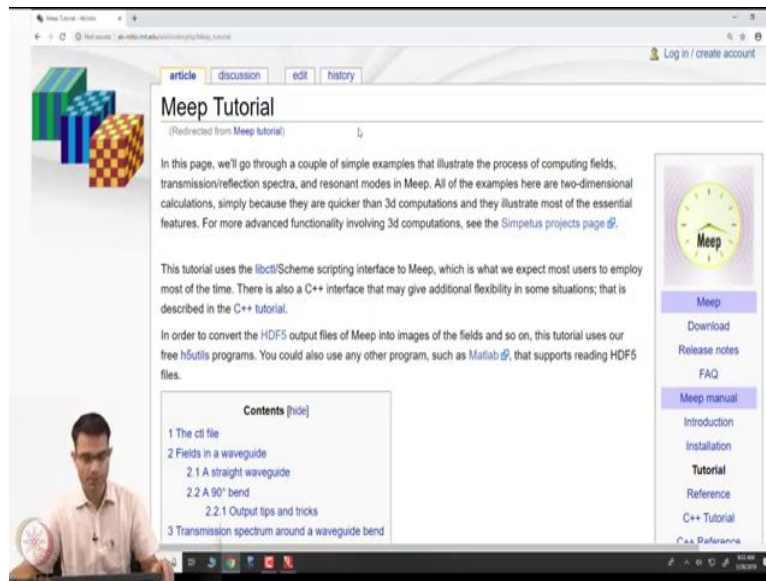


Computational Electromagnetics
Prof. Uday Khankhoje
Department of Electrical Engineering
Indian Institute of Technology, Madras

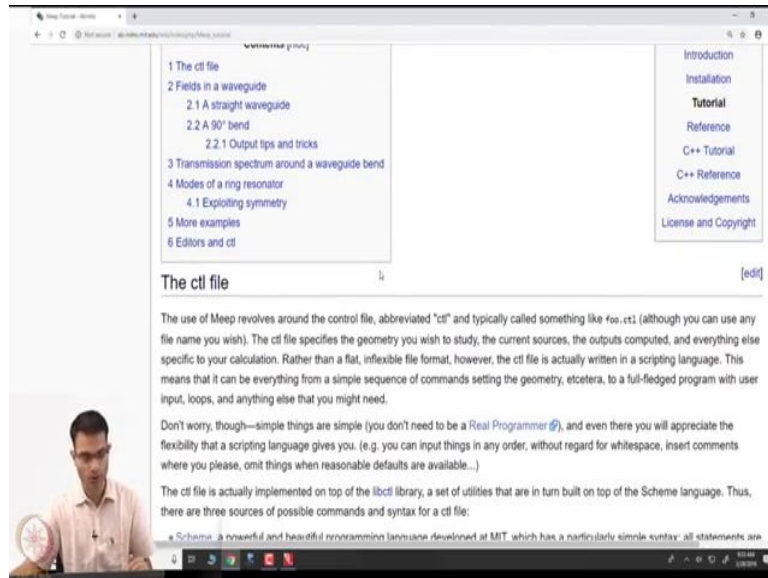
FDTD: Materials and Boundary Conditions
Lecture - 13.20
MEEP : FDTD in action

(Refer Slide Time: 00:14)



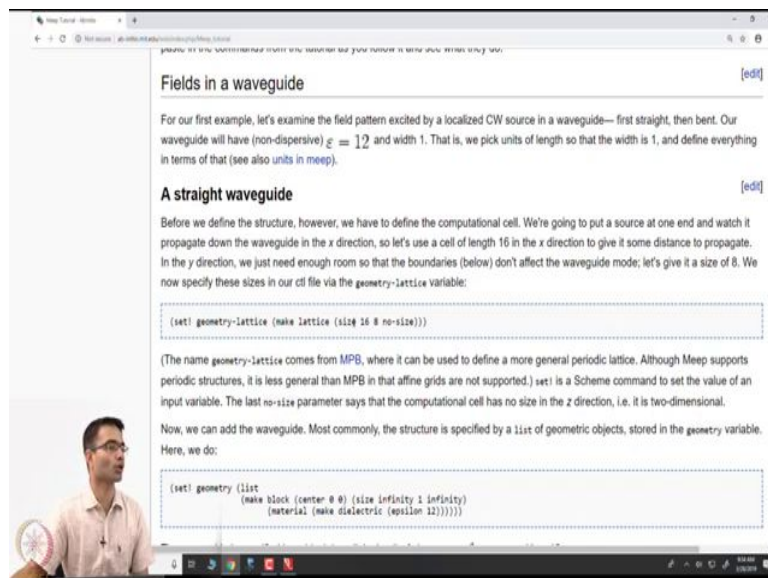
So it is called Meep the text is readable yeah ok.

(Refer Slide Time: 00:24)



So, I would not read through all of this I will just working through this we will also get an idea of how to what are the steps in running an FDTD program ok.

(Refer Slide Time: 00:31)

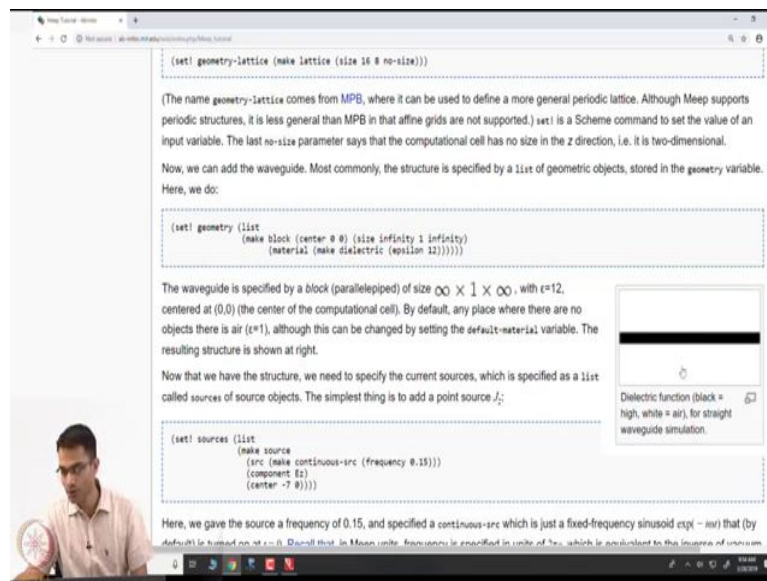


So, let us go to. So, the first example they have is for example, fields in a waveguide. So, imagine there is waveguide I want to send wave through very simple we know analytically in

these cases we can even analytically calculate what the fields look like, but that is not our objective let us see what this simulation does.

So, the first line over here is. So, this is as I mentioned is written in a language called scheme which is a slightly unusual language. So, we would not get into syntax, but what is being set over here this size is 16 8 no size this is telling you the size of the computational domain. So, 16 units in x, 8 units in y and no size in z means its 2D simulation that is all that it means.

(Refer Slide Time: 01:19)



```
(set! geometry-lattice (make lattice (size 16 8 no-size)))
```

(The name `geometry-lattice` comes from MPB, where it can be used to define a more general periodic lattice. Although Meep supports periodic structures, it is less general than MPB in that affine grids are not supported.) `set!` is a Scheme command to set the value of an input variable. The last `no-size` parameter says that the computational cell has no size in the z direction, i.e. it is two-dimensional.

Now, we can add the waveguide. Most commonly, the structure is specified by a list of geometric objects, stored in the `geometry` variable. Here, we do:

```
(set! geometry (list  
  (make block (center 0 0) (size infinity 1 infinity)  
    (material (make dielectric (epsilon 12))))))
```

The waveguide is specified by a block (parallelepiped) of size $\infty \times 1 \times \infty$, with $\epsilon=12$, centered at (0,0) (the center of the computational cell). By default, any place where there are no objects there is air ($\epsilon=1$), although this can be changed by setting the `default-material` variable. The resulting structure is shown at right.

Now that we have the structure, we need to specify the current sources, which is specified as a list called `sources` of source objects. The simplest thing is to add a point source J_z :

```
(set! sources (list  
  (make source  
    (src (make continuous-src (frequency 0.15)))  
    (component Ez)  
    (center -7 0))))
```

Here, we gave the source a frequency of 0.15, and specified a `continuous-src` which is just a fixed-frequency sinusoid $\exp(-i\omega t)$ that (by default) is turned on at $t=0$. (Recall that in Meep units, frequency is specified in units of 2π , which is equivalent to the inverse of one unit of time.)

Dielectric function (black = high, white = air), for straight waveguide simulation.

Next I have to make the object. What is the object? It is a waveguide. So, a waveguide as shown in this figure over here is this black strip over here. So, it is like a slab waveguide right. So, I have x is along the waveguide and then y is the other axis right. So, here the waveguide is being made. So, it is being made as a block of some ϵ . So, $\epsilon = 12$ has been given ok.

The width is given as 1 unit and infinite in the other dimension so, fairly intuitive right. So, the width of this black strip is 1 it has epsilon equal to 12 and it extends forever of course, it extends forever because I have defined the computational domain about we have 16 units over here right. So, in practice it is going to be 16 units right.

(Refer Slide Time: 15:16)

The waveguide is specified by a block (parallelepiped) of size $\infty \times 1 \times \infty$, with $\epsilon=12$, centered at (0,0) (the center of the computational cell). By default, any place where there are no objects there is air ($\epsilon=1$), although this can be changed by setting the `default-material` variable. The resulting structure is shown at right.

Now that we have the structure, we need to specify the current sources, which is specified as a list called `sources` of source objects. The simplest thing is to add a point source J_z :

```
(set! sources (list  
  (make source  
    (src (make continuous-src (frequency 0.15)))  
    (component Ez)  
    (center -7 0)))
```

Here, we gave the source a frequency of 0.15, and specified a `continuous-src` which is just a fixed-frequency sinusoid $\exp(-i\omega t)$ that (by default) is turned on at $t=0$. Recall that, in Meep units, frequency is specified in units of 2π , which is equivalent to the inverse of vacuum wavelength. Thus, 0.15 corresponds to a vacuum wavelength of about $1/0.15 = 6.67$, or a wavelength of about 2 in the $\epsilon = 12$ material—thus, our waveguide is half a wavelength wide, which should hopefully make it single-mode. (In fact, the cutoff for single-mode behavior in this waveguide is analytically solvable, and corresponds to a frequency of $1/2\sqrt{11}$ or roughly 0.15076.) Note also that to specify a J_z , we specify a component E_z (e.g. if we wanted a magnetic current, we would specify H_x , H_y , or H_z). The current is located at $(-7, 0)$, which is 1 unit to the right of the left edge of the cell—we always want to leave a little space between sources and the cell boundaries, to keep the boundary conditions from interfering with them.

Speaking of boundary conditions, we want to add absorbing boundaries around our cell. Absorbing boundaries in Meep are handled by perfectly matched layers (PML)—which aren't really a boundary condition at all, but rather a fictitious absorbing material added around the edges of the cell. To add an absorbing layer of thickness 1 around all sides of the cell, we do:

```
(set! pml-layers (list (make pml (radiusvec 1 0))))
```

Dielectric function (black = high, white = air), for straight waveguide simulation.

Now, by default everything else is vacuum that is what you would expect ok. Next comes a source I need to put a source to excite some field in the waveguide how will I get the wave in there. So, what have they done? They have made a $e^{j\omega t}$ kind of a field ok.

They have given some frequency in some normalised units 0.15 that is the frequency and they are saying that is E_z polarisation over here and they have given some centre for it ok. So, centre remember our size was 16 so, -8 to 8. So, at -7 that is at the almost at the left most part I am putting E_z polarised source and what kind of a source? Continuous wave it is not that source that starts in time and that decays in time continuous wave the dielectric is the waveguide this black strip over here is the waveguide why would not the wave go out ok? Good question. Why would not the wave go out? So, I am exciting it within this and I mean. So, this is something which is you should have studied in your undergraduate course.

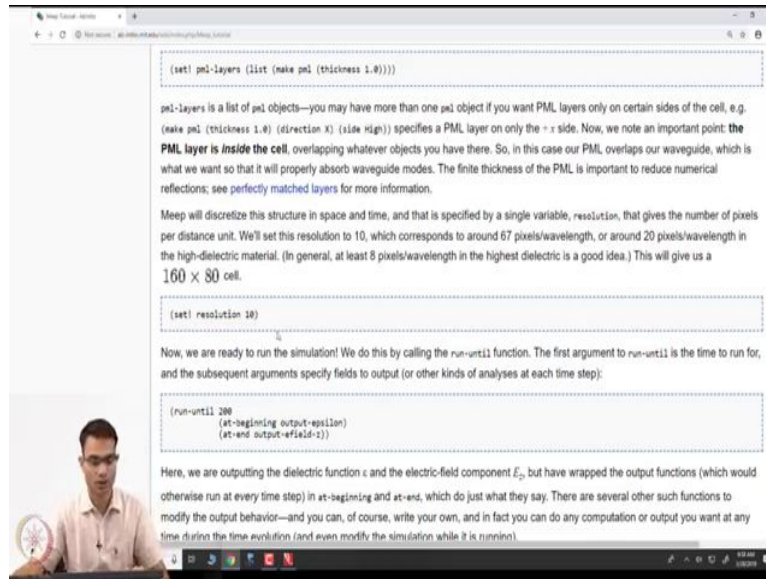
Student: (Refer Time: 03:26).

Ha no. So, the field does leak out it is not the case that the field is strictly confined within the dielectric the field does leak out exact we will see it is not confined purely inside some of it does leak out. We do not have to think worry about how much is undergoing total internal reflection the Maxwell's equation will take care of it.

Student: Only that.

Yeah some will be some will guide it some will get will leak out ok. So, we have we have specified the source what else do we need to specify boundary condition right, if I do not specify boundary condition my simulation will be wrong.

(Refer Slide Time: 04:10)

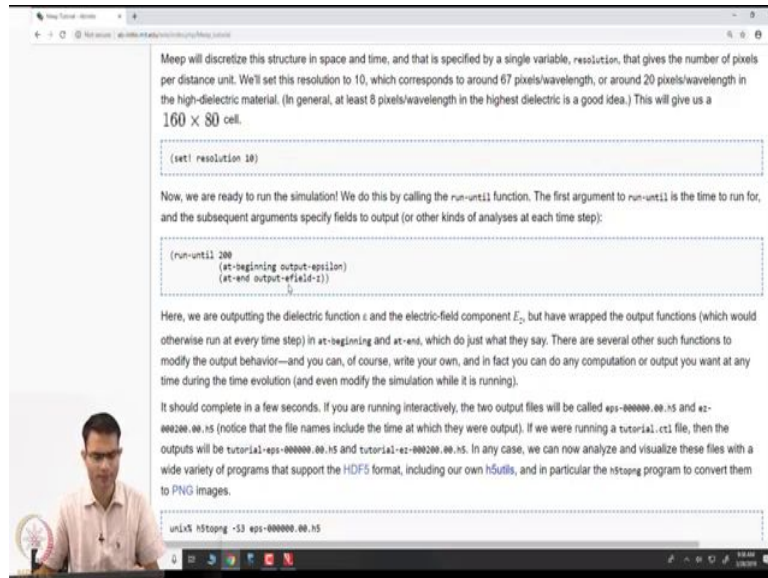


So, they have specified here PML this is just one line specification set a PML of thickness 1. So, by default what it will do you look at this computational domain over here it will put like a picture frame of thickness 1 all around ok.

That is why the source was put at the -7 not at -8 because you do not want source in the material you wanted just outside ok. So, the PML is from -8 to -7 and the source is at -7 ok. Now remember your Δx and Δt is yet to be specified right. So, here we set for example, this set resolution so, that setting Δx in some normalized way.

The Courant parameter is a default parameter that is already fixed in the simulation you can change it if you want, but they give you a default which will be a stable number. So, fixing Δx is good enough Δt will get fixed according to the Courant ok.

(Refer Slide Time: 05:15)



Meep will discretize this structure in space and time, and that is specified by a single variable, `resolution`, that gives the number of pixels per distance unit. We'll set this resolution to 10, which corresponds to around 67 pixels/wavelength, or around 20 pixels/wavelength in the high-dielectric material. (In general, at least 8 pixels/wavelength in the highest dielectric is a good idea.) This will give us a 160×80 cell.

```
(set! resolution 10)
```

Now, we are ready to run the simulation! We do this by calling the `run-until` function. The first argument to `run-until` is the time to run for, and the subsequent arguments specify fields to output (or other kinds of analyses at each time step):

```
(run-until 200  
  (at-beginning output-epsilon)  
  (at-end output-efield-z))
```

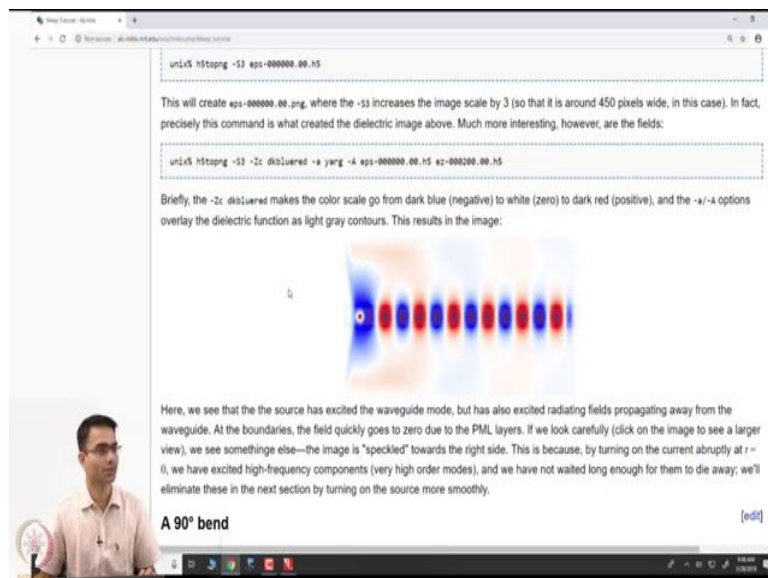
Here, we are outputting the dielectric function ϵ and the electric-field component E_z , but have wrapped the output functions (which would otherwise run at every time step) in `at-beginning` and `at-end`, which do just what they say. There are several other such functions to modify the output behavior—and you can, of course, write your own, and in fact you can do any computation or output you want at any time during the time evolution (and even modify the simulation while it is running).

It should complete in a few seconds. If you are running interactively, the two output files will be called `eps-000000.00.hs` and `ez-000200.00.hs` (notice that the file names include the time at which they were output). If we were running a `tutorial.cti` file, then the outputs will be `tutorial-eps-000000.00.hs` and `tutorial-ez-000200.00.hs`. In any case, we can now analyze and visualize these files with a wide variety of programs that support the HDF5 format, including our own `h5utils`, and in particular the `h5topng` program to convert them to PNG images.

```
unix$ h5topng -s3 eps-000000.00.hs
```

Then finally, you have to tell the simulation that how long do I wanted to run? So, you say 200 time units some more commands are given to output the fields and all of those things ok.

(Refer Slide Time: 05:27)

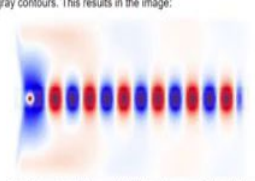


```
unix$ h5topng -s3 eps-000000.00.hs
```

This will create `eps-000000.00.png`, where the `-s3` increases the image scale by 3 (so that it is around 450 pixels wide, in this case). In fact, precisely this command is what created the dielectric image above. Much more interesting, however, are the fields:

```
unix$ h5topng -s3 -z:disk:ared -a yang -4 eps-000000.00.hs ez-000200.00.hs
```

Briefly, the `-z:disk:ared` makes the color scale go from dark blue (negative) to white (zero) to dark red (positive), and the `-a/-A` options overlay the dielectric function as light gray contours. This results in the image:



Here, we see that the the source has excited the waveguide mode, but has also excited radiating fields propagating away from the waveguide. At the boundaries, the field quickly goes to zero due to the PML layers. If we look carefully (click on the image to see a larger view), we see something else—the image is “speckled” towards the right side. This is because, by turning on the current abruptly at $t = 0$, we have excited high-frequency components (very high order modes), and we have not waited long enough for them to die away; we’ll eliminate these in the next section by turning on the source more smoothly.

A 90° bend [edit]

This is what the output of the simulation is at the end of 200 seconds this is what the fields look like right. So, the source was over here and looking at this can you say that there is definitely a wave guiding happening clearly right the field is very nicely confined to the waveguide, but not strictly some of it is leaking out no problem right it is not and this is not a

lossy material, it is just $\epsilon = 12$ there is no imaginary part. So, this field will go on for a very long time and distance right. But you can see its leaking out, next to the left hand side interface you can see there is something strange happening over here that is because there is a.

Student: PML.

PML over here so, its absorbing it almost perfectly, but I mean in practice some of it will not get absorbed right. So, they make a point here that because, they turned on the field abruptly right. So, the simulation started at $t=0$ and the next instance there was a $e^{j\omega t}$. So, we know that that as far as physics is concerned there is an abrupt turns on the field.

So, there will be some high frequency component right. So, that is what we say towards the right hand side over here the field looks a little what they called speckled you can see the field over here it has some the shading is slightly different from at the very beginning here is the very nice homogenous dark red here is a here there are little white dots inside the red. So, that they says coming from the high frequency components that came as a result of turning the field on abruptly.

Student: (Refer Time: 07:08).

A good question what is the red and blue? So, red will correspond to maximum positive blue will correspond to maximum negative right. So, it is like it is a wave that is travelling right. So, alternate maxima and minima in a wave that is what is happening.

Student: That thing in the.

That thing in the.

Student: (Refer Time: 07:27). Yeah, white it is 0; yeah white it is 0. So, that is what they called this dark blue red that is the you specify the legend for the colour. So, that is how they do. So, typically when you see a simulation of red and white then you should have assumed that one is maximum positive, one is maximum negative and white is 0 this dot that is the source.

Student: Dark.

The dark thing that actually that that is like the darkest red and this is the darkest blue that is how? I mean that is the shades of blue and red shades of not blue and red shades of dark blue and dark red. So, dark blue red is one legend blue red is another legend we can choose ok. So, people use different things for enhancing the contrast.

Student: Ok.

So, this is the example which you could have done analytically also right, but the power of the FDTD is that you can do things which are analytically not possible.

(Refer Slide Time: 08:25)

waveguide. At the boundaries, the field quickly goes to zero due to the PML layers. If we look carefully (click on the image to see a larger view), we see something else—the image is “speckled” towards the right side. This is because, by turning on the current abruptly at $t = 0$, we have excited high-frequency components (very high order modes), and we have not waited long enough for them to die away; we’ll eliminate these in the next section by turning on the source more smoothly.

A 90° bend

Now, we’ll start a new simulation where we look at the fields in a bent waveguide, and we’ll do a couple of other things differently as well. If you are running Meep interactively, you will want to get rid of the old structure and fields so that Meep will re-initialize them:

```
(reset-meep)
```

Then let’s set up the bent waveguide, in a slightly bigger computational cell, via:

```
(set! geometry-lattice (make lattice (size 16 16 no-size)))  
(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))))))  
(set! pml-layers (list (make pml (thickness 1.0))))  
(set! resolution 10)
```

Note that we now have two blocks, both off-center to produce the bent waveguide structure pictured at right. As illustrated in the figure, the origin (0,0) the coordinate system is at the center of the computational cell, with positive being downwards in istep, and thus the block of size 12x1 is centered at (-2, -3.5). Also shown in red in the figure is the entire plane at $x = -7$ (see below).

So, for example, they consider a 90 degree bend in the waveguide which you would not be solve analytically right. Now, what do you expect physically to happen I take a waveguide its guiding a wave I bend it by 90 degrees what would you expect will happen physically?

Student: That will leak it out.

Some of it will leak not it will its not like a pipe of water that I bend it and all the water goes out some of it will leak out that is what we will expect. So let us see what if our intuition is

confirmed over here. Now same thing they say make the computational domain as before 16, 16 and you have to make two blocks; one like this and the one the 90 degree bend.

(Refer Slide Time: 09:03)

```

(set! geometry-lattice (make lattice (size 16 16 no-size)))
(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 12 infinity)
    (material (make dielectric (epsilon 12)))))
(set! pml-layers (list (make pml (thickness 1.0))))
(set! resolution 10)
  
```

Note that we now have two blocks, both off-center to produce the bent waveguide structure pictured at right. As illustrated in the figure, the origin (0,0) of the coordinate system is at the center of the computational cell, with positive y being downwards in `xyplot`, and thus the block of size 12x1 is centered at (-2, -3.5). Also shown in green is the source plane at `x = -7` (see below).

We also need to shift our source to `y = -3.5` so that it is still inside the waveguide. While we're at it, we'll make a couple of other changes. First, a point source does not couple very efficiently to the waveguide mode, so we'll expand this into a line source the same width as the waveguide by adding a `size` property to the source (Meep also has an eigenmode source feature which can be used here and is covered in a separate tutorial). Second, instead of turning the source on suddenly at `t = 0` (which excites many other frequencies because of the discontinuity), we'll ramp it on slowly (technically, Meep uses a `turn-on` function) over a time proportional to the `size` of 20 time units (a little over three periods). Finally, just for variety, we'll specify the (vacuum) `wavelength` instead of the `frequency`; again, we'll use a wavelength such that the waveguide is half a wavelength wide.

So, they have shown it over here. So, one this way and one this way and they have done it by specifying two blocks same material, but this coordinates are different PML resolution everything as before ok.

(Refer Slide Time: 09:17)

```

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

Finally, we'll run the simulation. Instead of running `output-epsilon` only at the end of the simulation, however, we'll run it at every 0.6 time units (about 10 times per period) via `(at-every 0.6 output-epsilon)`. By itself, this would output a separate file for every different output time, but instead we'll use another feature of Meep to output to a single three-dimensional HDF5 file, where the third dimension is time:

```

(run-until 200
  (at-beginning output-epsilon)
  (to-appended "e2" (at-every 0.6 output-epsilon)))
  
```

Here, "e2" determines the name of the output file, which will be called `e2.h5` if you are running interactively or will be prefixed with the name of the file name for a cfl file (e.g. `tutorial-e2.h5` for `tutorial.cfl`). If we run `h5ls` on this file (a standard utility, included with HDF5, that lists the contents of the HDF5 file), we get:

```

unix$ h5ls e2.h5
e2      Dataset {161, 161, 330/Inf}
  
```

That is, the file contains a single dataset `e2` that is a 162x162x330 array, where the last dimension is time. (This is rather a large file, 69MB; later, we'll see ways to reduce this size if we only want images.) Now, we have a number of choices of how to output the fields. To output a single time slice, we can use the same `xyplot` command as before, but with an additional `-t` option to specify the time index: e.g. `h5plot -t 129` will output the last time slice, similar to before. Instead, let's create an animation of the fields as a function of time.

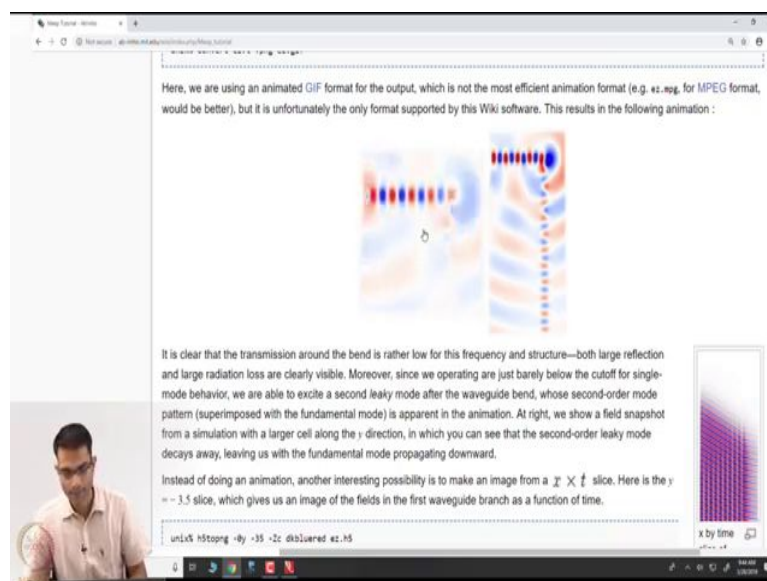
Now, to avoid that speckle phenomena what are they doing instead of turning the source on abruptly they are making it no they are still using a what are they will continuous source wavelength width so much yeah. So, what they are doing now is they are this instead of turning a source on suddenly at $t=0$, we ramp it slowly over a time proportional to the width of 20 time units. So, this width is equal to 20 time units is giving you the turn on time of the source and the wave length is given to you over here.

Yeah. So, technically Meep is implementing this turn on function using tan hyperbolic function and as I mentioned there is a open source we can go and look at the code and see all that information.

Student: Ok.

And as before run it for 200 time units. So, let us see what the output looks like. So, there is how the output looks like.

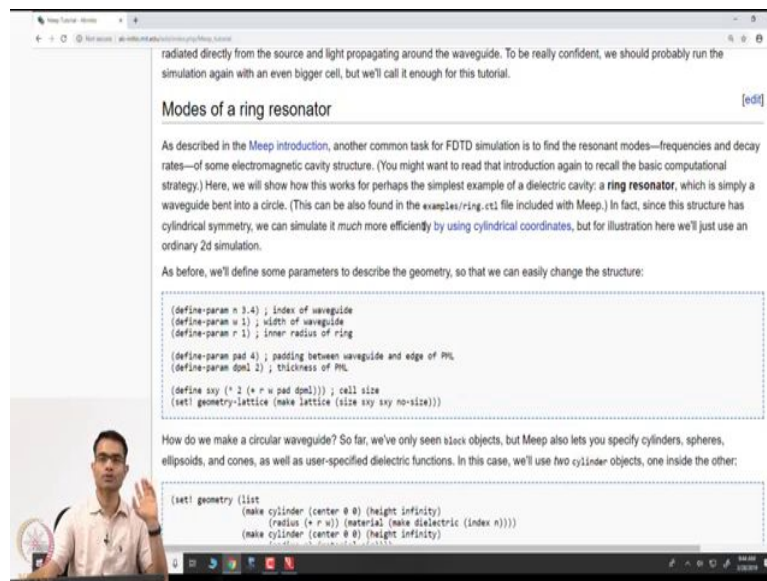
(Refer Slide Time: 10:26)



So, yeah this we can see this is the animation at every time instant you can record the field. So, you can see the field is going now it hits the bend and it begins to come into the lower part of the bend, but most of it is getting leaked out at that interface ok

So, that is another very cool aspect of FDTD. If I store the E_z at every time instant I can put it all together and make gif file. So, what they are showing over here is a gif file we will see the animation. So, then you can optimize various parameters if you want to minimize the leakage and all of that we would not go into that.

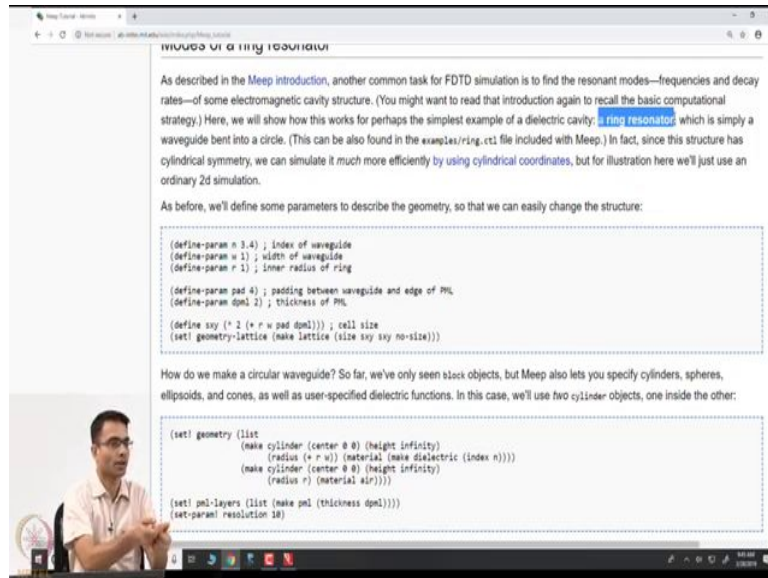
(Refer Slide Time: 11:13)



Then the final example I want to show you is modes of ring resonator ok. So, you all have heard the word resonator it means that its a structure which is selective in frequency it allows some frequency and it disregards the other frequencies. Now, supposing you want to design a resonator to work at some frequency how will FDTD will help us in that.

Student: Ok.

(Refer Slide Time: 11:43)



As described in the Meep introduction, another common task for FDTD simulation is to find the resonant modes—frequencies and decay rates—of some electromagnetic cavity structure. (You might want to read that introduction again to recall the basic computational strategy.) Here, we will show how this works for perhaps the simplest example of a dielectric cavity: [a ring resonator](#), which is simply a waveguide bent into a circle. (This can be also found in the `examples/ring.cti` file included with Meep.) In fact, since this structure has cylindrical symmetry, we can simulate it much more efficiently by using cylindrical coordinates, but for illustration here we'll just use an ordinary 2d simulation.

As before, we'll define some parameters to describe the geometry, so that we can easily change the structure:

```
(define-param n 3.4) ; index of waveguide
(define-param w 1) ; width of waveguide
(define-param r 1) ; inner radius of ring

(define-param pad 4) ; padding between waveguide and edge of PML
(define-param dpm1 2) ; thickness of PML

(define sxy (* 2 (+ r w pad dpm1))) ; cell size
(set! geometry-lattice (make lattice (size sxy sxy no-size)))
```

How do we make a circular waveguide? So far, we've only seen block objects, but Meep also lets you specify cylinders, spheres, ellipsoids, and cones, as well as user-specified dielectric functions. In this case, we'll use two `cylinder` objects, one inside the other:

```
(set! geometry (list
  (make cylinder (center 0 0) (height infinity)
    (radius (+ r w)) (material (make dielectric (index n))))
  (make cylinder (center 0 0) (height infinity)
    (radius r) (material air))))

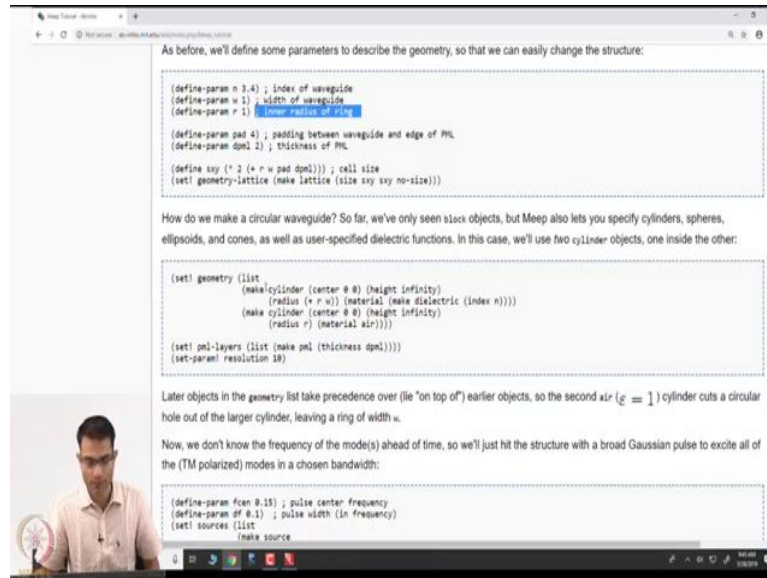
(set! pml-layers (list (make pml (thickness dpm1))))
(set-param! resolution 30)
```

So, let us look at the geometry over here. So, what have they done this they made a ring resonator ok. So, what is the ring resonator? Take the waveguide from before join it into a circle ok. So, we know that you know integer multiples of the distance around this are what will be the supported modes ok. So, here we want to confirm that intuition over here now hm. So, they have a you know the width of the waveguide radius all of these have been specified.

Yeah some, but $2\pi r = n\lambda$, but λ in the medium.

Student: Ok.

(Refer Slide Time: 12:18)



So, they making a cylinder over here all of this is what you would do to make the object ok. Now the question lies what should I if I excite this structure with the wrong frequency what will happen?

Student: Nothing will happen.

Nothing will happen right the fields will not decay I mean the field will not decay I mean the field will decay off it will not be a resonate phenomena, but once I designed a structure with some epsilon if the and if the geometry slightly complicated I would know what the resonate frequency is. So, how would I find that resonate frequency?

I can in FDTD I can specify a source after that what supposing my task someone gives me structure and says find out the resonate frequency of the structure.

Student: Excite.

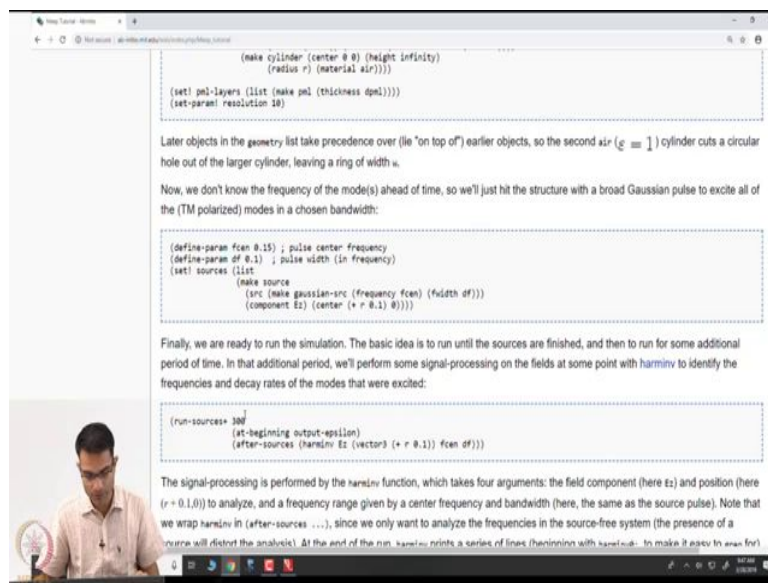
Excited with all frequencies right; so, that is well all frequencies is technically impossible white band right. So, the step 1 is excite it with the white band ok. So, here they say they are making a pulse kind of a frequency. So, they are saying a centre of the pulse is 0.15 in normalised units and a df right and like a like we have discussed earlier they use a Gaussian

source. So, they give a Gaussian source where they specify the centre frequency and the width.

Student: (Refer Time: 13:38).

Yeah so, they work these are physicist they work in a strange set of coordinate units where speed of light is 1 ok. So, everything is normalized very nicely to 1 right.

(Refer Slide Time: 13:53)



The screenshot shows a Jupyter Notebook window with the following content:

```
(make cylinder (center 0 0) (height infinity)
              (radius r) (material air)))

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

Later objects in the `geometry` list take precedence over (lie "on top of") earlier objects, so the second `air` ($\epsilon = 1$) cylinder cuts a circular hole out of the larger cylinder, leaving a ring of width u .

Now, we don't know the frequency of the mode(s) ahead of time, so we'll just hit the structure with a broad Gaussian pulse to excite all of the (TM polarized) modes in a chosen bandwidth:

```
(define-param fcen 0.15) ; pulse center frequency
(define-param df 0.1) ; pulse width (in frequency)
(set! sources (list
  (make source
    (src (make gaussian-src (frequency fcen) (width df)))
    (component Ez) (center (+ r 0.1) 0))))
```

Finally, we are ready to run the simulation. The basic idea is to run until the sources are finished, and then to run for some additional period of time. In that additional period, we'll perform some signal-processing on the fields at some point with `harminv` to identify the frequencies and decay rates of the modes that were excited:

```
(run-sources 300)
(at-beginning output-epsilon)
(after-sources (harminv Ez (vector3 (+ r 0.1) fcen df)))
```

The signal-processing is performed by the `harminv` function, which takes four arguments: the field component (here `Ez`) and position (here $(r = 0.1, 0)$) to analyze, and a frequency range given by a center frequency and bandwidth (here, the same as the source pulse). Note that we wrap `harminv` in `(after-sources ...)`, since we only want to analyze the frequencies in the source-free system (the presence of a source will distort the analysis). At the end of the run, `harminv` prints a series of lines (beginning with `harminv:`) to make it easy to scan for

So, it takes a little getting used, but once you get to used it is very powerful everything is normal yeah speed of light is 1 wave length is 1. So, yeah whatever then you run it for some time 300 ok. So, now, let us imagine we have done the simulation. How will you interpret the output? Output will be fields that all times and in positions now what.

Student: (Refer Time: 14:20).

No, but we are exciting into the wideband, but I am not getting frequency information in my FDTDs output is what E as a function of space and time below frequency. So, what do I have to do?

Student: (Refer Time: 14:36).

That is yeah. So, he is saying go around the structure and find out where the nodes are that is a very crude way is there a smarter way. Find out the Fourier transform I have the time domain in the delta calculate the Fourier transforms that is the much smarter way it will give me not just the frequency peaks, but also how resonate they are how sharp they are right. So, these guys have a inbuilt function called harm inverse this is right this is like the inverse Fourier transforms.

So, the simulation itself says that run your simulation for the 300 time instance at the end of it after sources get over run the inverse Fourier transform at this position and some centre frequency is given to you ok.

(Refer Slide Time: 15:25)

```

(setl sources (list
  (make source
    (src (make gaussian-src (frequency fcen) (fwidth df)))
    (component Ez) (center (+ r 0.1) 0))))

Finally, we are ready to run the simulation. The basic idea is to run until the sources are finished, and then to run for some additional
period of time. In that additional period, we'll perform some signal-processing on the fields at some point with harminv to identify the
frequencies and decay rates of the modes that were excited:

(run-sources+ 300
  (at-beginning output-epsilon)
  (after-sources (harminv Ez (vector3 (+ r 0.1)) fcen df)))

The signal-processing is performed by the harminv function, which takes four arguments: the field component (here Ez) and position (here
(r + 0.1,0)) to analyze, and a frequency range given by a center frequency and bandwidth (here, the same as the source pulse). Note that
we wrap harminv in (after-sources ...), since we only want to analyze the frequencies in the source-free system (the presence of a
source will distort the analysis). At the end of the run, harminv prints a series of lines (beginning with harminv:, to make it easy to grep for)
listing the frequencies it found:

harminv:, frequency, imag. freq., Q, amp1, amplitude, error
harminv:, 0.14716255528154, -7.3285628253225e-4, 80.683090881382, 0.00341388964004578, -0.00305022905194175-0.001533214023956484i, 1.0
harminv:, 0.14716255528154, -2.32636643253225e-4, 316.29272471914, 0.0286457651908165, 0.0193127882016469-0.0211564681361411i, 7.3252
harminv:, 0.17524675072263, -5.2234980117105e-5, 1677.48481212767, 0.0072113125656089, -8.12770506080189e-4-0.00716538314235885i, 1.

There are six columns (in addition to the label), comma-delimited for easy import into other programs. The meaning of these columns is
as follows. Harminv analyzes the fields f(r) at the given point, and expresses this as a sum of modes (in the specified bandwidth):


$$f(t) = \sum_n a_n e^{-i\omega_n t}$$


```

So, again this is something that they have coded and output comes in text. So, you can see that when they do this the peaks are extracted out. So, you get 3 peaks. So, that is the frequency and the Q the quality factor right that tells me if you have studied this before the quality tells me how good the resonance is.

So, the Q factor also comes out. So, for example, the first mode have the Q of 80 then 300 and then 1677. So, 1677 corresponds to the Q which lasts the longest ok.

(Refer Slide Time: 15:55)

We wrap `harminv` in (`after-sources ...`), since we only want to analyze the frequencies in the source-free system (the presence of a source will distort the analysis). At the end of the run, `harminv` prints a series of lines (beginning with `harminv:`, to make it easy to `grep` for) listing the frequencies it found:

```
harminv: frequency, imag. freq., Q, amp, amplitude, error
harminv: 0.11816157943463, -7.3185828251851e-4, 80.68369981382, 0.0034138934904579, -0.00399822985294175, 0.00153114829564841, 1.0
harminv: 0.147162559528314, -2.32636842353229e-4, 726.28272471934, 0.028645761988165, 0.029312782826689, 0.0211546483361431, 7.72572
harminv: 0.175246798722663, -5.2234988171685e-5, 476.48461212767, 0.00721131215656889, -8.1277058086189e-4, 0.007165383142358851, 1.1
```

There are six columns (in addition to the label), comma-delimited for easy import into other programs. The meaning of these columns is as follows. `harminv` analyzes the fields $f(r)$ at the given point, and expresses this as a sum of modes (in the specified bandwidth):

$$f(t) = \sum_n a_n e^{-i\omega_n t}$$

for complex amplitudes a_n and complex frequencies ω_n . The six columns relate to these quantities. The first column is the real part of ω_n , expressed in our usual 2 π units, and the second column is the imaginary part—a negative imaginary part corresponds to an exponential decay. This decay rate, for a cavity, is more often expressed as a dimensionless “lifetime” Q , defined by:

$$Q = \frac{\text{Re } \omega}{-2\text{Im } \omega}$$

(Q is the number of optical periods for the energy to decay by $\exp(-2x)$, and $1/Q$ is the fractional bandwidth at half-maximum of the resonance peak in Fourier domain.) This Q is the third column of the output. The fourth and fifth columns are the absolute value $|a_n|$ and complex amplitudes a_n . The last column is a crude measure of the error in the frequency (both real and imaginary)...if the error is much larger than the imaginary part, for example, then you can't trust the Q to be accurate. **Note:** this error is only the uncertainty in the signal processing, and tells you nothing about the errors from finite resolution, finite cell size, and so on!

An interesting question is how long should we run the simulation, after the sources are turned off, in order to analyze the frequencies.

And so, there is some theory of a inverse Fourier transform, but you all know that.

(Refer Slide Time: 15:59)

In this case, we found three modes in the specified bandwidth, at frequencies of 0.118, 0.147, and 0.175, with corresponding Q values of 81, 316, and 1677. (As was shown by Marcatili in 1969, the Q of a ring resonator increases exponentially with the product of ω and ring radius.) Now, suppose that we want to actually see the field patterns of these modes. No problem: we just re-run the simulation with a narrow-band source around each mode and output the field at the end.

In particular, to output the field at the end we might add an (`at-end output=efield-z`) argument to our `run-sources` function, but this is problematic: we might be unlucky and output at a time when the E_z field is almost zero (i.e. when all of the energy is in the magnetic field), in which case the picture will be deceptive. Instead, at the end of the run we'll output 20 field snapshots over a whole period $1/f_{cen}$ by appending the command:

```
(run-until (/ 1 fcen) (at-every (/ 1 fcen 20) output=efield-z))
```

Now, we can get our modes just by running e.g.:

```
unix$ mep fcm=0.118 pf=0.01 ring.ctl
```

After each one of these commands, we'll convert the fields into PNG images and thence into an animated GIF (as with the bend movie, above), via:

```
unix$ hstopng -Rz diskdared -C ring-ss-000000.00.h5 ring-ez-*.h5
unix$ convert ring-ez-*.png ring-ez-0.118.gif
```

The resulting animations for (from left to right) 0.118, 0.147, and 0.175, are below, in which you can clearly see the radiating fields that produce the losses:

And so, the next thing I could do is pick each one of those frequencies and re run the simulation with the narrow band at those frequencies that will excite only that mode and not the others right.

(Refer Slide Time: 16:15)

Now, we can get our modes just by running e.g.:

```
unix$ mep fcm=0.118 of=0.01 ring.ctf
```

After each one of these commands, we'll convert the fields into PNG images and thence into an animated GIF (as with the bend movie, above), via:

```
unix$ htopng -Ez disblured -C ring-eps-000000.00.05 ring-eps-1.05  
unix$ convert ring-eps-1.png ring-eps-0.118.gif
```

The resulting animations for (from left to right) 0.118, 0.147, and 0.175, are below, in which you can clearly see the radiating fields that produce the losses:

(Each of these modes is, of course, doubly-degenerate according to the representations of the $C_{\infty v}$ symmetry group. The other mode is simply a slight rotation of this mode to make it odd through the x axis, whereas we excited only the even modes due to our source symmetry. Equivalently, one can form clockwise and counter-clockwise propagating modes by taking linear combinations of the even/odd

So, what they are showing here are those three modes. So, the first mode the second mode and the third mode and these are again gif files produced. So, this is you can see this had the lowest Q right.

(Refer Slide Time: 16:30)

complex amplitudes \tilde{a}_m . The real counter is a crude measure of the error in the frequency (both real and imaginary)...a real error is much larger than the imaginary part, for example, then you can't trust the Q to be accurate. **Note:** this error is only the uncertainty in the signal processing, and tells you nothing about the errors from finite resolution, finite cell size, and so on!

An interesting question is how long should we run the simulation, after the sources are turned off, in order to analyze the frequencies. With traditional Fourier analysis, the time would be proportional to the frequency resolution required, but with harmvib the time is much shorter. Here, for example, there are three modes. The last has a Q of 1677, which means that the mode decays for about 2000 periods or about $2000 \cdot 0.175 = 10^4$ time units. We have only analyzed it for about 300 time units, however, and the estimated uncertainty in the frequency is 10^{-7} (with an actual error of about 10^{-6} , from below!) In general, you need to increase the run time to get more accuracy, and to find very high Q values, but not by much—in our own work, we have successfully found $Q = 10^8$ modes by analyzing only 200 periods.

In this case, we found three modes in the specified bandwidth, at frequencies of 0.118, 0.147, and 0.175, with corresponding Q values of 81, 316, and 1677. (As was shown by Marcatili in 1969, the Q of a ring resonator increases exponentially with the product of ω and ring radius.) Now, suppose that we want to actually see the field patterns of these modes. No problem: we just re-run the simulation with a narrow-band source around each mode and output the field at the end.

In particular, to output the field at the end we might add an `(at-end output-efield-z)` argument to our `run-sources` function, but this is problematic: we might be unlucky and output at a time when the E_z field is almost zero (i.e. when all of the energy is in the magnetic field), in which case the picture will be deceptive. Instead, at the end of the run we'll output 20 field snapshots over a whole period $1/f_{cen}$ by appending the command:

```
(run-until (/ 1 fcen) (at-every (/ 1 fcen 20) output-efield-z))
```

Now, we can get our modes just by running e.g.:

```
unix$ mep fcm=0.118 of=0.01 ring.ctf
```

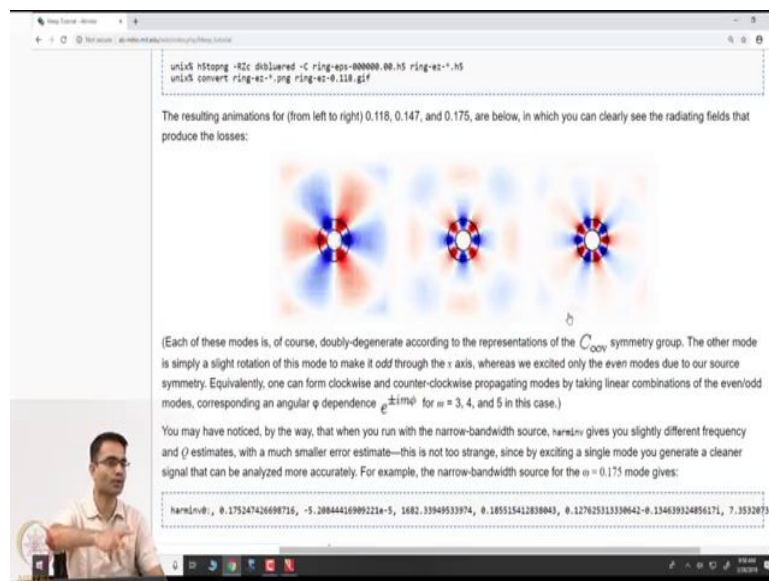
What were our Qs 80316 and about 1600. So, this is the lowest Q and that is you can see that a lot of the field is leaking out right outside the waveguide outside this resonator lot of field is

there as the Q becomes larger and finely larger over here. Here the field is much more strongly concentrated inside the resonator.

Student: Right.

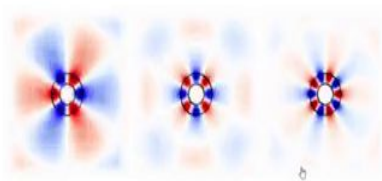
So, I have made a structure and I am able to identify these are the resonant modes by using simple FDTD and inverse Fourier transform signal processor ok.

(Refer Slide Time: 17:03)



unix\$ hftopng -Rz disabled -C ring-eps-00000.00.h5 ring-eps-1.h5
unix\$ convert ring-eps-1.png ring-eps-0.118.gif

The resulting animations for (from left to right) 0.118, 0.147, and 0.175, are below, in which you can clearly see the radiating fields that produce the losses:



(Each of these modes is, of course, doubly-degenerate according to the representations of the C_{6v} symmetry group. The other mode is simply a slight rotation of this mode to make it odd through the x axis, whereas we excited only the even modes due to our source symmetry. Equivalently, one can form clockwise and counter-clockwise propagating modes by taking linear combinations of the even/odd modes, corresponding an angular φ dependence $e^{\pm i m \varphi}$ for $m = 3, 4,$ and 5 in this case.)

You may have noticed, by the way, that when you run with the narrow-bandwidth source, `harm1nv` gives you slightly different frequency and Q estimates, with a much smaller error estimate—this is not too strange, since by exciting a single mode you generate a cleaner signal that can be analyzed more accurately. For example, the narrow-bandwidth source for the $m = 0.175$ mode gives:

```
harm1nv@: 0.175147426698716, -5.20844416989221e-5, 1682.33949533974, 0.18515412838043, 0.127625113338642-0.134639324856171i, 7.35328734
```

So, that is to sort of show you what should you say in practice how would you use FDTD and is very very powerful design any structure excite any fields right. And, they have lots of other solid examples for photonic crystals and other structures like that and its easiest to run it on a Linux machine. So, let brings to an end this demo work on FDTD.