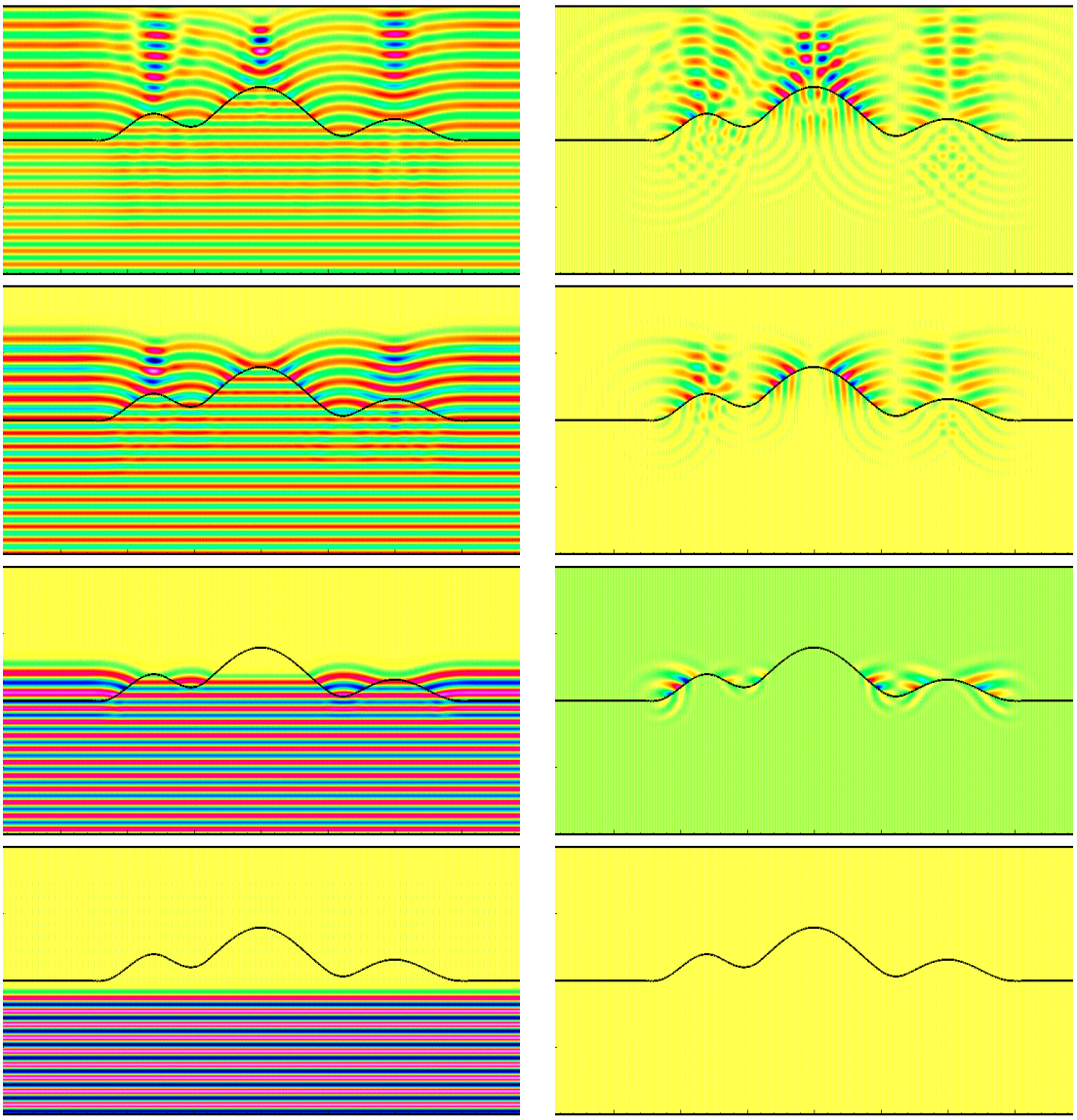

Figure 10: Transmission of a plane wave through a bumpy glass air interface. Left E_x and right E_y at times (bottom to top) $t = 0$, $t = 0.2$, $t = 0.4$ and $t = 0.6$

Notes:

1. This example uses the plane wave initial condition with the initial condition bounding box.
2. The speed of light in the lower glass region is $c_g = 1/\sqrt{2.25} = 2/3$ compared to the speed of light $c_a = 1$ in the upper air domain.

2.11 Scattering of plane wave from an array of dielectric cylinders

The command file `lattice.cmd` can be used to compute the scattering of a plane wave from an array of dielectric cylinders. A plane wave with wave number k_x moves from left to right past an array of dielectric cylinders. Periodic boundary conditions are imposed on the top and bottom.
(1) Generate the grid: (using the command file in `Overture/sampleGrids`):

```
ogen -noplot lattice -order=4 -interp=e -nCylx=3 -nCyly=3 -factor=2 -name="lattice3x3yFactor2.order4.hdf"
```

(2) Run cgmx:

```
cgmx -noplot lattice -g=lattice3x3yFactor4.order4 -eps1=.25 -eps2=1. -kx=4 ...
                     -plotIntensity=1 -xb=-2. -go=halt
```

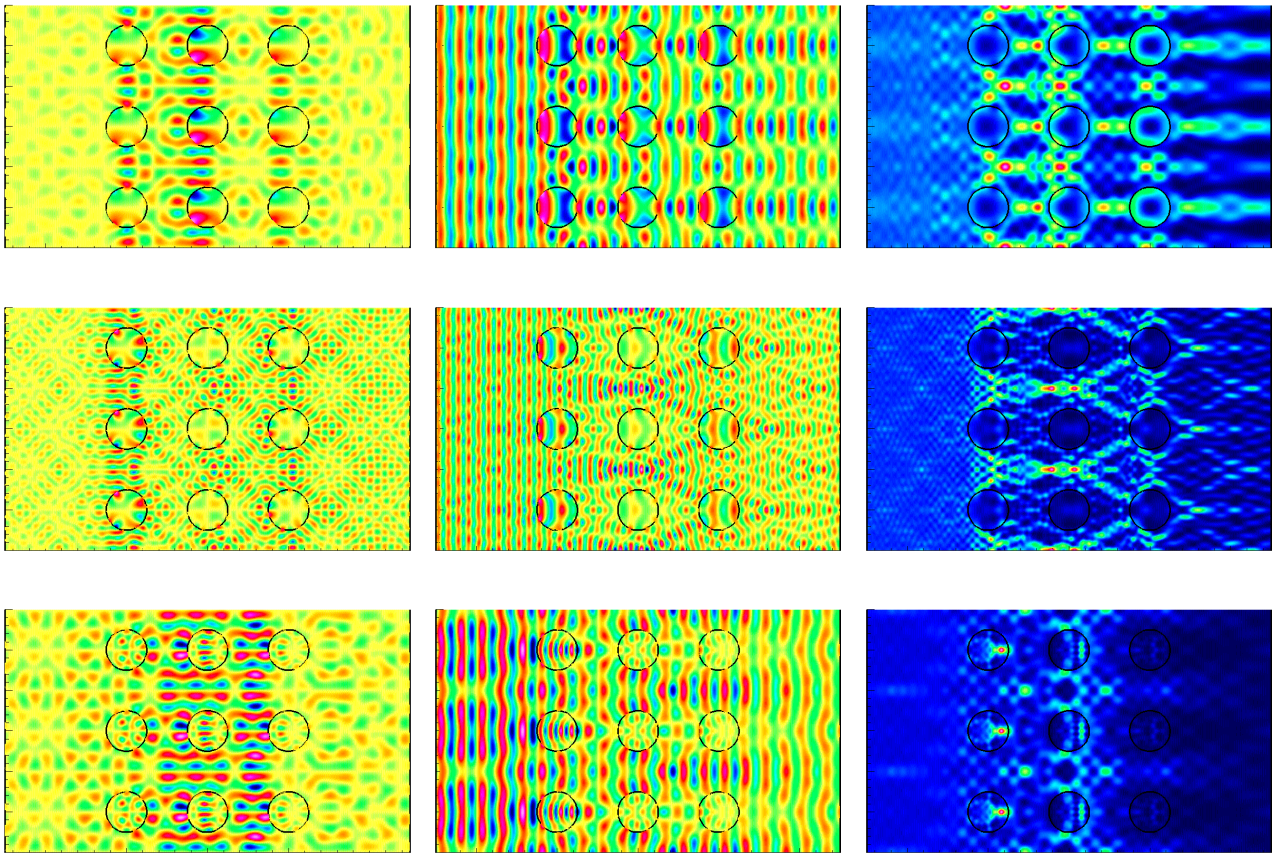

Figure 11: Scattering of a plane wave from an array of dielectric cylinders. Computed solution at $t = 6.0$, E_x, E_y and Intensity. Top row: $k_x = 4$, $eps_1 = 0.25$. Middle row: $k_x = 8$, $eps_1 = 0.25$. Bottom row: $k_x = 4$, $eps_1 = 4.0$.

3 Boundary Conditions

The general format for setting a boundary condition in a command file is

```
bc: name=bcName
```

where "name" specifies the name of a grid face, or grid and "bcName" is the name of the boundary condition (given below),

```
name = [all][gridName][gridName([0|1],[0|1|2])]
bcName=[dirichlet][perfectElectricalConductor][planeWaveBoundaryCondition][abcEM2]...
```

Here are some examples of setting boundary conditions in a command file,

```
bc: all=perfectElectricalConductor
bc: backGround=abcEM2
bc: backGround(0,0)=planeWaveBoundaryCondition
bc: square(1,0)=planeWaveBoundaryCondition
```

1. Note that *backGround* and *square* are the names of the grids (specified when the grid was constructed with Ogen).
2. Note that *backGround(0,0)* specifies the *left* face of the grid *backGround* using the standard Overture conventions for specifying the face of a grid as *gridName(side,axis)*.
3. Note that the special name *all* means apply the boundary condition to all faces of all grids.
4. The last boundary condition given to a face is the one that will be used.

The following boundary conditions are (more or less) available with cgmx,

periodic : chosen automatically to match the grid created with Ogen.

dirichlet : a non-physical boundary condition where all components of the field are set to some known function. For example, the known function could be an exact solution or a twilight-zone function.

perfectElectricalConductor : PEC boundary condition, see Section 3.1.

perfectMagneticConductor : PMC boundary condition (NOT implemented yet).

planeWaveBoundaryCondition : solution on the boundary is set equal to a plane wave solution, see Section 3.2.

symmetry : apply a symmetry boundary condition.

interfaceBoundaryCondition : for the interface between two regions with different properties

abcEM2 : absorbing BC, Engquist-Majda order 2, see Section 3.3.

abcPML : perfectly matched layer far field condition, see Section 3.4.

rbcNonLocal : radiation BC, non-local

rbcLocal : radiation BC, local

3.1 Perfect electrical conductor boundary condition

The perfect electrical conductor boundary condition sets the tangential components of the electric field to zero

$$\boldsymbol{\tau}_m \cdot \mathbf{E} = 0, \qquad \text{for } \mathbf{x} \in \partial\Omega_{\text{pec}}. \tag{1}$$

Here $\boldsymbol{\tau}_m$, $m = 1, 2$, denote the tangent vectors to the boundary surface, $\partial\Omega_{\text{pec}}$.

3.2 Plane wave boundary condition

The plane wave boundary condition set the electric (and magnetic) fields on the boundary to the plane wave solution (9)-(10) defined in Section 4.1,

$$\mathbf{E}(\mathbf{x},t) = \mathbf{E}_{\text{pw}}(\mathbf{x},t), \qquad \text{for } \mathbf{x} \in \partial\Omega_{\text{pw}}, \tag{2}$$
$$\mathbf{H}(\mathbf{x},t) = \mathbf{H}_{\text{pw}}(\mathbf{x},t), \qquad \text{for } \mathbf{x} \in \partial\Omega_{\text{pw}}. \tag{3}$$

3.3 Engquist-Majda absorbing boundary conditions

The boundary condition `abcEM2` uses the Engquist-Majda absorbing boundary condition (defined here of a boundary $x = \text{constant}$),

$$\partial_t \partial_x u = \alpha \partial_x^2 u + \beta(\partial_y^2 + \partial_z^2)u \tag{4}$$

With $\alpha = c$ and $\beta = \frac{1}{2}c$, this gives a *second-order accurate* approximation to a pseudo-differential operator that absorbs outgoing traveling waves. Here u is any field which satisfies the second-order wave equation.

3.4 Perfectly matched layer boundary condition

The boundary condition `abcPML` imposes a perfectly matched layer boundary condition. With this boundary condition, auxiliary equations are solved over a layer (of some number of specified grid points) next to the boundary. The PML equations we solve are those suggested in Hagstrom [3] and given by (defined here for a boundary $x = \text{constant}$), (*check me*)

$$u_{tt} = c^2\Big(\Delta u - \partial_x v - w\Big), \tag{5}$$
$$v_t = \sigma(-v + \partial_x u), \tag{6}$$
$$w_t = \sigma(-w - \partial_x v + \partial_x^2 u). \tag{7}$$

Here u is any field which satisfies the second-order wave equation and v and w are auxiliary variables that only live in the layer domain. The PML damping function $\sigma_1(\xi)$ is given by

$$\sigma(\xi) = a\xi^p \tag{8}$$

where a is the strength, p is the power and where ξ varies from 0 to 1 through the layer.

4 Initial conditions

The following initial conditions are currently available with cgmx,

planeWaveInitialCondition : See Section 4.1.

gaussianPlaneWave :

gaussianPulseInitialCondition :

squareEigenfunctionInitialCondition :

annulusEigenfunctionInitialCondition :

zeroInitialCondition :

planeWaveScatteredFieldInitialCondition :

planeMaterialInterfaceInitialCondition :

gaussianIntegralInitialCondition : from Tom Hagstrom

twilightZoneInitialCondition : for use with twilight-zone forcing.

userDefinedInitialConditions : see Section 4.3

To do: The file `userDefinedInitialConditions.C` can be used to define new initial conditions for use with cgmx.

4.1 The plane-wave initial condition

The plane wave initial condition is defined from the plane wave solution

$$\mathbf{E} = \sin(2\pi(\mathbf{k}\cdot\mathbf{x} - \omega t))\ \mathbf{a}, \tag{9}$$

$$\mathbf{H} = \sin(2\pi(\mathbf{k}\cdot\mathbf{x} - \omega t))\ \mathbf{b}, \tag{10}$$

$$\omega = c_m|\mathbf{k}|, \tag{11}$$

$$c_m = \frac{1}{\sqrt{\epsilon\mu}} \qquad \text{(speed of light in the material)}\ . \tag{12}$$

where $\mathbf{k} = (k_x, k_y, k_z)$, $|\mathbf{k}| = \sqrt{k_x^2 + k_y^2 + k_z^2}$, $\mathbf{a} = (a_x, a_y, a_z)$, and $\mathbf{b} = (b_x, b_y, b_z)$ satisfy

$$\mathbf{k}\cdot\mathbf{a} = 0, \quad \mathbf{k}\cdot\mathbf{b} = 0, \quad \text{(from } \nabla\cdot\mathbf{E} = 0 \text{ and } \nabla\cdot\mathbf{H} = 0), \tag{13}$$

$$\mathbf{b} = \sqrt{\frac{\epsilon}{\mu}}\ \frac{\mathbf{k}\times\mathbf{a}}{|\mathbf{k}|}, \quad \text{(from } \mu\mathbf{H}_t = -\nabla\times\mathbf{E}). \tag{14}$$

Thus given $\mathbf{a}$ with $\mathbf{k}\cdot\mathbf{a} = 0$, $\mathbf{b}$ is determined by (14). The parameters in the plane wave initial condition can be set with the commands:

```
kx,ky,kz $kx $ky $kz
plane wave coefficients $ax $ay $az $epsPW $muPW
```

Here `epsPW` and `muPW` (which could be different from the values of ϵ and μ used for the simulation) are used to define $c_{\rm pw} = 1/\sqrt{\epsilon_{\rm pw}\cdot\epsilon_{\rm pw}}$.

4.2 The initial condition bounding box

The initial condition bounding box can be used to restrict the region where the initial condition is assigned (see the example in Section 2.7). The command to define the box is

```
initial condition bounding box $xa $xb $ya $yb $za $zb
```

This bounding box may only currently work with the plane wave initial condition.

The plane-wave initial conditions are turned off in a smooth way at the boundary of the box using the function

$$g(\mathbf{x}) = \frac{1}{2}(1 - \tanh(\beta 2\pi(k_x(x - x_0) + k_y(y - y_0)))). \tag{15}$$

where x_0 or y_0 a point on the edge of the box (usually the transition only happens on one face of the box and it is assumed here that the plane wave is parallel to this face). The coefficient β in this function can be set with the command

```
bounding box decay exponent $beta
```

4.3 User defined initial conditions

The file `userDefinedInitialConditions.C` can be used to define new initial conditions for use with Cgmx. Here are the steps to take to add your own initial conditions:

1. Edit the file `cg/mx/src/userDefinedInitialConditions.C`.
2. Add a new option to the function `setupUserDefinedInitialConditions()`. Follow one of the previous examples.
3. Implement the initial condition option in the function `userDefinedInitialConditions(...)`.
4. Type *make* from the `cg/mx` directory to recompile Cgmx.
5. Run Cgmx and choose the *userDefinedInitalConditions* option from the *forcing options...* menu. (see the example command file `mx/cmd/userDefinedInitialConditions.cmd`).

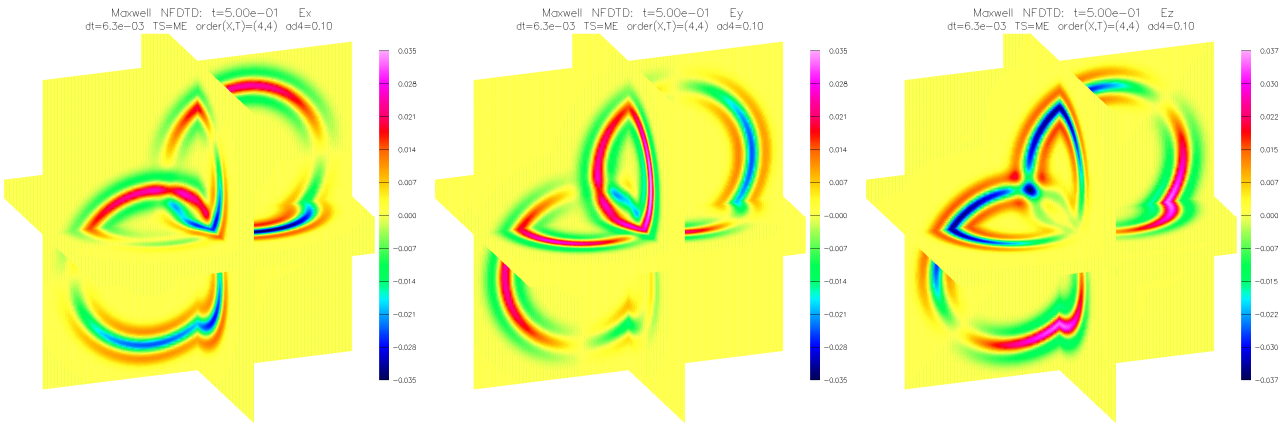

Figure 12: User defined initial conditions. Results for two Gaussian pulses in a box.

5 Forcing functions

The forcing functions are added to the right-hand side of Maxwell's equations in second-order form,

$$\partial_t^2 \mathbf{E} = c^2 \ \Delta \mathbf{E} + \mathbf{F}_{\mathbf{E}}(\mathbf{x}, t), \tag{16}$$
$$\partial_t^2 \mathbf{H} = c^2 \ \Delta \mathbf{H} + \mathbf{F}_{\mathbf{H}}(\mathbf{x}, t). \tag{17}$$

The following forcing options are currently available with cgmx,

noForcing : the default is to have no forcing.

gaussianSource : See Section 5.1.

twilightZoneForcing : The forcing to make the twilight-zone function an exact solution.

planeWaveBoundaryForcing : The scattered field from an incident plane wave can be computed directly using this forcing which is added to the right-hand side of the PEC boundary condition.

gaussianChargeSource : See Section 5.2.

userDefinedForcing : See Section 5.3.

5.1 Gaussian source

The Gaussian source $\mathbf{F_E}$ in three dimensions is defined by

$$\begin{aligned} g(\mathbf{x}, t) &= \beta^2 \cos(2\pi\omega(t - t_0)) \ \exp\{-\beta|\mathbf{x} - \mathbf{x}_0|^2\}, \\ F_{E_x} &= ((z - z_0) - (y - y_0)) \ g(\mathbf{x}, t), \\ F_{E_y} &= ((x - x_0) - (z - z_0)) \ g(\mathbf{x}, t), \\ F_{E_z} &= ((y - y_0) - (x - x_0)) \ g(\mathbf{x}, t). \end{aligned}$$

Here $|\mathbf{x}|$ denotes the usual length of a vector, $|\mathbf{x}|^2 = x_1^2 + x_2^2 + x_3^2$.
The Gaussian source in two-dimensions is defined in a somewhat different fashion (for some reason?)

$$\begin{aligned} g(\mathbf{x}, t) &= 2\beta \sin(2\pi\omega(t - t_0)) \ \exp\{-\beta|\mathbf{x} - \mathbf{x}_0|^2\}, \\ F_{H_z} &= 2\pi\omega \cos(2\pi\omega(t - t_0)) \ \exp\{-\beta|\mathbf{x} - \mathbf{x}_0|^2\}, \\ F_{E_x} &= -(y - y_0) \ g(\mathbf{x}, t), \\ F_{E_y} &= (x - x_0) \ g(\mathbf{x}, t). \end{aligned}$$

5.2 Gaussian charge source

The Gaussian charge source is defined by a moving charge density

$$\rho(\mathbf{x}, t) = a \exp\{-\big[\beta|(\mathbf{x} - \mathbf{x}_0) - \mathbf{v}t|\big]^p\}. \tag{18}$$

Here $\mathbf{x}_0$ is the initial location of the charge, $\mathbf{v}$ is the velocity of the charge, a is the strength and β and p define the shape of the pulse in space. The forcing functions for Maxwell's equations are then defined from

$$\partial_t^2 \mathbf{E} = c^2[\Delta \mathbf{E} - \nabla(\rho/\epsilon) - \mu \mathbf{J}_t], \tag{19}$$
$$\partial_t^2 \mathbf{H} = c^2[\Delta \mathbf{H} + \nabla \times \mathbf{J}], \tag{20}$$
$$\mathbf{J} = \rho \mathbf{v}, \tag{21}$$

which gives

$$\mathbf{F_E} = c^2[-\nabla(\rho/\epsilon) - \mu \mathbf{J}_t]. \tag{22}$$

5.3 User defined forcing

The file `userDefinedForcing.C` can be used to define new forcing functions for use with Cgmx. Here are the steps to take to add your own forcing:

1. Edit the file `cg/mx/src/userDefinedForcing.C`.
2. Add a new option to the function `setupUserDefinedForcing()`. Follow one of the previous examples.
3. Implement the forcing option in the function `userDefinedForcing(...)`.
4. Type *make* from the `cg/mx` directory to recompile Cgmx.
5. Run Cgmx and choose the *userDefinedForcing* option from the *forcing options...* menu. (see the example command file `mx/cmd/userDefinedForcing.cmd`).

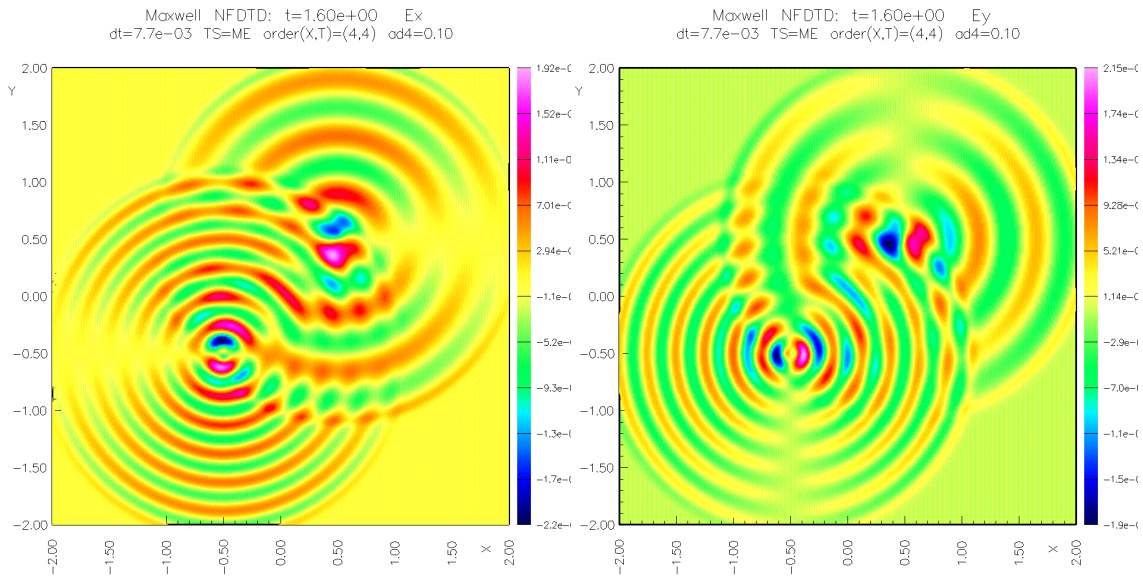

Figure 13: User defined forcing example showing results from specifying two Gaussian sources. Left E_x and right E_y

6 Options

6.1 Options affecting the scheme

cfl *value* : set the CFL number. The schemes are usually stable to CFL=1. The default is CFL=.9.

order of dissipation [2|4|6] : set the order of the dissipation.

dissipation *value* : set the coefficient of the dissipation (e.g. 1). Dissipation is usually needed on overlapping grids.

coefficients ϵ μ *gridName*: specify the values of ϵ and μ on a grid.

adjustFarFieldBoundariesForIncidentField [0|1] [*gridName*|*all*] : subtract off the plane-wave solution before applying the far-field boundary condition.

6.2 Run time options

tFinal *value* : solve the equations to this time.

tPlot *value* : time increment to save results and/or plot the solution.

debug *value* : an integer bit flag that turns on debugging information. For example, set debug=1 for some info, debug=3 (=1+2) for some more, debug=7 (=1=2+4)for even more.

6.3 Plotting options

error norm [0|1|2] : compute errors in the max-norm, L_1-norm or L_2-norm. (*check me*)

plot scattered field [0|1] : when computing the scattered field directly use this option to plot the scattered field.

plot total field [0|1] : when computing the scattered field directly use this option to plot the total field (i.e. add in the plane wave solution before plotting).

plot errors [0|1] :

check errors [0|1] :

plot intensity [0|1] : plot the intensity (really only make senses for fields that are time-harmonic).

plot harmonic E field [0|1] : plot the components of the complex harmonic fields (for problems that are time-harmonic).

6.4 Output options

specify probes : specify a list of probe locations as x, y, z values, one per line, finishing the list with 'done'. The solution values at these locations (actually the closest grid point to each location, with these values written to the screen) are written to a text file whose default name is "probeFile.dat". In the following example we specify two probe locations, $(.2, .3, .1)$ and $(.4, .6, .3)$,

```
specify probes
  .2 .3 .1.
  .4 .6 .3
done
```

Each line of the probe file contains the time followed by the three components of **E** (or in two-dimensions E_x, E_y and H_z) for each probe location. For example, with two probes specified the file in three dimensions would contain data of the form

```
t1 Ex11 Ey11 Ez11  Ex12 Ey12 Ez12
t2 Ex21 Ey21 Ez21  Ex22 Ey22 Ez22
...
```

probe file: *name* : specify the name of the probe file.

probe frequency *value* : specify the frequency at which values are saved in the probe file. For example, if the probe frequency is set to 2 then the solution will be saved every 2nd time step to the probe file.

7 Maxwell's Equations

The time dependent Maxwell's equations for linear, isotropic and non-dispersive materials are

$$\partial_t \mathbf{E} = \frac{1}{\epsilon}\nabla \times \mathbf{H} - \frac{1}{\epsilon}\mathbf{J}, \tag{23}$$

$$\partial_t \mathbf{H} = -\frac{1}{\mu}\nabla \times \mathbf{E}, \tag{24}$$

$$\nabla \cdot (\epsilon \mathbf{E}) = \rho, \quad \nabla \cdot (\mu \mathbf{H}) = 0, \tag{25}$$

Here $\mathbf{E} = \mathbf{E}(\mathbf{x}, t)$ is the electric field, $\mathbf{H} = \mathbf{H}(\mathbf{x}, t)$ is the magnetic field, $\rho = \rho(\mathbf{x}, t)$ is the electric charge density, $\mathbf{J} = \mathbf{J}(\mathbf{x}, t)$ is the electric current density, $\epsilon = \epsilon(\mathbf{x})$ is the electric permittivity, and $\mu = \mu(\mathbf{x})$ is the magnetic permeability. This first-order system for Maxwell's equations can also be written in a second-order form. By taking the time derivatives of (24) and (23) and using (25) it follows that

$$\epsilon\mu\ \partial_t^2 \mathbf{E} = \Delta \mathbf{E} + \nabla\Big(\nabla \ln \epsilon\ \cdot \mathbf{E}\Big) + \nabla \ln \mu \times \Big(\nabla \times \mathbf{E}\Big) - \nabla(\frac{1}{\epsilon}\rho) - \mu \partial_t \mathbf{J}, \tag{26}$$

$$\epsilon\mu\ \partial_t^2 \mathbf{H} = \Delta \mathbf{H} + \nabla\Big(\nabla \ln \mu\ \cdot \mathbf{H}\Big) + \nabla \ln \epsilon \times \Big(\nabla \times \mathbf{H}\Big) + \epsilon \nabla \times (\frac{1}{\epsilon}\mathbf{J}). \tag{27}$$

It is evident that the equations for the electric and magnetic field are decoupled with each satisfying a vector wave equation with lower order terms. In the case of constant μ and ϵ and no charges, $\rho = \mathbf{J} = 0$, the equations simplify to the classical second-order wave equations,

$$\partial_t^2 \mathbf{E} = c^2\ \Delta \mathbf{E}, \qquad \partial_t^2 \mathbf{H} = c^2\ \Delta \mathbf{H} \tag{28}$$

where $c^2 = 1/(\epsilon\mu)$. There are some advantages to solving the second-order form of the equations rather than the first-order system. One advantage is that in some cases it is only necessary to solve for one of the variables, say $\mathbf{E}$. If the other variable, $\mathbf{H}$ is required, it can be determined by integrating equation (24) as an ordinary differential equation with known $\mathbf{E}$. Alternatively, as a post-processing step $\mathbf{H}$ can be computed from an elliptic boundary value problem formed by taking the curl of equation (23). Another advantage of the second-order form, which simplifies the implementation on an overlapping grid, is that there is no need to use a staggered grid formulation. Many schemes approximating the first order system (24-25) rely on a staggered arrangement of the components of $\mathbf{E}$ and $\mathbf{H}$ such as the popular Yee scheme [7] for Cartesian grids.

8 Acknowledgments

Thanks to Kyle Chand who contributed to parts of Cgmx. Thanks to Tom Hagstrom for contributing some far field boundary condition routines.

This work was performed under the auspices of the U.S. Department of Energy (DOE) by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and by DOE contracts from the ASCR Applied Math Program.

References

[1] D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented software system for solving partial differential equations in serial and parallel environments*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, 1997, p. .

[2] D. L. Brown, W. D. Henshaw, and D. J. Quinlan, *Overture: An object oriented framework for solving partial differential equations*, in Scientific Computing in Object-Oriented Parallel Environments, Springer Lecture Notes in Computer Science, 1343, 1997, pp. 177–194.

[3] T. Hagstrom, *Radiation boundary conditions for the numerical simulation of waves*, Acta Numerica, 8 (1999), pp. 47–106.

[4] W. D. Henshaw, *Overture: An object-oriented system for solving PDEs in moving geometries on overlapping grids*, in First AFOSR Conference on Dynamic Motion CFD, June 1996, L. Sakell and D. D. Knight, eds., 1996, pp. 281–290.

[5] ——, *A high-order accurate parallel solver for Maxwell's equations on overlapping grids*, SIAM J. Sci. Comput., 28 (2006), pp. 1730–1765. publications/henshawMaxwell2006.pdf.

[6] ——, *Cgmx reference manual: An Overture solver for Maxwell's equations on composite grids*, Software Manual to appear, Lawrence Livermore National Laboratory, 2012. publications/CgmxReferenceGuide.pdf.

[7] K. S. Yee, *Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media*, IEEE Transactions on Antennas and Propogation, 14 (1966), pp. 302–307.