Meep User Guide

Jeffrey Chen
Spring 2015

Introduction: Meep (or MEEP) which stands for MIT Electromagnetic Equation Propagation, is a free finite-difference time-domain (FDTD) simulation software package developed at MIT to model electromagnetic systems.

Background: Meep is written in C++, there is two main interfaces that are the most used (libctl/scheme). C++ provides the interfaces with the greatest level of flexibility in designing FDTD simulations. Although the foundations are built in C++ you do not need to be a real programmer to use Meep.

Ctl File: The use of Meep revolves around the control file, abbreviated "ctl" and typically called something like foo.ctl (although you can use any file name you wish). The ctl file specifies the geometry you wish to study, the current sources, the outputs computed, and everything else specific to your calculation.

Scheme: programming language developed at MIT, which has a particularly simple syntax: all statements are of the form (*function arguments*...).

Maxwell's Equations: Meep simulates Maxwell's equations, which describe the interactions of electric (**E**) and magnetic (**H**) fields with one another and with matter and sources.

$$\frac{d\mathbf{B}}{dt} = -\nabla \times \mathbf{E} - \mathbf{J}_B - \sigma_B \mathbf{B}$$

$$\mathbf{B} = \mu \mathbf{H}$$

$$\frac{d\mathbf{D}}{dt} = \nabla \times \mathbf{H} - \mathbf{J} - \sigma_D \mathbf{D}$$

$$\mathbf{D} = \varepsilon \mathbf{E}$$

$$\nabla \cdot \mathbf{B} = -\int^t \nabla \cdot (\mathbf{J}_B(t') + \sigma_B \mathbf{B}) dt'$$

$$\nabla \cdot \mathbf{D} = -\int^t \nabla \cdot (\mathbf{J}(t') + \sigma_D \mathbf{D}) dt' \equiv \rho$$

Units in Meep: Meep uses "dimensionless" units where all these constants are in unity**.** Almost everything you might want to compute (transmission spectra, frequencies, etcetera) is expressed as a ratio anyway, so the units end up canceling.

Boundary conditions and symmetries: we can only simulate a finite region of space, which means that we must terminate our simulation with some boundary conditions. Three basic types of terminations are supported in Meep: Bloch-periodic boundaries, metallic walls, and PML absorbing layers.

Start-up:

It is important to note that Meep is an Unix bases shell program or like one. Which means that all operations will be done through a Unix machine in the terminal, in my case I am using Ubuntu 14.10 but any Unix interface should work as long as you correctly build the package.

Recommended installations: Below are some highly recommended things that should be also installed with your Meep package. You do however must go through the installation page and manually build each of them since there is no built packages.

- **Libctl**: Front-end to Meep
- **HDF5**: Meep's output of visuals
- **BLAS and LAPACK:** Algebra packages
- **Imagemagick:** (optional) Format converter

Now that you have your package built, Meep is ready for use.

Open up your Terminal and type in "meep" and it should pull up your active shell.

Before we work out way through a tutorial, you should take a look at the very basic code and syntax that I have listed below so that you can a feel of what code will look like in the program.

Starting Meep:

```
meep
```

Resetting the Meep Shell:

```
(reset-meep)
```

Reading a file and executes it:

```
unix% meep YOUR_FILE_NAME.ctl
```

Converting image to PNG:

```
unix% h5topng -S3 Name_OF_YOUR_IMAGE.h5
```

-S3 is the scale of the image, so when you use (3) that is the increase that you are making

I think the best way to get started is to actually build something. So finally I will walk you through a simple simulation in Meep and hopefully by the end of this you will had the basic knowledge of how to use Meep.

90 Degree Bend Tutorial:

First you want to make sure that your shell is clear of any other code that could be running so it is always good to run the reset script before starting.

Lets begin by setting up your boundaries by using:

```
(set! geometry-lattice (make lattice (size 16 16 no-size)))
```

-This piece of code will create a 16x16 area for your waveguide to compute. Remember that Meep is unit-less so 16 will just become a ratio in respects to your waveguide.

Setting up your blocks and fitting:

```
(set! geometry (list
                (make block (center -2 -3.5) (size 12 1 infinity)
                     (material (make dielectric (epsilon 12))))
                (make block (center 3.5 2) (size 1 12 infinity)
                     (material (make dielectric (epsilon 12)))))
```

- The code here breaks up your waveguide into two blocks, as you can see the size of each block is 12 while it is centered at the given coordinates in respect to your 16x16 boundary. Finally the material is added at the end that will travel the same 12 units as your blocks.

Setting the thickness:

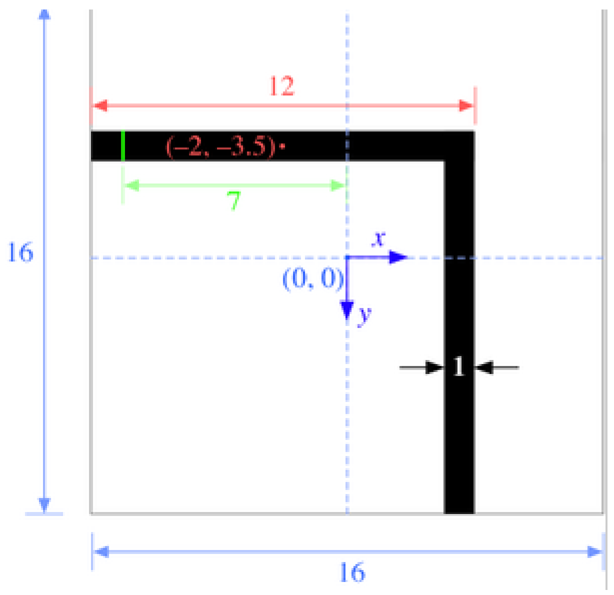
```
(set! pml-layers (list (make pml (thickness 1.0))))
(set! resolution 10)
```

-now your block thickness is set to 1 and the resolution of your system is created

Setting your wavelength:

```
(set! sources (list
               (make source
                 (src (make continuous-src
                        (wavelength (* 2 (sqrt 12))) (width 20)))
                 (component Ez)
                 (center -7 -3.5) (size 0 1))))
```

-This piece is what will create your waveguide, the idea help this code is that it will expand into a line source which will be the same with of the waveguide. This is done by adding a size property to the source. The (width 20) part is the time proportional to time. So you can think of it as "20 time units). Finally we make our XY shift at the end to offset flush with the blocks.



Run:

```
(run-until 200
        (at-beginning output-epsilon)
        (to-appended "ez" (at-every 0.6 output-efield-z)))
```

-*Now we can run the code with this script. It is important to note that 0.6 is the time of our simulation. You can think of it as,* at every 0.6 time units (about 10 times per period) via (at-every 0.6 output-efield-z).

Saving the Tutorial:

The saving aspect of MEEP is important in that you can convert your simulation into a .png or .gif from a dataset. Below is some code that will first save your data into a .h5 file and creates an array. These step compliment the 90 Degree bend tutorial, so if you were using these steps to save something else you may or may not need to make an adjustments.

.h5 File into Array:

```
unix% h5ls ez.h5
ez                  Dataset {161, 161, 330/Inf}
```

- *this uses the unix% command to build your file, which is named "ez" of course you can name it whatever you like. If you open up the file it will contain a data set in a 163x162x330 array. This is extra information but a formated array in this array can be converted into an image.*

Saving as an Animation:

```
unix% h5topng -t 0:329 -R -Zc dkbluered -a yarg -A eps-000000.00.h5 ez.h5
```

- *This line of code is very similar to the array one but instead Meep will save your file over a set of time. This is represented by 0:329, which stands for all indices from T = 0 to T = 329.*

Converting into .png and .gif:

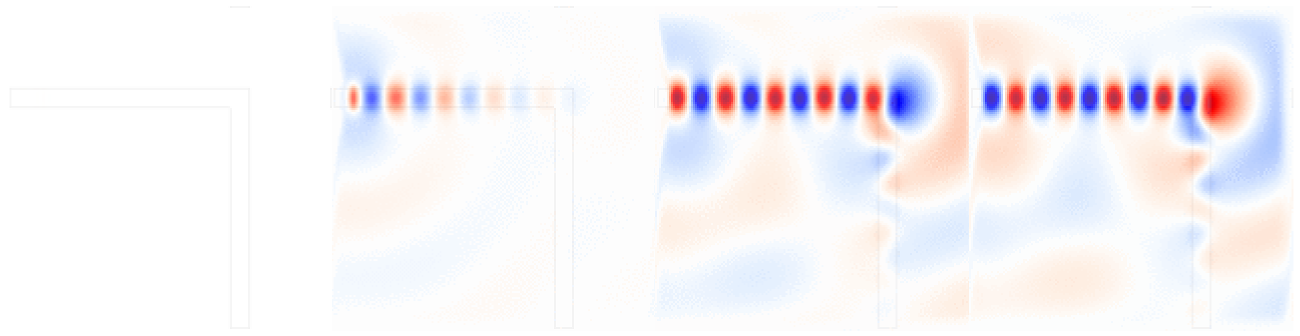
This in my opinion is the trickiest part because we need to first build another source code page that can convert our .h5 files into a different format. MIT recommends using ImageMagick. This is just an image editing tool that can do file conversion. The great thing about this program is that it can be shell scripted so once you have the package installed correctly, all you need to do is copy the code below.

```
unix% convert ez.t*.png ez.gif
```

Final thoughts on the Tutorial:

If all is done correctly then you should receive an image of the waveguide and also a GIF of the simulation. Of course if you are starting from a blank slate the hardest part I would say is getting a clean install of everything that you need to make MEEP work. If all is done correctly you should get the image/gif below.

90 Degree GIF:



PNG image: