User Manual

June 15, 2011

1 What is B-CALM?

B-CALM or Belgium California Light Machine is a super fast 3D GPU based Finite Difference Time Domain simulator. It utilizes NVDIA Graphical Processing Units with a compute capability of 1.3 or higher. B-CALM has a simulation speed between 1.0e10 and 1.5e10 cell-updates per minute and GPU unit. B-CALM is interfaced with Matlab for both input and output. It incorporates the following features

- Variable meshing
- Multiple-pole Lorentz materials
- CPML-Absorbing conditions
- User defined exitation
- Built in exitation(Gaussian beam at any angle, plane wave)
- Comprehensive input output structure in Matlab

This manual assumes that the user is acquinted with FDTD and electromagnetic simulations in general.

2 Installation

2.1 Installation of the GPU unit

2.1.1 Download and install the drivers for the GPU compatible card

See: *http://www.nvidia.com/Download/index.aspx?lang=en-us* for more info.

2.1.2 Download and install the latest CUDA toolkit

See: *http://developer.nvidia.com/object/cuda_3_1_downloads.html* for more info.

2.2 Installation of B-CALM

2.2.1 Download the latest B-CALM version from the Stanford servers

http://www.stanford.edu/~pwahl/BCALM/latest_backup.tar.gz/

2.2.2 Untar the downloaded file in the directory where you want to install it

ex: ~/BCALM/

2.2.3 Create the target directory ./obj where the Make will dump the compiled objects and the linked program

mkdir ~/BCALM/obj

2.2.4 Make the program

~/BCALM/CUDA_code/make

It is possible that some libraries are not installed on your computer by default. Typically the following ones are not installed by default

1. *sudo apt-get install apt-file sudo apt-file update*
2. *sudo apt-get install g++*
3. *sudo apt-get install fftw3-dev*
4. *sudo apt-get install libhdf5-serial-dev*
5. *sudo apt-get install libhdf5-mpich-dev*
6. *sudo apt-get install libhdf5-serial-1.6.4-0c2*

If succesful the executable fdtd is created in the *obj* directory

3 Using B-CALM

3.1 Introduction

The compiled program *fdtd* takes a *HDF5-file* as input that is generated by Matlab that contains all the information relevant to the simulation.This HDF5 file contains the following information:

- The amount timesteps
- The utilized mesh
- The optical properties of the materials used in the simulation
- The structres within the mesh utilizing those materials
- The boundary conditions of the simulation space
- The sources and their properties
- The zones within the simulation space that need to be outputted.

Once the simultion is finished *fdtd* creates two files per requested per output zone. One containing the timetraces, the other containting their Fourier transform. These files can be read in Matlab in order to visualize and process the results. In this elementary manual we will go over each of these steps trough a illustratory exemples.

4 Example: Dielectric Cube

This example can be found as *./Matlab/Testcases/DielectricCube*. The goal of the simulation is to compute the fields of a dielectric cube in air illuminated by a Plane Wave or a Gaussian Beam. When creating a simulation in Matlab, there is one object in which all the information concerning the simulation is stacked. In this exemple it is called *GRID*.

4.1 Create the mesh

In this part we discretize a cubic part of space in a small fixed sized rectangles of a **fixed** size creating a mesh. How to create a mesh of **variable sized** rectangles for improved precision in will be illustrated in a later exemple. A object *simul* that contains the following fields is passed to the function *CreateConstantGrid*.

- x: Approximate total simulation size in the X direction
- y: Approximate total simulation size in the Y direction
- z: Approximate total simulation size in the Z direction
- dx: Cell size in the X direction
- dy: Cell size in the Y direction
- dz: Cell size in the Z direction
- pad: The size of the simulation space in the **X and the Y direction has to be divisible by 16.** When set to one a grid is padded so that this is true.

The mesh is stored in the Grid object as

- Grid.grid.x
- Grid.grid.y
- Grid.grid.z

and consists of an array of the size of individual cells in each direction. In this exemple, this is a constant array.

4.2 Defining the number of timesteps and temporal increment per timestep

The number of timesteps is the amount of iterartions *fdtd* will perform a field update. This number must be high enough so that all the fields have reasonably been absorbed in the absorbing boundaries.

Grid.info.tt=nsteps sets the number of timesteps.

Grid.info.dt=dt sets the temporal increment per timestep where

$$dt < \frac{CF}{c\sqrt{\frac{1}{dx^2} + \frac{1}{dy^2} + \frac{1}{dz^2}}}$$

with c the velocity of light and dx, dy and dz the cell size in the x y z direction. $CF < 1$ is the *Courant Factor* has to be smaller than one for stability.

4.3 Defining the materials

In this simulation we define two types of materials

1. **Ambient Material**: Is the material that fills the simulation space by default.

   Air.name='Air';

   Air.epsilon=1;

   Air.sigma=0;

   Air.ambient=1;

   LinearGrid=AddMat(LinearGrid,Air);

   Air is the name of the material *1* is the real part of the relative permitivitty, sigma its electrical conductivity. 'Ambient' sets that particular material as Ambient

2. **Regular Dielectric:** Is a material with a different refractive index than ambient material.

   Glass.name='Glass';

   Glass.ambient=0;

   Glass.epsilon=1.4^2;

   Grid=AddMat(Grid,Glass);

   where *Glass* is the name of the material 1.4^2 is the real part of the relative permitivitty.

4.4 Add structures

For every structure one has to define a *zone* object that defines the dimensions of a cube that contains the material with the following properties

- zone.x, zone.y, zone.z: Starting point of the material cube in the x, y and z direction in number of cells.
  - zone.x=1 adds a material starting from the first cell in the x direction.
- zone.dx, zone.dy, zone.dz: Size of the cube in the x ,y, and z direction.
  - zone.dx=0 adds 1 cell in the X direction
  - zone.dx='end' adds material until the simulation border.
- zone.name='Glass' Adds a the material Glass

grid=AddMatZone(grid,zone) adds the material to the grid.

4.5 Add Sources

In BCALM all sources are so called *soft sources* meaning that a they act as a source of electromagnetic radation in the simulation space,while remaining transparant. In other words, a source feeds a structure but doesn't change its optical properties.

In BCALM all sources are sinusoidal with a Gaussian envelope by

$$F_{source}(t) = |F_{in}| \sin(\omega t + \angle F_{in}) e^{\left[\frac{(t/t_s - \mu_t)}{\sigma_t}\right]^2} \quad (1)$$

where F_{in} is the complex amplitude of the sourced field, ω the angular frequency, t the time in seconds, μ_t the temporal mean of the gaussian envelope in timesteps and σ_t the standard deviation of the Gaussian envelope in timesteps.

4.5.1 Plane Wave

To add a plane wave source on has to define an object *sourcePW* that contains all the properties of the source and pass it to the function *Grid=DefineSpecialSource(Grid,sourcePW)* to add the source to the grid.

- sourcePW.x, sourcePW.y, sourcePW.z: Starting point of the source zone in the x,y,z direction.
- sourcePW.dx, sourcePW.dy, sourcePW.dz: Size of the exitation zone in the x,y,z direction.
- sourcePW.type : Sets the type of the source
  - constant: sources in the zone are in phase as required for a plane wave.
- sourcePW.omega: ω in (1).
- sourcePW.mut: μ_t in (1).
- sourcePW.sigmat: σ_t in (1).
- sourcePW .Ex, .Ey, .Ez,.Hx, .Hy, .Hz: Complex amplitude of the sourced field Ex, Ey, Ez, Hx, Hy, Hz.

4.5.2 Gaussian Beam

BCALM has a built in Gaussian beam exitation that can exite a Gaussian beam from an angle and with a certain bean waist onto a specific point in the simulation space. The spacial properties of the Gaussian beam are defined by following subseccquent equations.

$$F(z,r) = \frac{w_0}{w(z)} \exp\left(\frac{-r^2}{w^2(z)}\right) \exp\left(-ikz - ik\frac{r^2}{2R(z)} + i\zeta(z)\right) \tag{2}$$

$$w(z) = w_0\sqrt{1 + \left(\frac{z}{z_\mathrm{R}}\right)^2} \tag{3}$$

$$z_\mathrm{R} = \frac{\pi w_0^2}{\lambda} \tag{4}$$

$$\zeta(z) = \arctan\left(\frac{z}{z_\mathrm{R}}\right) \tag{5}$$

where $F(z,r)$ denotes the exited field, $w_0$the minimal beamwaist, z the distance parallel to the propagation direction and r the radial distance to the center of the beam. A Gaussian beam is exited from one or several planes in the simulation space.

To define a Gaussian Beam one has to define an object with the following additional properties upon the Plane Wave properties:

- source.x,.y,.z denotes the postion of the focus of the Gaussian Beam in the x, y and z direction.
- source.facex, .facey, .facez denotes, if used, the fixed coordinate in the x, y or z direction of the plane from wich the Gaussian beam is exited

  - $source.facex = 20$ means the Gaussian Beam is exited from the surface $x = 20$. The value of the Gaussian Beam is calculated and exited for *all* the cells on the plane $x = 20$. More than one surface should be used if the beam is exited at a shallow angle.

- source.type='gaussianbeam' to exite a Gaussian Beam
- source.w0: $w_0$in (2)
- source.n: Refractive index of the material in which the Gaussian Beam propagates.
- source.phi: Angle of the k vector of the Gaussian Beam to and the Z axis.
- source.tetha: Angle between the projection of the k vector of the Gaussian Beam on the XY plane and the X axis.

Grid=DefineSpecialSource(Grid,source) adds the Gaussian Beam source.

4.6 Define Abosorbing Boundary Conditions

In order to limit the simulation space with a reflectionless boundary BCALM has preprogrammed *Perfect Matched Layers* that absorb the incoming fields at angle with very little reflections.

To define a CPML one has to define an object with the follwing properties

- cpml.dx, .dy, .dz sets the thickness of the CPML layer in the x y and z direction. 8 to 10 cells are sufficient, 15 is the maximum.
- cpml.xneg, yneg, zneg: If one, adds a CPML on the left border in the x y or z direction.
- cpml.xpos, ypos, zpos: If one, adds a CPML on the right border in the x y or z direction.
- cpml.m: Order of the CPML, usually 4.
- cpml.amax: CPML specific, usually 0.2.
- cpml.kmax: CPML specific, usually 1.
- cpml.smax: CPML specific, usually 210000.

Grid=AddCPML(Grid,cpml) adds the CPML to the Grid.

4.7 Define Outputs

To save memory space and to keep output files relatively small BCALM requires the user to define the zones in the simulation space that need to be outputted. BCALM returns both, the timetraces and their Fourier transforms. In order to define an output, an object with following properties has to be defined.

- output.x, output.y, output.z: Starting point of the output zone in the x,y,z direction.
- output.dx, output.dy, output.dz: Size of the output zone in the x,y,z direction.
- output.foutstart: Lower boundary of the frequency interval of the output.
- output.foutstop: Upper boudary of the frequency interval of the output

- output.deltafmin: Minimal frequency resolution of the frequency output.
- output.deltaT: (Facultative) Number of timesteps between each output in time domain. If not set, minimal deltaT is taken to accomondate the required frequency bandwidth.
- output.name: Name of the output.
  - Filename of the file containing the timesteps '($output.name)'
  - Filename of the file containing the Fourier Transforms '($output.name)FFT'
- output.field: Sets the fields that need to be outputted
  - *output.field=[{'Ex'},{'Hy'},{'Hz'}]* outputs the Ex, Hy and Hz fields.

4.8 Launch the Simultation

4.8.1 In matlab

To launch a simulation one has to create an object with the following fields.

- SimulInfo.WorkPlace: This path is set as base path in Matlab.
- SimulInfo.SimlatorExec: Path to the executable *fdtd*.
- SimulInfo.infile: Path to the input HDF5 file to be created.
- SimulInfo.outfile: Path to the directory where the outputfiles are created.
- SimulInfo.PrintStep: (facultative) Number of timesteps every other which fdtd outputs the progress of the simulation.

LocalSimulate(Grid, SimulInfo) creates and executes the simulation.

4.8.2 By command line

To lauch a simulation by command line:

1. Create the input HDF5 file in Matlab
   (a) *hd5_export(infile,Grid)* creates the HDF5 file at location *inflile*
2. Exucute the command line with the right options
   (a) -i Input File (default is InputFile)
   (b) -o Output File (default is OutputFile)
   (c) -f Progress frequency updates (in time steps)
   (d) -F Generate FFT's
   (e) -T Generate Timesteps
3. Example: *fdtd -i Inputfile -o Outputfile -f 1500 -F -T*

4.9 Process the ouputs

Once the simulation is completed the outputs can be plotted and postprocessed using the Matlab function *[Data,pos]=PlotData (full,Grid,outputfile)*.

Full is an object that contains the following fields.

- full.x: denotes which part of *outputfile* needs to be plotted in the x dimention.
  - *full.x='all'* means that the whole x dimention in *outputfile* plotted
  - *full.x=(1:10)* means that the first ten elements in the x dimention in *outputfile* are plotted.
  - *full.x=1* means that only the first element in the x dimention in *outputfile* is plotted.

  NOTE: The coordinates are in number of cells **in the output file** not in the actual simulation space. In other words, if one only outputs the plane x=64 in the simulation space, full.x must be one as the output is only one cell wide in the x direction.
- full.y: Same as full.x but in the y dimention.
- full.z: Same as full.z but in the z dimention.
- full.field: Denotes which field present in outputfile has to be plotted

  The following fields can be exported.
  - (F)_real: Real part of F where (F = Ex, Ey, Ez, Hx, Hy, Hz) - *ex full.field=Ex_real*
  - (F)_imag: Imaginary part of F where (F = Ex, Ey, Ez, Hx, Hy, Hz) - *ex full.field=Ex_imag*
  - (F)_phase: Phase of F where (F = Ex, Ey, Ez, Hx, Hy, Hz) - *ex full.field=Ex_phase*
  - (F)_abs: Absolute value of F where (F = Ex, Ey, Ez, Hx, Hy, Hz) - *ex full.field=Ex_abs*
  - (F)_complex: Complex F where (F = Ex, Ey, Ez, Hx, Hy, Hz) - *ex full.field=Ex_complex*
  - topology: Plots the properties of the structure i.e. to verify the structure has been properly defined.
  - ME2_abs: Absolute value of the total Electric field i.e $E_x^2+E_y^2+E_z^2$- *ex full.field=ME2_abs*
  - MH2_abs: Absolute value of the total Electric field i.e $H_x^2+H_y^2+H_z^2$- *ex full.field=MH2_abs*
  - P(D)_(T): Real part, imaginary part, phase,absolute value or complex(T=real, imag, phase, abs, complex) Poynting vector in the x, y or z(D=x, y or z) direction-*ex full.field=Py_complex*
- full.data: Denotes which part data in frequency or in time domain has to be plotted
  - *full.data='all'* plots all the timesteps (if *outputfile* contains timetraces) or all the frequency bins (if *outputfile* contains FFT's)
  - *full.data= 54* plots the 54th frequency bin or 54th outputted timestep.
    - The function *frequency2index* converts a frequency to an index.
- full.plottype: Donotes the type of plot
  - *full.plottype='Linear'* Linear plot.
  - *full.plottype='log'* Logaritmic plot.

- full.noplot: If one no plot is generated, the data and pos is availble.
- full.figure: Axis to use for the plot
  - *full.figure=gca;*

[Data,pos]=PlotData (full,Grid,outputfile) generates the plot using with the results with path *outputfile*. In addition the data is outputted available as Data and the axis of the plot are outputted as an object pos.

- pos.x contains the x axis of the generated plot.
- pos.y contains the y axis of the generated plot.

pos.z contains the z axis of the generated plot.

5 Exemple : Plasmonic waveguide with dipole antenna.

This example can be found as *./Matlab/Testcases/FullWaveAntenna_Dipole.m*. The goal of this exemple is to exite a dipole like metallic antenna exited with a **Gaussian Beam** and utilizing **metals, variable gridding** and **symmetry**. Some aspects are the same as in the previous example.

5.1 Create the variable grid

Creating a variable grid is a *two step process*. In the first step a basic constant coarse grid is created in the same way as in previous exemple. Then, using the coordinate system of the coarse grid, a *zone* in the coarse grid is defined which will be have a smaller mesh, creating a new grid, with variable cell size. The physical size of the total grid doesn't change but but now contains more cells as part of the grid will be more discretized as is depicted in figure 1 on the following page.

5.1.1 Creating the coarse grid

A coarse grid is created like in previous exemple with large cell size. See 4.1 on page 3 for more info.

5.1.2 Creating the variable grid within the coarse grid.

In order to create a finer grid the one object *fine* needs to be created with the following properties.

- fine.dxmin: Minimal cellsize in the X direction in the finely gridded zone.
- fine.dymin: Minimal cellsize in the Y direction in the finely gridded zone.
- fine.dzmin: Minimal cellsize in the Z direction in the finely gridded zone.
- fine.x: Start position of the finely gridded zone in the X direction in number of cells in the coarse grid.
- fine.y: Start position of the finely gridded zone in the Y direction in number of cells in the coarse grid.

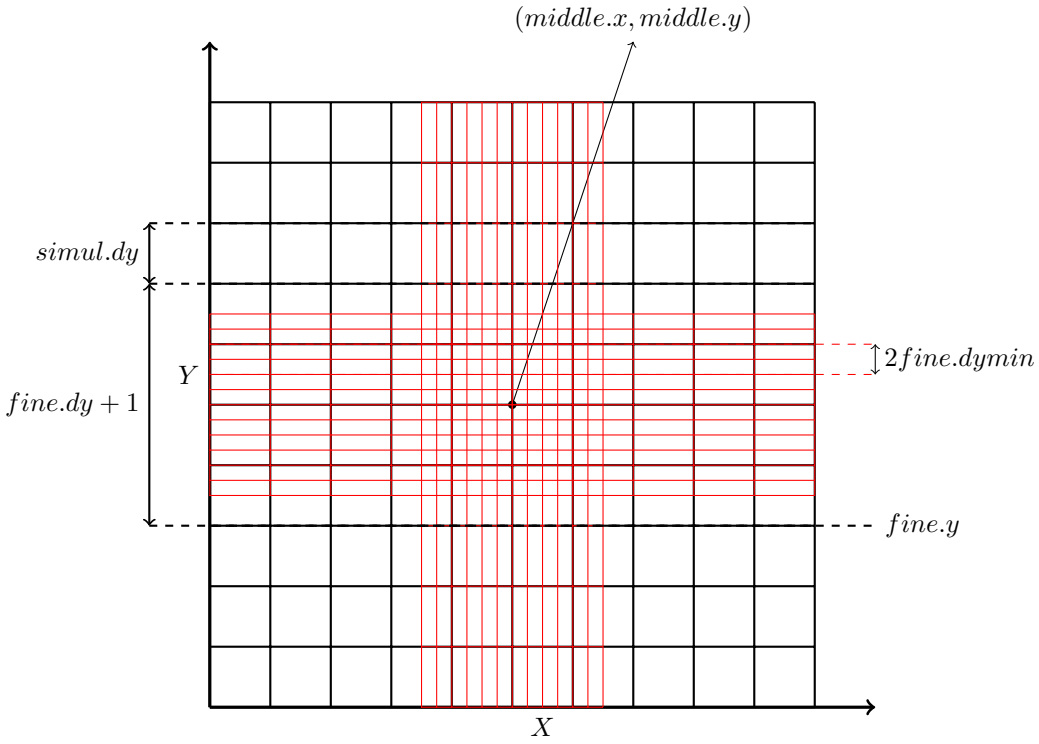

Figure 1: Coarse and variable gridding

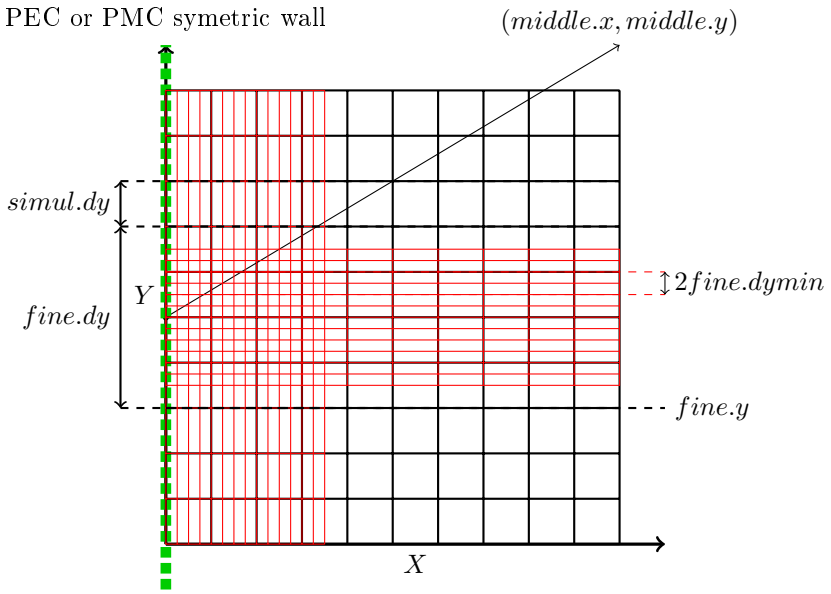

Figure 2: Coarse and variable gridding with $fine.uponly.x = 1$ and $fine.x = 1$

- fine.z: Start position of the finely gridded zone in the Z direction in number of cells in the coarse grid.
- fine.dx: Size of the finely gridded zone in the X direction in number of cells in the coarse grid.
- fine.dy: Size of the finely gridded zone in the Y direction in number of cells in the coarse grid.
- fine.dz: Size of the finely gridded zone in the Z direction in number of cells in the coarse grid.
- fine.pad: If one, the new grid will be padded in the X and the Y direction so that the total size of the simulation is divisible by 16 in the X direction and the Y direction.
- fine.padtype: Sets weather the simulation space needs to be padded symmetricaly or not.
  - *fine.padtype='symetric'* will add half the cells on the left and half the cells on the right in each direction.
- fine.uponly.x: When symmetric boundary conditions are used, the variable grid needs to start without a smooth transition from the coarse grid. When one the obtained grid is depicted in figure 2. It works similarly in the y and z direction.
- fine.downonly.x: When symmetric boundary conditions are used, the variable grid needs to start without a smooth transition from the coarse grid. When one the obtained grid is depicted in figure 3. It works similarly in the y and z direction.

A *fine* object is created for each part of the coarse grid whose mesh needs to be refined and are added to an object array FINE.

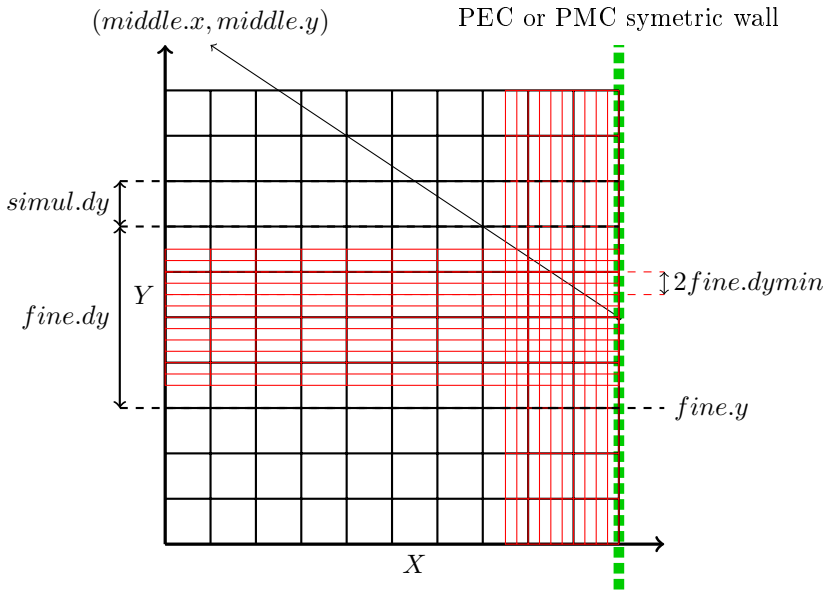

Figure 3: Coarse and variable gridding with $fine.downonly.x = 1$ and $fine.dx =$ 'end'

FINE{1}=fine;
FINE{2}=otherfinezone;
[Grid,middle]=encap(Grid,FINE) creates the finer grid. *Grid* is the new grid that containes the refined mesh. Middle is an object that contains the center of each zone in the new grid.

- middle(1).x : X position of the center of the finely gridded zone *fine* in the new grid.
- middle.(2).y: Y position of the center of the finely gridded zone *otherfinezone* in the new grid.

5.2 Adding dispersive materials

Dispersive materials like metals have a refractive index that varies with frequency. In BCALM those materials can be modelled using *multiple lorentz poles* whose parameters can be entered. For each material the w_{pm}, ω_m and Γ_m have to be fitted to match the $\epsilon(\omega)$ in the frequency band of interest.

$$\epsilon(\omega) = 1 + \sum_{m=1}^{\#poles} \frac{w_{pm}^2}{\omega_m^2 + j\omega\Gamma_m - \omega^2} \tag{6}$$

wpm=[4.38622782 4e15,1.008519648e15,7.251716026e15,1.332784755e16];
wm=[0,1.874386332e15,3.534954035e15,0];
gammam=[1.565488534e15 8.109675964e14 0 ,0];
Au.poles=[wpm',wm',gammam'];
Au.name='Au'; Au.epsilon=1;
LinearGrid=AddMat(LinearGrid,Au); shows how it is done for gold.

5.3 Adding symmetric boundary conditions

In order to reduce the simulation size, symmetric boundary conditions can be used when the structure is indeed symetric or anti-symetric. In BCALM one has to add Perfect Electric Conducting wall(symmetric normal electric fields) or Perfect Magnetic Conducting wall(symmetric normal magnetic fields).

- *Grid=SetAllBorders(Grid,'100000',1,'perfectlayerzone','PEC')* adds a PEC wall on the left side in the X direction. For symmentry in TE modes
- *Grid=SetAllBorders(Grid,'100000',1,'perfectlayerzone','PMC');* adds a PMC wall on the left side in the X direction. For symmentry in TM modes
- *Grid=SetAllBorders(Grid,'010000',1,'perfectlayerzone','PMC');* adds a PMC wall on the right side in the X direction. For symmentry in TM modes
- *Grid=SetAllBorders(Grid,'001000',1,'perfectlayerzone','PMC');* adds a PMC wall on the left size in the Y direction. For symmentry in TM modes

The binary number toggles the walls that have to be set as a PMC or a PEC from left to right:

- left side in the x direction
- right side in the x direction
- left side in the y direction
- right side in the y direction
- left side in the z direction
- right side in the z direction

NOTE: Don't forget to toggle off the CPML's on that particular border and use the uponly and downonly functions in the variable gridding.

5.3.1 Plotting the with symmetry

In order to visualize the complete structure when using symmetric boundary conditions it is possible to specify the symmetry of the output in the object sent to the *PlotData* function.

full.sym.x =1 will make sure that PlotData(full,Grid,[SimulInfoM.outfile 'In_Plane' tag 'FFT']) mirrors the x axis in the plot.

6 Used Yee cell and termination of the simulation space.

FDTD discretizes the continious space and time. Electric and magnetic fields are staggered in space and in time in a configuration called the *Yee cell.* The position of the fields within a Yee cell can be of importance when one wants to match simulation results to theory and to comprehend how the simulations space is terminated.

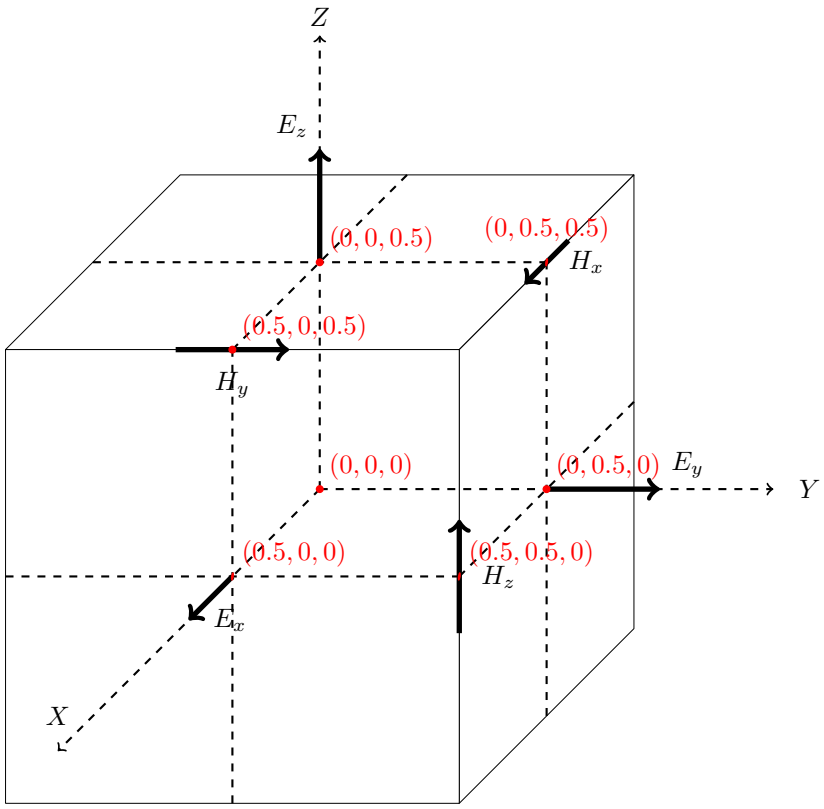

Figure 4: Yee Cell used

6.1 Used Yee cell

The way the fields are staggered in space is depicted in figure 4. In consequence, *sources* and *the outputted values* of different fields withing a single Yee cell are not exactly positioned at its center. This needs to be taken in mind in the analysis of results. Note that a different formalism of the Yee cell is possible(i.e. with the E and H substituted). The Yee cell as depicted in figure 4 is the only one used in BCALM. To remember this, one can think that as the X coordinate increases we cross H_x first and E_x last. This is valid for all directions.

6.2 Boundary conditions.

It is important to understand how BCALM terminates its simulations by default. Numereous errors originate in a misinterpretation on how this actually happens. *When not specifying any boundaries,* the simulation space is terminated by a *Perfect Magnetic Conducting* wall on the left bound in every direction and by a *Perfect Electric Conducting Wall* on the right bound in every direction. This is illustrated in the *X direction* by figure 5. The fields that are *in* the simulation space are denoted in black(i.e. these are the cells that can be sourced and are outputted). The cell 1 in figure 5 is therefore the the first cell in the *X direction* and cell *end* the last cell in the *X direction* in the actual Matlab interface. Imaginary cells that are *outside* the simulation space are denoted in by dashed blue lines.(i.e. these are cells that cannot be sourced and they are not part of the outputted fields). BCALM always assumes the fields that precede the first cell and

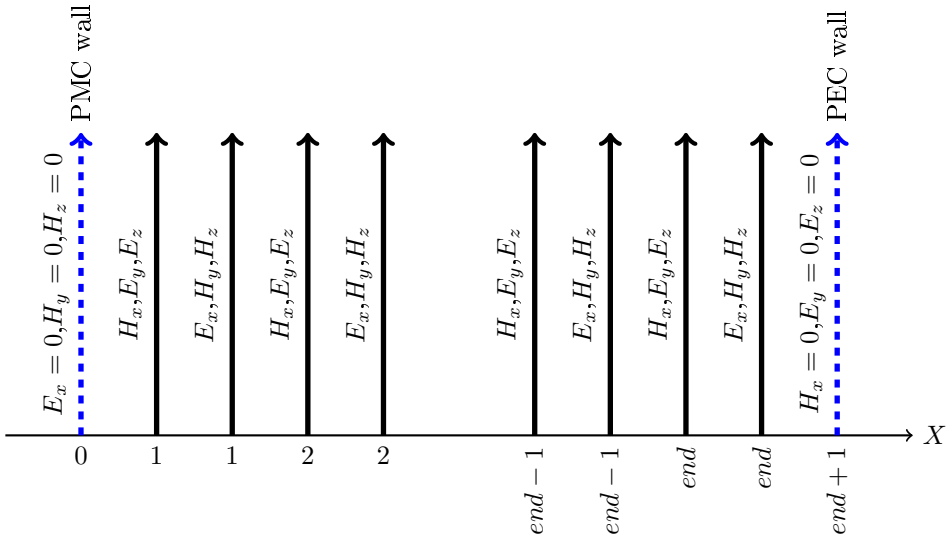

Figure 5: Default boundary conditions

follow the the last cell in every direction to be zero, creating a *PMC* wall on the left bound and *PEC* wall on the right bound in every direction.

6.2.1 User defined boundary conditions

User defined boundary conditions overrule the default ones but they potentially create fields that are unphysical at the border. There artifacts do not have a influence on the simulation results. This is illustrated in the X direction in figure 6 where a PMC wall is defined(in green) on both sides(analogue in other directions and for PEC). In this case, on the left bound, fields(in red) exist that are squeezed between the imaginary PMC and the user defined PMC. Despite the fact that these artifacts are blocked of by the user defined PMC wall, those fields will be unphysical. Consequently, it is better define PMC walls on the right bound and use the imaginary PMC on the left bound as depicted in figure 7.

7 Boundaries between materials

The spatial staggering of the electric and magnetic fields in FDTD carries inherent assymetries in the stucture. Averaging of the effective permittivity along borders as implemented in MEEP is **not implemented** in BCALM at the moment. This compromise was made to keep the simulation speed high. In this section different kinds of boundaries present in B-CALM are discussed.

7.1 Boundary between a dielectric material and a dielectric material

For dielectric to dielectric boundaries no special action is taken. The material properties remain defined on a per cell basis as is depicted in figure 8.

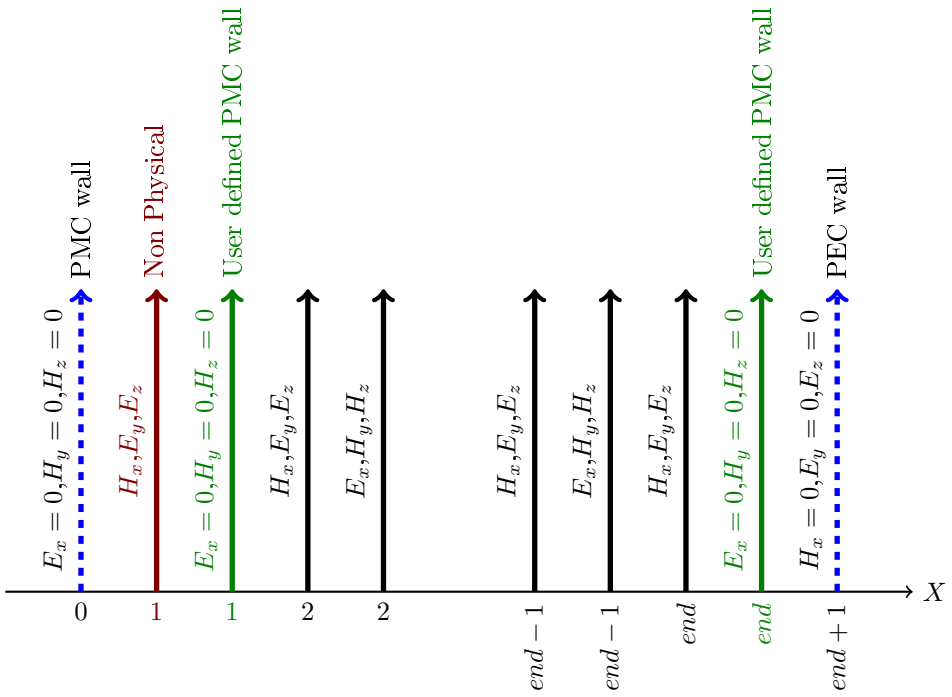

Figure 6: User defined boundary conditions

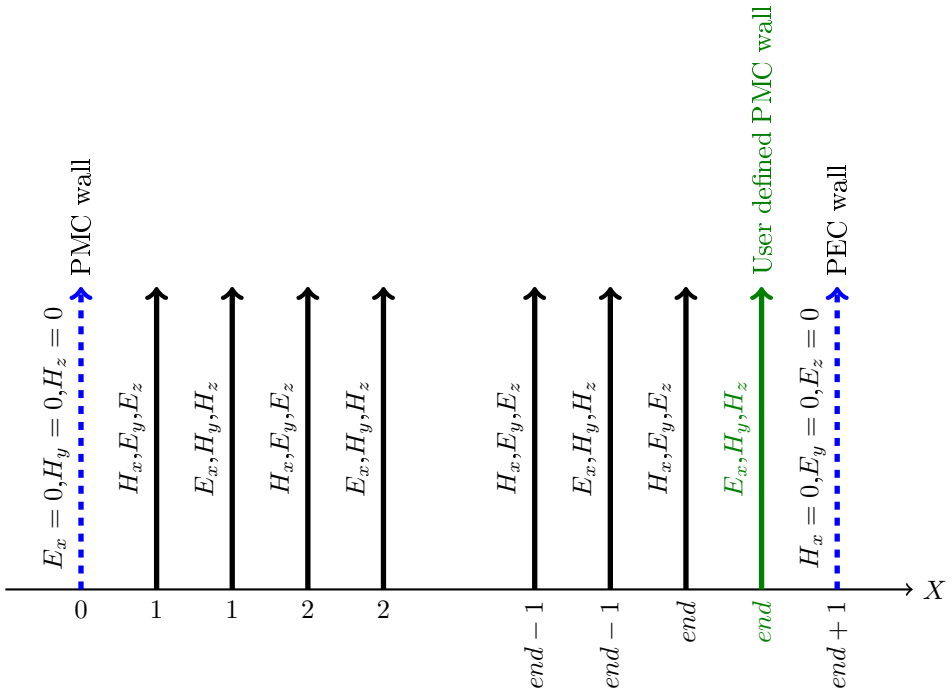

Figure 7: User defined boundary conditions

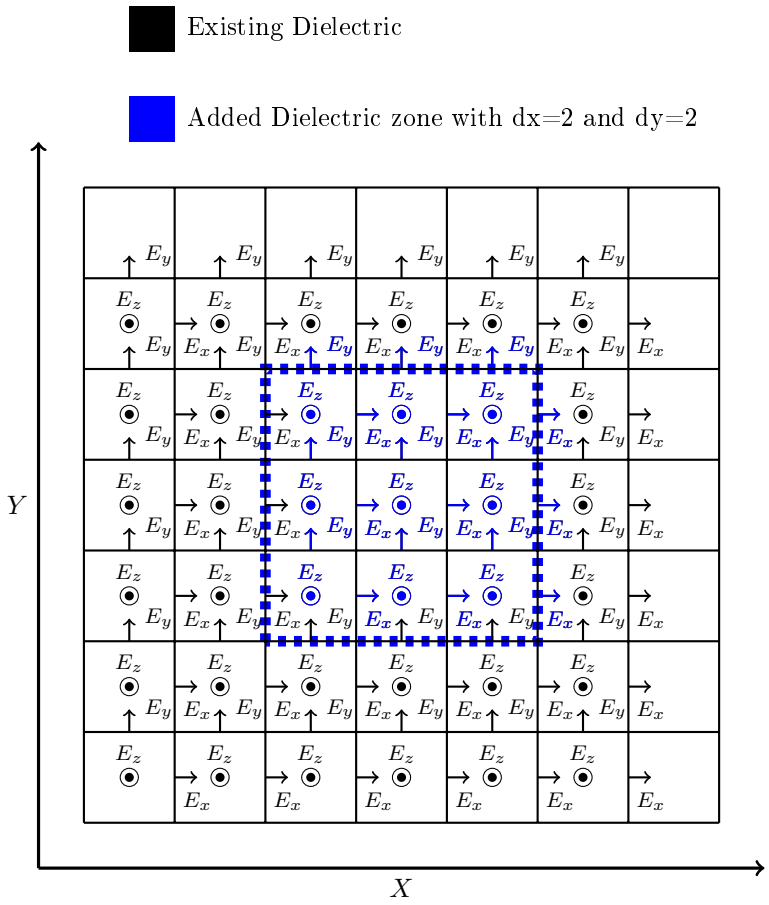

Figure 8: Dielectric to dielectric boundary

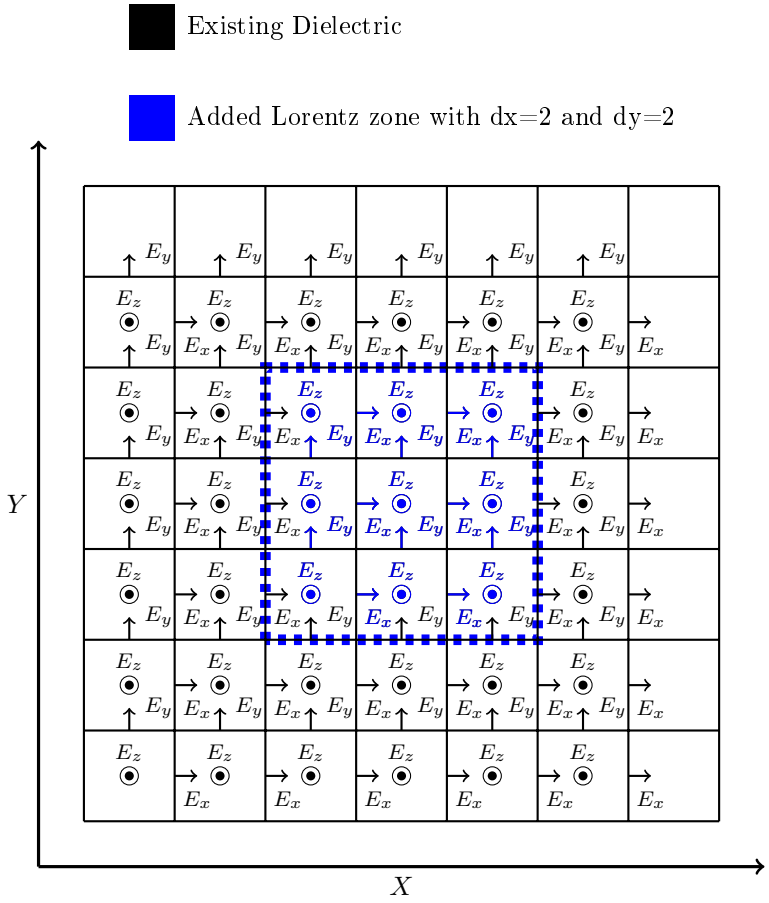

Figure 9: Lorentz to dielectric boundary

7.2 Boundary between a dielectric material and a lorentz material

When a lorentz material is added in a dielectric background the fields normal to the boundary on the last cell are treated as the dielectric that comes next. This means that the added lorentz material is in fact a half a cell smaller in each direction when added in a dielectric. This is illustrated in figure 9

7.3 Boundary between a lorentz material and a lorentz material

For lorentz to lorentz boundaries no special care is taken. The material properties remain defined on a per cell basis as is depicted in figure 10.

8 Generalities to get maximum performance in simulation.

In order to get maximal speedup one has to define the simulations in a way that maps well onto the core algorithm inside *fdtd*. When following the following rules that should, this is the case.

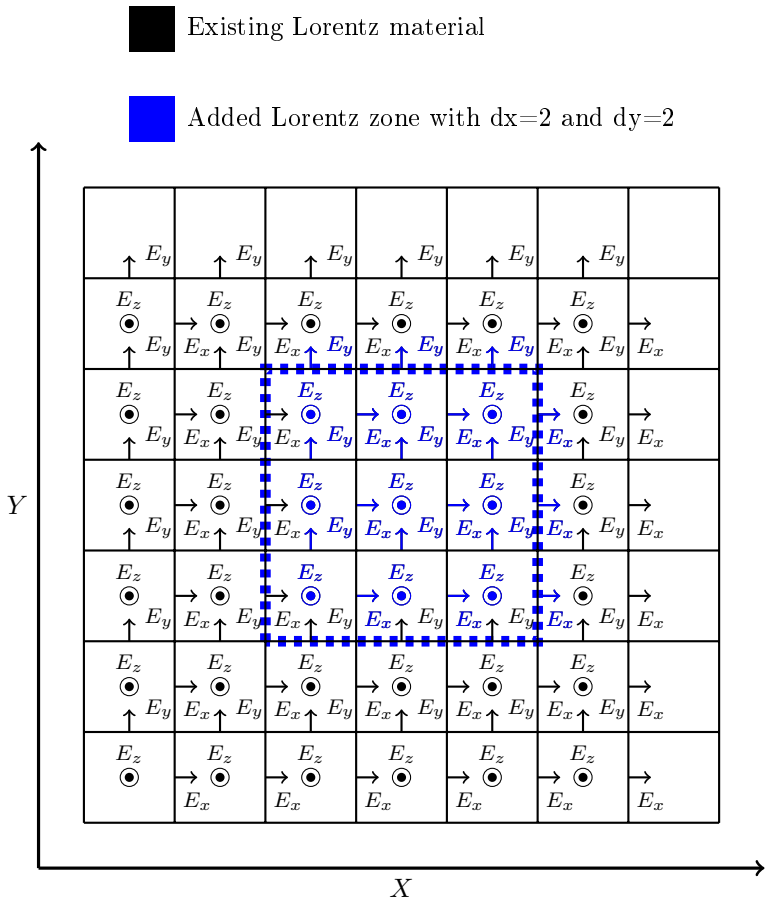

Figure 10: Lorentz to Lorentz boundary

Simulation	X	Y	Z
A	128 cells	64 cells	256 cells
B	128 cells	256 cells	64 cells

Table 1: Simulation orientation

Output	X	Y	Z
Output A	128 cells	128 cells	1 cell
Output B	128 cells	1 cell	128 cells
Output C	1cell	128 cells	128 cells

Table 2: Output zones speed

8.1 Importance of the simulation orientation

The simulation speed improves if the length of Z dimension of the simulation space is the smaller than the length of the X and Y dimension. In table 1 Simulation B would be substantially faster than simulation A.

8.2 Orientation of the output zones

The simulation speed improves if the user preferentially defines output zones containing all the cells in the X direction rather than all the cells in the Y or the Z dimension. In table 2 exporting Output A and Output B will be equally fast as one of their dimensions contains all 128 cells in the X direction. Output C, will slow down the simulation as 128 cells are exported in the Y and the Z direction.