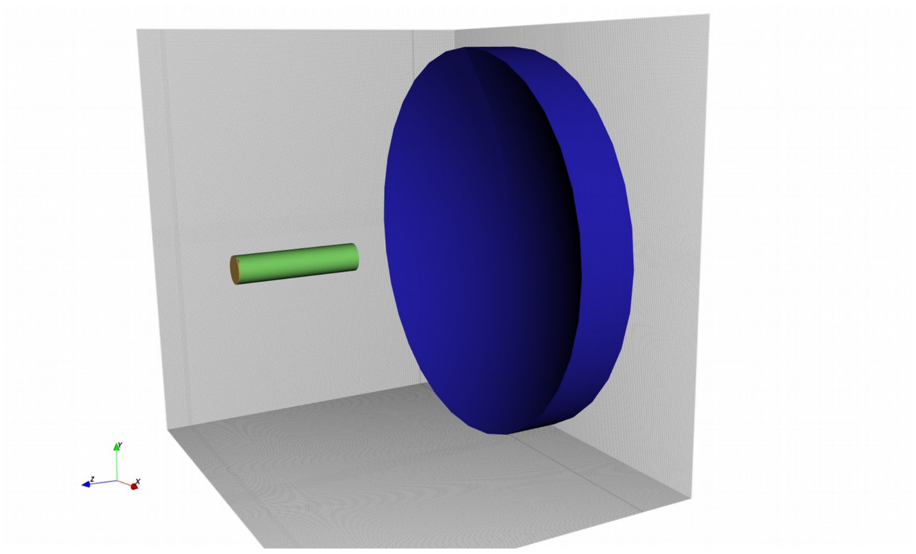
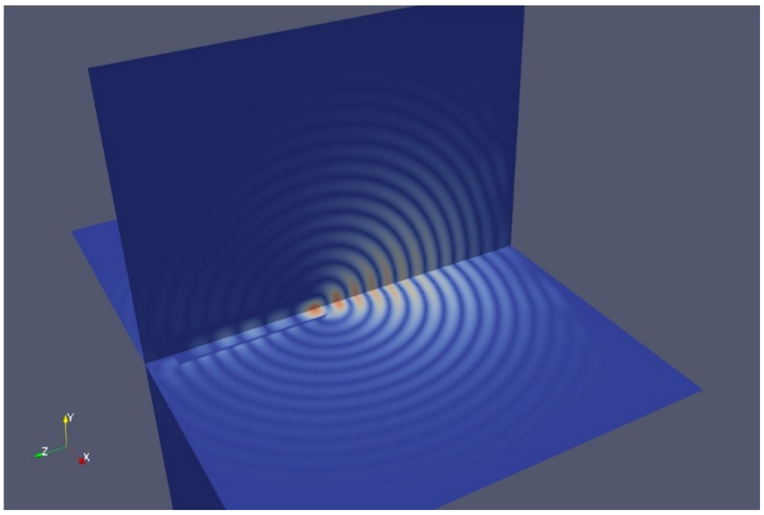
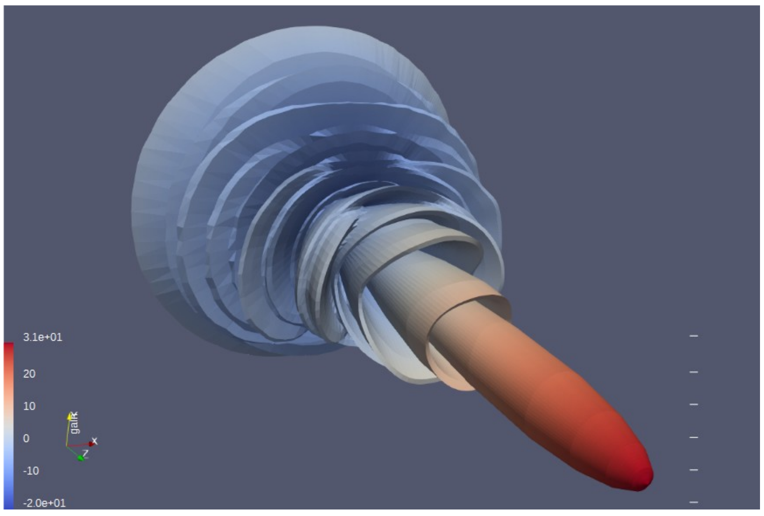
How To Set Up a Parabolic Reflector Simulation in openEMS

Dr Paul Klasmann (2E0PMK)

paulklasmann@hotmail.com







Aim of Exercise

The following topics will be covered in this tutorial.

- Setting up constants and variables.
- How to define the parabola shape and obtain a solid 3D model.
- How to define and setup an open ended circular waveguide as the feed antenna.
- Mesh and simulation setup.
- How to obtain plots for S_{11} and various ways to plot the far fields.
- How to add dump boxes to obtain various E-field cuts and display them in Paraview.
- How to export the physical model to VTK or STL.
- Final remarks.

Introduction

This tutorial describes a script to simulate a 340 mm diameter parabolic reflector antenna illuminated by a simple open ended circular waveguide as the “feed horn”. The focal length of the reflector is 155 mm and these dimensions can easily be changed by the user. The focal point is at the origin of the simulation domain (x=y=z=0). The phase center of the feed/horn should coincide at the origin.

The circular waveguide can be replaced by an antenna of your design, but it’s kept simple here to demonstrate the setup. The feed has a diameter of 19 mm and is chosen to work over the Ku-band satcoms frequency allocation which is 10.7 to 14.5 GHz. The frequency at which farfield patterns will be plotted is 12 GHz. The aim of the simulation is to obtain the S-parameters for the feed, 2D and 3D far field radiation patterns. 3D patterns will be displayed in Octave as well as ParaView, and two 2D E-field cuts will be displayed in ParaView. The E-fields can be animated and saved to an AVI video file.

Script Walkthrough

Comments in Matlab or Octave begin with a %.
We start by closing any open windows, clearing the console and defining the constants and units in mm with:

```
close all
clear
clc
% Simulation constants
physical_constants; % calls a function that initializes c, ep0, mu0 and eta0
unit = 1e-3; % all lengths in mm
```

Next we define a number of parameters that we will be using throughout the script:

```
wg.radius  = 19/2;       % Horn radius
wg.length = 100;         % Horn length in z-direction
wg.thickness = 2;        % Horn's wall thickness
FL=155;                  % Reflectors focal length
R = 170;                 % Reflector radius in mm
A = pi*(R*unit)^2;       % Aperture Area

% frequency range of interest Ku Satcom Band
f_start =  10e9;         % Start Frequency
f_stop  =  15e9;         % Stop Frequency
f0 = 12e9;               % Frequency of interest
```

Define the Parabola

The following lines define the profile of the parabola and adds some extra points so that a closed polygon can be generated. This allows the function AddRotPoly() to generate a body of revolution (BOR) solid object.

First we define a vector to contain the x-axis values from 0 to the reflector's radius 'R' in 0.5mm increments. The equation for a parabola normal to the z-axis is defined to calculate z for every value of x. The term "-FL" moves the position of the reflector so that the focal point is at the origin for convenience. The x and z coordinates are stored as an array "curve". The array "extrapoints" contain the extra points of the polygon to form a closed shape. Next, the contents of the "curve" and "extrapoints" arrays are combined into a single array called "coords", this will be passed to the AddRotPoly() that comes later.

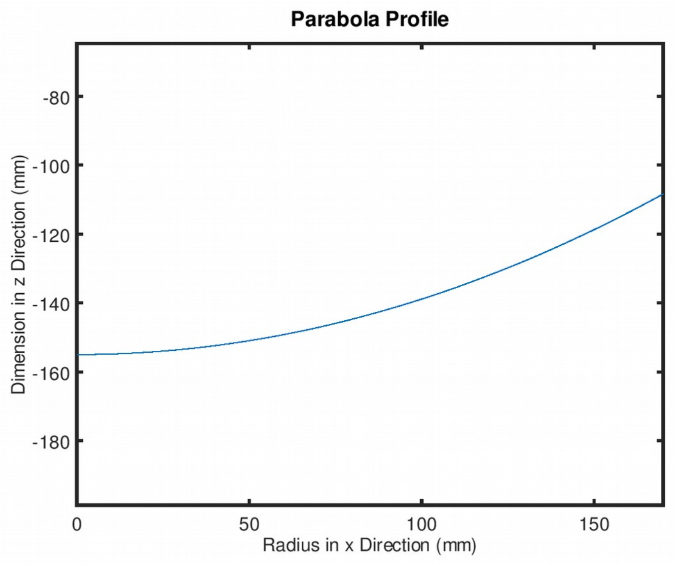
To check the profile visually we can plot the profile of the parabola on a graph with equal unit spacing on both axis to preserve the correct aspect ratio of 1:1 so that it does not appear distorted. This is just to check that what is being constructed is what we expect.

```
x=[0:0.5:R];                      % Vector to store x values for parabola

z=(((x).^2)/(4*FL))-FL;           % Parabola curve definition in z axix and offset by FL
curve = [x;z];                    % Store x and z coordinates in "curve"
depth = ((2*R)^2)/(16*FL);        % Calculate depth of reflector, depth=D^2/16FL
extrapoints = [R 0 0; -FL-2 -FL-2 0]; % Extra coords to make solid reflector
coords = [curve,extrapoints];         % Full coords for RotPoly stored here

plot(x,z);                            % Plot parabola to check
axis equal                            % Scale axis equally for an aspect ratio of 1:1
set(gca, "linewidth",2, "fontsize", 14 )
xlabel( 'Radius in x Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
title( 'Parabola Profile', 'FontSize', 16 );
```

An example of the parabola is shown in the plot below. Note that this does not show the extra points to form the closed polygon.



Simulation and Mesh Setup

The simulation domain is set up and the Cartesian mesh is defined to encompass the antenna with at least a quarter of a wavelength at the lowest frequency between the structure and the nearfield dump planes. These are defined later but here we allow a simulation domain that is 380 x 380 x 500 mm in the x, y and z plane respectively.

```
% Size of the simulation box, should be at least lambda/4 at lowest frequency
SimBox = [380 380 500];              % Size in x, y and z directions
```

The FDTD structure is defined and it's arguments set the number of time steps (NrTS), here we have 3500 time steps so that the simulation does not take too long. The end criteria is set to -33 dB. Due to the nature of this simulation, it can run for a very long time because the energy is reflected back into the feed a number of times. For the purpose of demonstration, we can truncate the simulation so that most of the energy has decayed to -33 dB or 3500 time steps. The simulation will exit depending on which is reached first.

The excitation is defined as a Gaussian pulse using the start and stop frequencies to control the pulse period.

```
% Initialise the FDTD structure
FDTD = InitFDTD( 'NrTS', 3500, 'EndCriteria', 0.5e-3 );    % End criteria -33 dB
FDTD = SetGaussExcite(FDTD,0.5*(f_start+f_stop),0.5*(f_stop-f_start));
```

Here the absorbing boundary is defined. openEMS provides two types of absorbing boundary, both are listed below and one of them must be commented out. The PML is a perfectly matched layer and can absorb an incident wave at a range of angles of incidence. The other absorbing boundary is the MUR method and this is most effective when the incident wave is normal to it's surface. The MUR method is simpler and results in a faster simulation run time. You can try it by commenting out the line that uses PMLs and uncomment the line that uses the MUR method. The number in PML_x sets the number of mesh cells that are used for the PML region. It is recommended to use 8 cells although 6 to 20 cells can be specified.

Note, MUR is named after G. Mur who first implemented this method, it should not be confused with permeability!

```
BC = {'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8'}; % PML boundary
%BC = {'MUR' 'MUR' 'MUR' 'MUR' 'MUR' 'MUR'}; % Simple MUR boundary
FDTD = SetBoundaryCond( FDTD, BC );   % Store FDTD and BC in FDTD structure
```

The following line sets maximum cell resolution. This is the cells per wavelength value that must be specified by dividing the speed of light by the lowest frequency to obtain the wavelength, which is then divided by the unit which is in millimetres. Finally this is divided by the number of mesh cells that we want per wavelength. 20 cells per wavelength is quite a fine mesh but we want to represent the parabola accurately. You may try 15 cells per wavelength for faster run times and compare the accuracy.

```
% Initialize the mesh with the "air-box" dimensions and key mesh lines for ports
max_res = c0 / (f_stop) / unit / 20; % cell size: lambda/20
```

The mesh is defined such that important features/edges of the model, excitations, probes and dump boxes lie on the grid. An easy way to do this is to construct the geometry with variable names defining important dimensions so that if the dimension is changed, you don't have to edit the mesh definition. This has been done in the following lines that define the mesh for the x, y and z axis. The SmoothMeshLines() function smooths the mesh lines between the fixed mesh lines and the rest of the mesh.

```
mesh.x = [-SimBox(1)/2 -wg.radius 0 wg.radius SimBox(1)/2];
% create a smooth mesh between specified fixed mesh lines
mesh.x = SmoothMeshLines( mesh.x, max_res, 1.4);
mesh.y = mesh.x;                          % Structure is axis symmetric in x & y
mesh.z = [-FL-50 -FL-2 -FL -FL+depth 0 wg.length-5 wg.length wg.length+1 wg.length+20];
mesh.z = SmoothMeshLines(mesh.z, max_res, 1.4);
```

The CSX structure is initialised and the defined mesh and the physical model of the parabola is added to the CSX structure. To do this, the DefineRectGrid() function is used with the arguments “unit” to specify that the mesh is in millimetres and the mesh stored in the 3 dimensional “mesh” structure is stored in CSX. AddMetal() makes the object “Parabola” a perfect electrical conductor (PEC) and AddRotPoly() takes the “coords” array with a number of other parameters to create the BOR parabola. The argument “x” is the plane where the parabola’s radius values are plotted. “z” is the axis of revolution.

```
% Setup CSXCAD geometry & mesh
CSX = InitCSX();
CSX = DefineRectGrid( CSX, unit, mesh );
CSX = AddMetal(CSX, 'Parabola');
% x is normal direction, "coords" is array to plot shape in 2D, z is the rotational
% axis direction.
CSX = AddRotPoly(CSX,'Parabola',10,'x',coords,'z');
```

The following lines define the waveguide as a rotated polygon. It would have been simpler to define a cylinder shell, however the discrete points of the polygon are used here to show how you can define a much more complicated feed horn that may not be cylindrical. Examples are multimode horns with both conical and cylindrical cross sections, choke rings or corrugated feed horns.

The opening of the cylindrical waveguide is at z=0 (focal point of the reflector). Refer to the wiki to understand how the points of a polygon are defined. The BOR is created in the same way as the parabolic reflector with the function AddRotPoly().

```
% Waveguide horn is a rotational polygon
% The open aperture of the horn is at z=0 (at the reflector's focal point)
CSX = AddMetal(CSX, 'Circular_Waveguide');
p(1,1) = wg.radius;
p(2,1) = 0;
p(1,2) = wg.radius+wg.thickness;
p(2,2) = 0;
p(1,3) = wg.radius+wg.thickness;
p(2,3) = wg.length;
p(1,4) = wg.radius;
p(2,4) = wg.length;
CSX = AddRotPoly(CSX,'Circular_Waveguide',10,'x',p,'z');
```

Because the circular waveguide is in free space and the rear of the waveguide is not in an absorbing boundary, a closed waveguide is required and this is done with a thin cylinder of PEC that is created with the AddCylinder() function. It is important that the cylinder facets lie on mesh lines in the z axis. Note the use of the variables/parameters used for the waveguide definition, this ensures that changes in the waveguide dimensions automatically update the position of the conducting end cap and port, and ensure that they always lie on mesh lines.

```
% End cap to prevent the radiation coming out of the back of the "horn"
CSX = AddMetal(CSX,'Cap');
CSX = AddCylinder(CSX,'Cap',10,[0 0 wg.length],[0 0 wg.length+1],wg.radius+wg.thickness);
% End of model geometry
```

Excitation

Since we are using a circular waveguide that we want to excite, we add a circular waveguide port. First we define the start and stop planes for the port box. The start plane coincides with the z-axis mesh lines at the closed end of the waveguide. The x and y-axis dimensions for the port box are to coincide with the inside radius of the waveguide. The length of the port box is 5 mesh cells. This represents the reference plane whereas the closed end of the waveguide is where the excitation begins. A TE11 mode is defined. The number “0” after the mode definition sets the E-field direction to be along the x-axis. More information on the AddCircWaveguidePort() function can be found on the wiki or in the function header.

```
% Apply the excitation
start=[-wg.radius -wg.radius wg.length ];
stop =[+wg.radius +wg.radius wg.length-5 ];
[CSX,port] = AddCircWaveGuidePort( CSX, 0, 1, start, stop, wg.radius*unit, 'TE11', 0, 1);
```

The way that we capture the fields that we want to plot is to define dump boxes or dump planes, here two dump planes are defined to record the E-field on two orthogonal planes, one normal to the x-axis and the other normal to the y-axis centred on the boresight of the antenna. Each plane spans the whole simulation extents but this does not have to be the case.

The first step is to use the AddDump() function, define it’s name and SubSampling if required. SubSampling can reduce the amount of data recorded. Define the start and stop coordinates for the dump box, if a plane is required then one axis corresponding to that plane is set to 0. The plane where the fields are recorded can be anywhere within the simulation domain, it does not have to be at the 0 position of that axis. We have called the two dump planes “Et_V_dump” and “Et_H_dump” and this will be the name of the data file saved to disk for each timestep/sub sampling that we specified to record.

```
% Dump box for Electric field at Phi=0 (vertical cut)
CSX = AddDump(CSX,'Et_V_dump', 'SubSampling', '4,4,4');
start=[0 -380/2 -FL];
stop =[0 380/2 wg.length+20];
CSX = AddBox(CSX,'Et_V_dump',0,start,stop);

% Dump box for Electric field at Phi=90 (horizontal cut)
CSX = AddDump(CSX,'Et_H_dump', 'SubSampling', '4,4,4');
start=[-380/2 0 -FL];
stop =[380/2 0 wg.length+20];
CSX = AddBox(CSX,'Et_H_dump',0,start,stop);
```

Near Field to Far Field

In order to plot the far fields of an antenna, a box must me defined that will be used to record the near fields. The near field to far field transformation is performed later but the data required is captured by the CreateNF2FFBox() function. The box must not lie inside the absorbing boundary, it must be at least one mesh cell inside the MUR or PML boundaries and at least a quarter of a wavelength at the lowest frequency away from the antenna.

```
% nf2ff calc
start = [mesh.x(10) mesh.y(10) mesh.z(10)];
stop  = [mesh.x(end-11) mesh.y(end-11) mesh.z(end-11)];
[CSX nf2ff] = CreateNF2FFBox(CSX, 'nf2ff', start, stop, 'Directions',
[1 1 1 1 1 1]),'OptResolution', max_res*2;
```

Use of the option ‘OptResolution’ is used to reduce the size of the nf2ff data files.

Setup Simulation Folder

The following lines create the simulation folder that will store all the files generated by the simulation. Here, the folder is called “tmp” and the XML file that will contain the simulation setup and physical structures and material properties is called “Parabola.xml”.

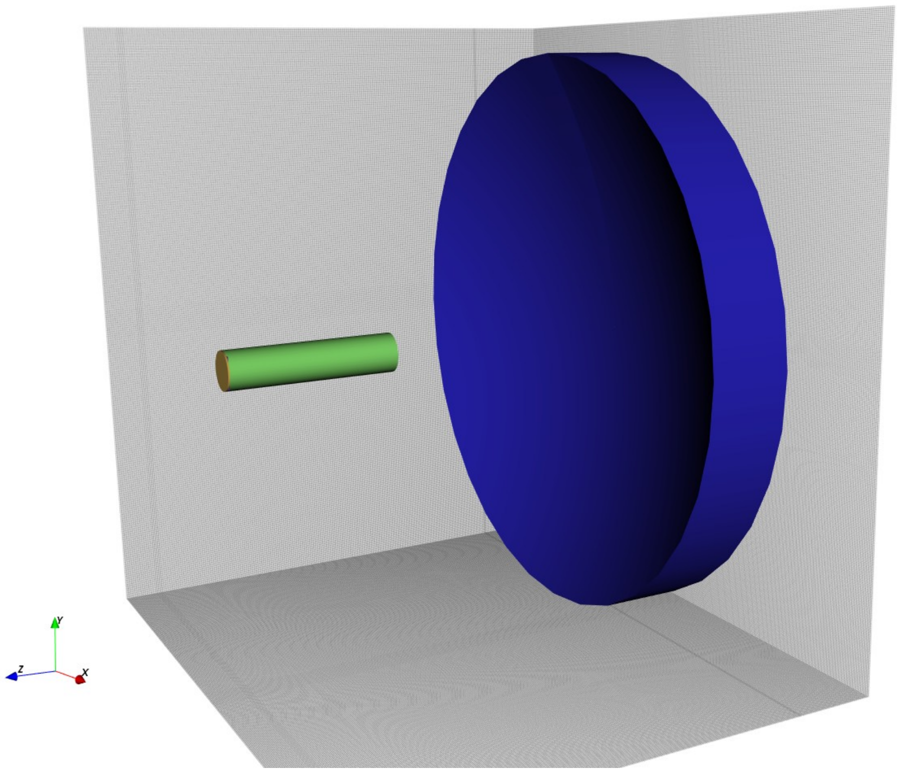
```
% Prepare simulation folder
Sim_Path = 'tmp';
Sim_CSX = 'Parabola.xml';
[status, message, messageid] = rmdir( Sim_Path, 's' ); % clear previous directory
[status, message, messageid] = mkdir( Sim_Path ); % create empty simulation folder
% Write openEMS compatible xml-file
WriteOpenEMS( [Sim_Path '/' Sim_CSX], FDTD, CSX );
```

Viewing the Antenna and Mesh

The CSXGeomPlot() function will launch the CSXCAD program that will enable you to check the mesh and structure of the antenna, positions of the excitations and dump boxes. The plane of the mesh can be moved in each axis to ensure that the mesh lines lie at feature edges, especially those of the excitation and dump planes. Each entity can be switched on or off by clicking on the light bulb symbol.

```
% Show the structure
CSXGeomPlot( [Sim_Path '/' Sim_CSX], ['--export-polydata-vtk=tmp'] );
```

We are now at the point where we can run the script up to this point and view what the antenna looks like.



The reflector, feed and end cap can be seen and the mesh can be checked. Note the areas of the mesh where the lines have been constrained to match the features as specified.

Running the Simulation

The RunOpenEMS() function is used to run the simulation. A number of options are available that will define how it is run. It may be useful to define options that will save a file of the mesh to view how the structure is meshed. This file can be opened with ParaView. To do this, uncomment “openEMS_opts” and add it as an argument to the RunOpenEMS() function. You can also specify the number of processor cores to use.

```
% Run openEMS
%openEMS_opts = '--debug-PEC --no-simulation'; % uncomment to visualise mesh
RunOpenEMS( Sim_Path, Sim_CSX, '--numThreads=4');
```

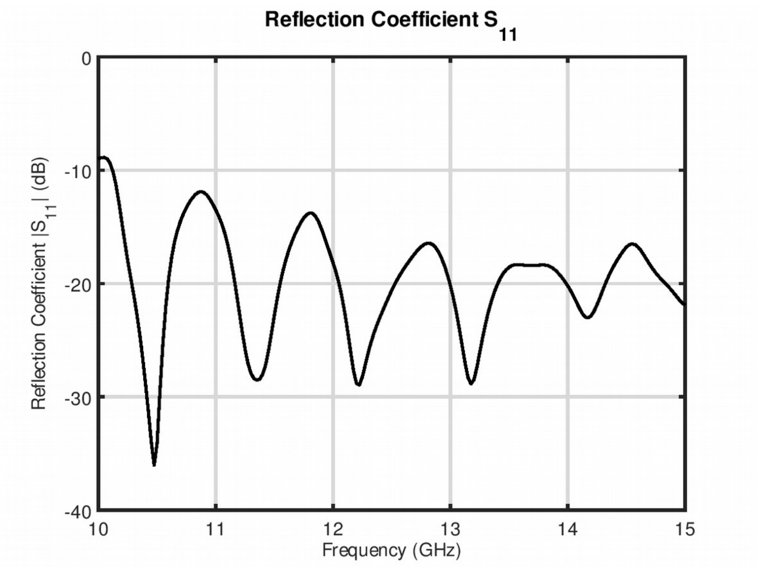
If the simulation is run at this point we will not see any plots of the data generated. Next we will add functions to visualise various antenna parameters. The lines below will calculate the S-parameter S_{11} and input impedance. This is done by creating a vector to store the frequency points that will be plot on the x-axis in 201 increments. The “port” is a structure that the function calcPort() uses to store a variety of data in both the time and frequency domains. More details are on the wiki or in the function header. Here we will use the frequency domain total voltage (port.uf.tot), total current (port.if.tot), reflected voltage (port.uf.ref) and incident voltage (port.uf.inc). For every frequency point, Zin and S_{11} is calculated.

```
% Postprocessing & do the plots
freq = linspace(f_start,f_stop,201);
port = calcPort(port, Sim_Path, freq);

Zin = port.uf.tot ./ port.if.tot;
s11 = port.uf.ref ./ port.uf.inc;
```

The next lines will plot S_{11} in dB and format the plot items.

```
% Plot reflection coefficient S11
figure
plot( freq/1e9, 20*log10(abs(s11)), 'k-', 'Linewidth', 2 );
ylim([-40 0]);
set(gca, "linewidth",2, "fontsize", 14 )
grid on
title( 'Reflection Coefficient S_{11}', 'FontSize', 16 );
xlabel( 'Frequency (GHz)','FontSize', 14 );
ylabel( 'Reflection Coefficient |S_{11}| (dB)','FontSize', 14 );
drawnow
```



Plotting Far Field Radiation Patterns

Now we will be setting up the angular range and phi cuts over which the far fields shall be plotted. Theta is defined over the range -180 to 180 degrees. Phi cuts are defined at 0, 45 and 90 degrees. Note the conversion to radians in the CalcNF2FF() function. In this function we define the name of the data structure (nf2ff), the path to the simulation data (Sim_Path), the frequency we wish to plot (f0), the theta range in radians and finally, the phi cut angles.

```
% NFFF plots

% calculate the far field at phi=0, 45 and at phi=90 degrees
thetaRange = (0:0.2:359) - 180;
disp( 'calculating far field at phi=[0 90] deg...' );
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, [0 45 90]*pi/180);
```

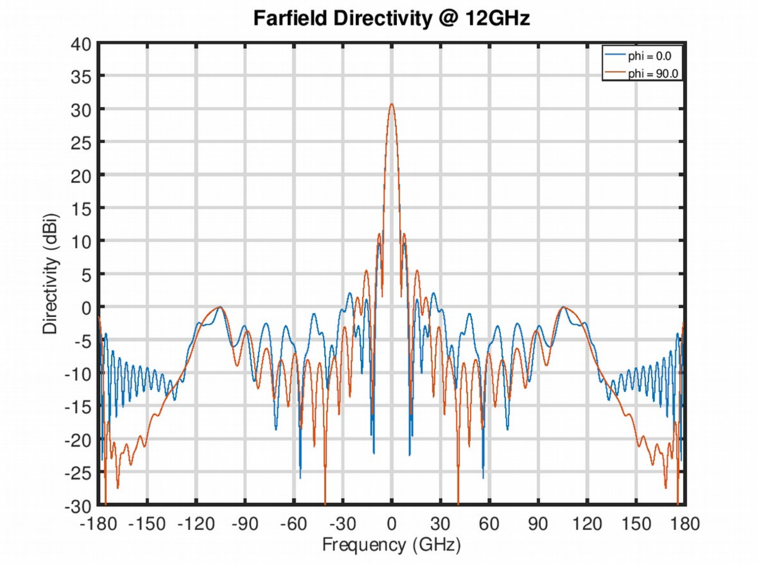
Here we calculate and print the peak directivity in dB, the theoretical gain for the aperture area and the efficiency.

```
Dlog=10*log10(nf2ff.Dmax);        % Calculate maximum Directivity in dB
G_a = 4*pi*A/(c0/f0)^2;           % Calculate theoretical gain for given aperture
e_a = nf2ff.Dmax/G_a;             % Calculate Efficiency

% Display some antenna parameters from above calculations
disp( ['radiated power: Prad = ' num2str(nf2ff.Prad) ' Watt']);
disp( ['directivity: Dmax = ' num2str(Dlog) ' dBi'] );
disp( ['aperture efficiency: e_a = ' num2str(e_a*100) '%'] );
```

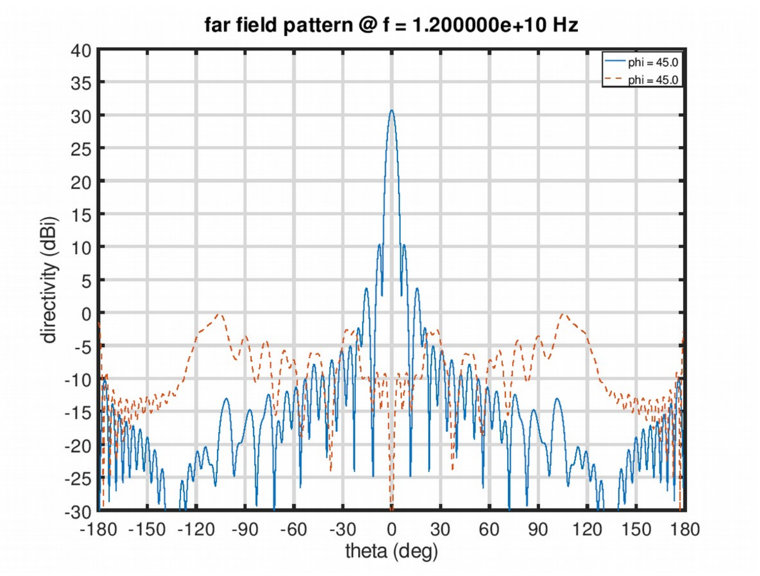
First we plot the directivity for the principle phi cuts on a Cartesian plot.

```
% Directivity
figure
plotFFdB(nf2ff,'xaxis','theta','param',[1 3]);
ylim([-30 40]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40 )
title( 'Farfield Directivity @ 12GHz','FontSize', 16 );
xlabel( 'Frequency (GHz)','FontSize', 14 );
ylabel( 'Directivity (dBi)','FontSize', 14 );
drawnow
```



This plot shows the copolar and Ludwig3 cross polar cuts at 45 degrees.

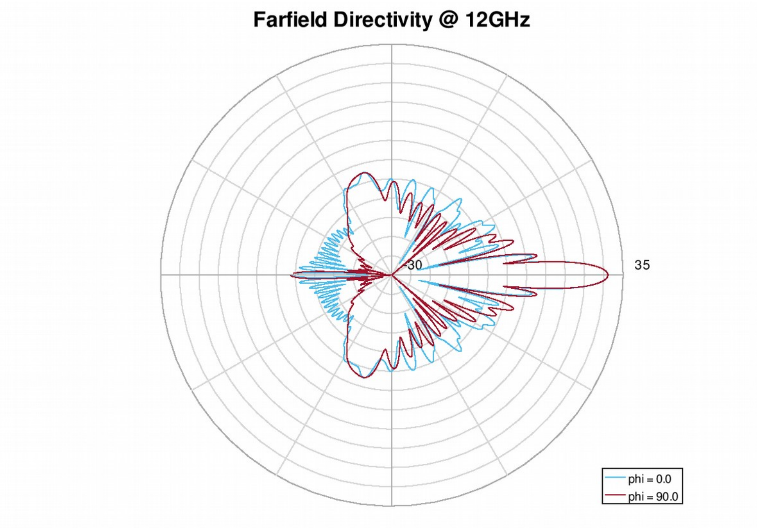
```
% Plot Ludwig3 cross polar
plotFFcocx(nf2ff,'xaxis','theta','param',[2]);
ylim([-30 40]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40 )
drawnow
```



Note, the cross polar plot will be compared to a simulation from other software to validate it’s accuracy.

A polar plot can be generated with:

```
% Polar plot
figure
leg=[]; %legend
polarFF(nf2ff,'xaxis','theta','param',[1 3],'logscale',[-30 35], 'xtics', 12);
title( 'Farfield Directivity @ 12GHz','FontSize', 16 );
xlabel( 'Frequency (GHz)','FontSize', 14 );
ylabel( 'Directivity (dBi)','FontSize', 14 );
drawnow
```



The axis are not labelled here, this is work in progress.

3D Polar Plots of the Far Field

3D polar plots can be made within Octave/Matlab, or VTK data can be opened in ParaView. First we define the angular ranges for phi and theta.

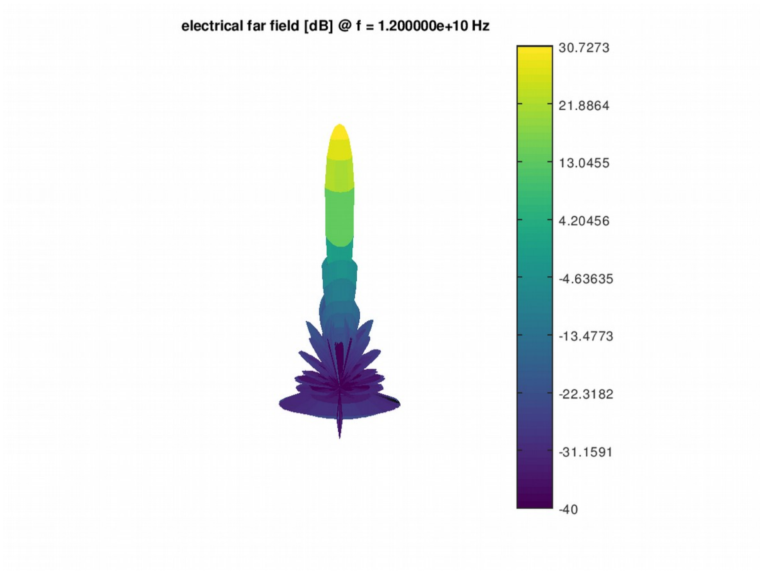
```
%% Calculate 3D pattern
phiRange = sort( unique( [-180:5:-100 -100:2.5:-50 -50:1:50 50:2.5:100 100:5:180] ) );
thetaRange = sort( unique([ 0:1:50 50:2.:100 100:5:180 ]));
```

CalcNF2FF() is used to generate the nf2ff data structure to store the far field data.

```
disp( 'calculating 3D far field...' );
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, phiRange*pi/180,
'Verbose',2,'Outfile','nf2ff_3D.h5');
```

Plot the 3D far field in Octave.

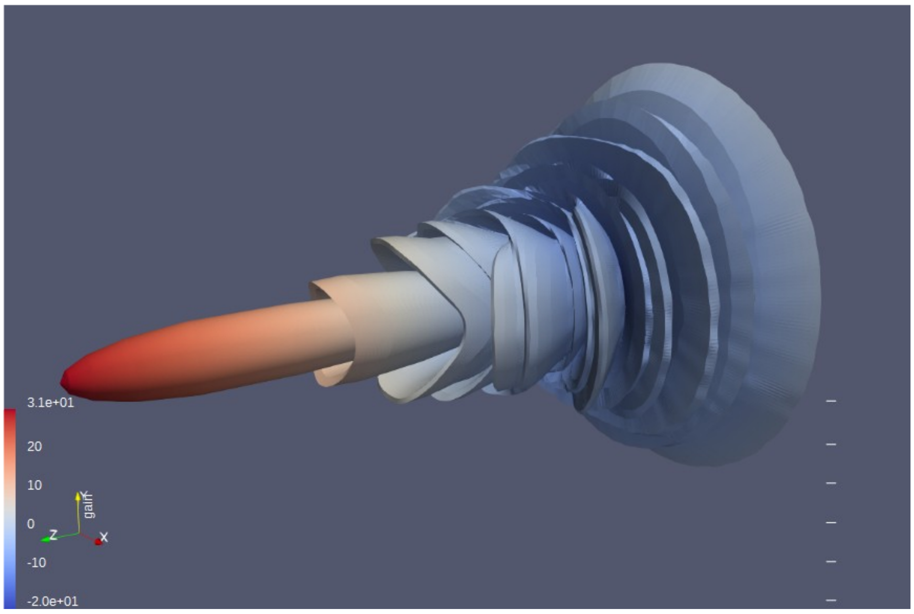
```
figure
plotFF3D( nf2ff, 'logscale', -40);        % plot 3D far field in dB
```

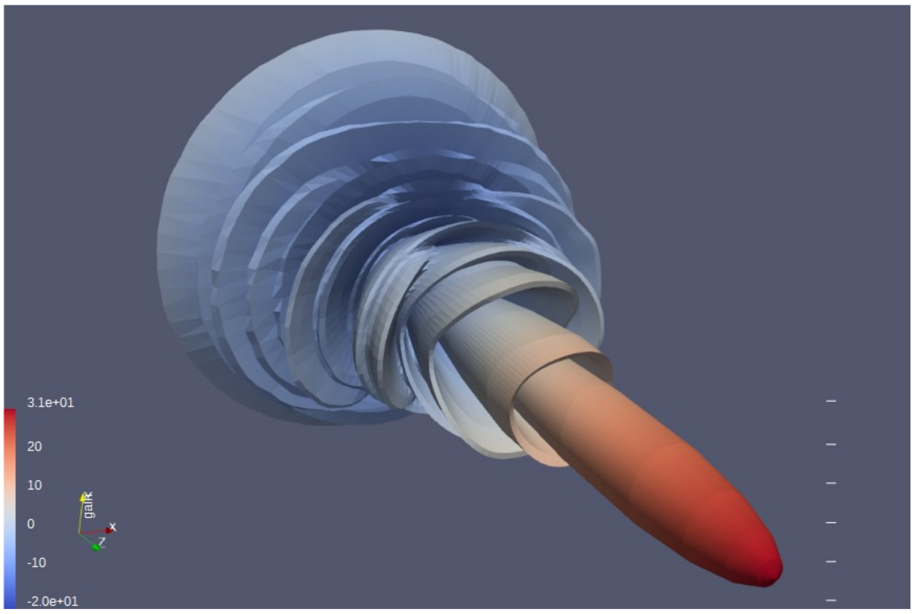


The last line saves data from nf2ff in VTK format to be opened in ParaView.

```
% Save far field in VTK to plot in ParaView
E_far_normalized = nf2ff.E_norm{1}/max(nf2ff.E_norm{1}(:));
DumpFF2VTK([Sim_Path '/Farfield.vtk'],E_far_normalized,thetaRange,phiRange,'scale',
0.005, 'logscale', -20, 'maxgain', 31);
```

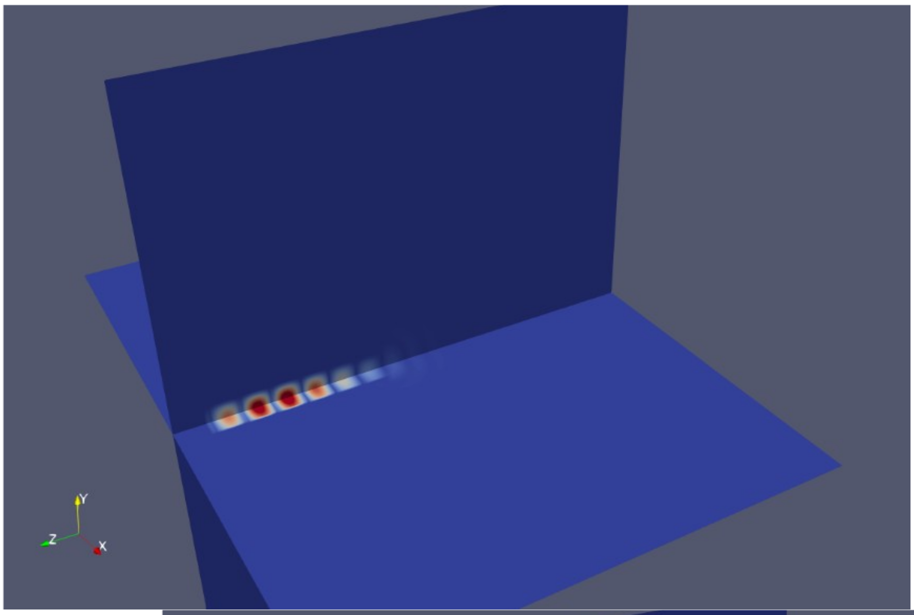
Two views of the far field generated with ParaView.

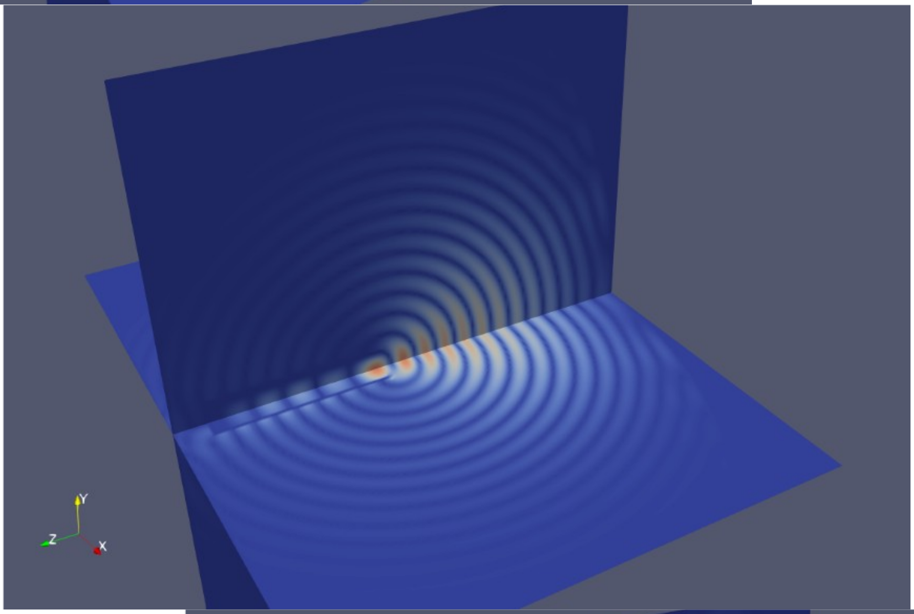


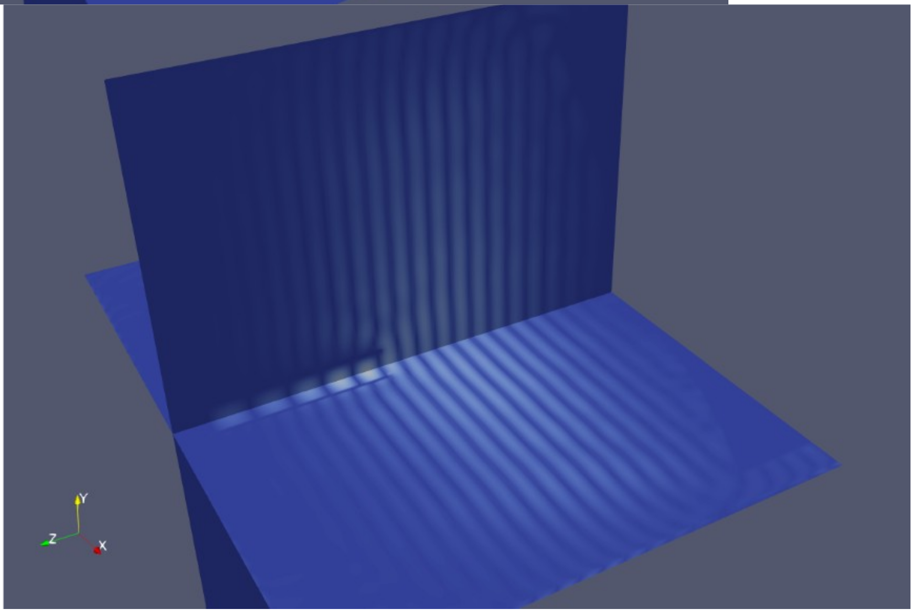


E-Field Dump Planes

The two E-field dump planes that we created earlier can now be plotted in ParaView for any timestep. Here are some results (with some adjustments in the colour scaling).





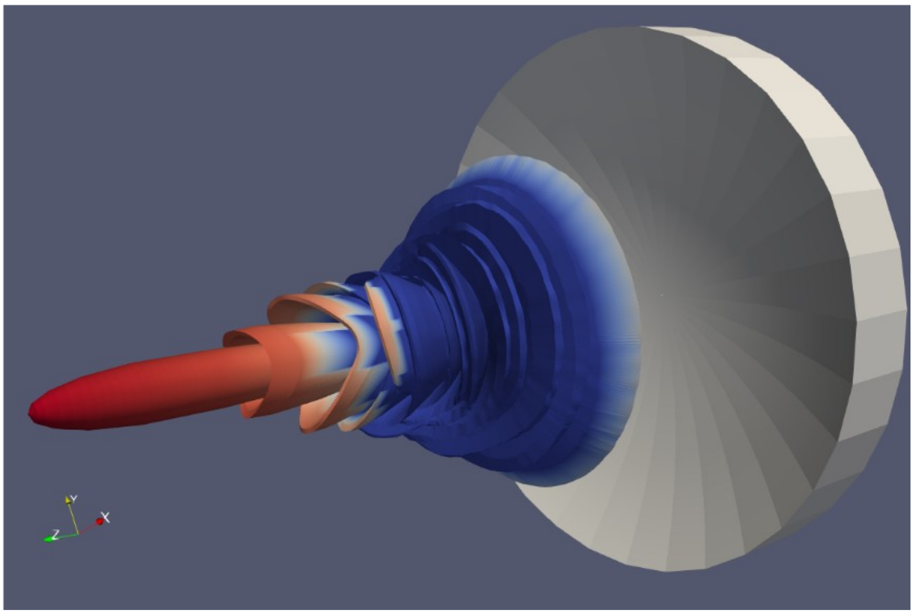


Export Physical Model to VTK

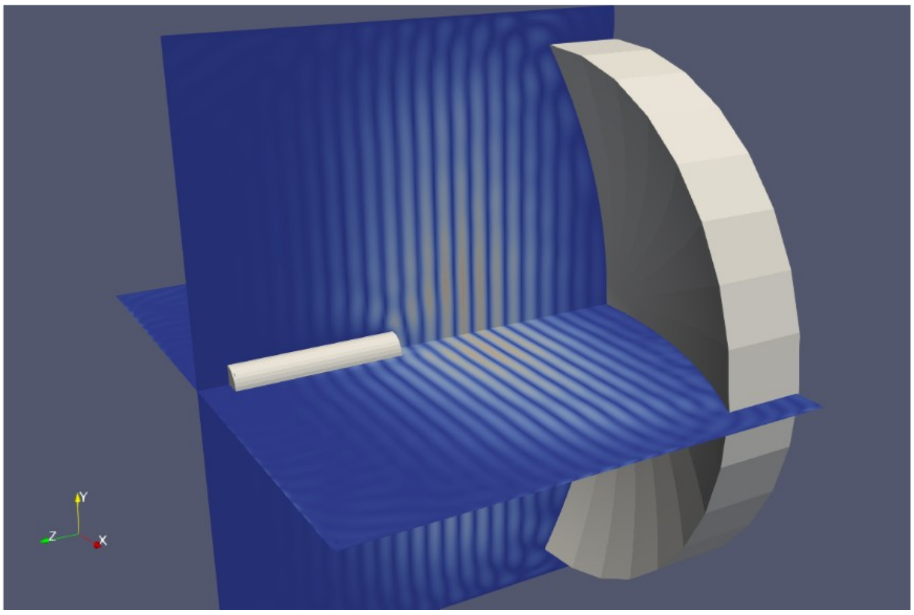
The physical model may be exported to VTK via CSXCAD by choosing File >> Export >> Polydata-VTK. Alternatively, it can be included in the sctipt:

```
CSXGeomPlot( [Sim_Path '/' Sim_CSX], ['--export-polydata-vtk=tmp'] );
```

Where “tmp” is the folder where the results are saved. The files have the extension .vtp and can now be opened with ParaView along with the far field as shown below.



The same can be done with any field quantity, here are the two E-field cuts shown with the antenna.



Export to STL

This can be achieved from CSXCAD by File >> Export >> STL. Alternatively use this command in your script:

```
CSXGeomPlot( [Sim_Path '/' Sim_CSX], ['--export-STL=tmp'] );
```

Conclusion and Final Comments

We have seen how a reflector antenna can be setup and simulated in openEMS to obtain some important antenna parameters and plot the far field radiation patterns and E-field cuts to give a visual indication of how the fields behave. Although we modelled a simple feed, this can be replaced with a more realistic and practical feed. It would also be useful to include feed support struts in a more realistic model to see how scattering from the struts will impact the sidelobes and cross polar performance. If you use a circularly polarised feed or a high performance linearly polarised feed, I recommend using four feed struts, one in each quadrant, not in the principle planes (not in the yz or xz planes).

Important points to consider are ensuring that mesh lines coincide with important features of the physical model. Although the simulation will run if you don't carefully set up the mesh to coincide with the model features, this will lead to inaccuracies especially if you are simulating a corrugated feed or a feed with choke rings.

More importantly, you must make sure that the mesh lines coincide with the port box. This means having a z axis mesh line at the excitation location as well as the probe location which are the start and stop dimensions of the box. If this is not done, the simulation will not run properly or you may not see any decay in energy. It is vitally important to make sure that the mesh coincides with the port box.

To visualise the mesh in ParaView, add the option argument "openEMS_opts = '--debug-PEC –no-simulation';" to the RunOpenEMS() function. This will save the mesh representation of the model so that you can open it in ParaView to check for errors. Use CSXCAD to view the mesh at the port by sliding the position of the mesh planes.

If you change the size of the reflector, make sure you change the mesh accordingly. The dump planes (and PML) must not be too close to or intersect the model.

The use of ParaView in this example demonstrates just some of the basic functionality that is available, and it is suggested that you look for dedicated ParaView tutorials to get the most from it.

Thank you, Thorsten for writing this software, making it freely available and for your generous assistance with my questions!

Full Script Listing

```
% Parabolic Reflector Example
%
% A simple Ku band feed is used to illuminate a 0.34m parabolic reflector
%
% Tested in Windows 7 with
%  - Octave 4.4.1
%  - openEMS v0.0.35
% Requires around 5 Gb of memory in addition to OS memory overheads
% Author: Paul Klasmann with a lot of help from Thorsten Liebig
% 01/12/2018
%
% In this script, the waveguide is the "horn" even though it is not flared.

close all
clear
clc
% Simulation constants
physical_constants; % calls a function that initializes c, ep0, mu0 and eta0
unit = 1e-3; % all lengths in mm

wg.radius  = 19/2;      % Horn radius
wg.length = 100;        % Horn length in z-direction
wg.thickness = 2;       % Horn's wall thickness
FL=155;                 % Reflectors focal length
R = 170;                % Reflector radius in mm
A = pi*(R*unit)^2;      % Aperture Area

% frequency range of interest Ku Satcom Band
f_start =  10e9;        % Start Frequency
f_stop  =  15e9;        % Stop Frequency
f0 = 12e9;              % Frequency of interest

x=[0:0.5:R];            % Vector to store x values for parabola

z=(((x).^2)/(4*FL))-FL; % Parabola curve definition in z axix and offset by FL
curve = [x;z];          % Store x and z coordinates in "curve"
depth = ((2*R)^2)/(16*FL);    % Calculate depth of reflector, depth=D^2/16FL
extrapoints = [R 0 0; -FL-2 -FL-2 0]; % Extra coords to make solid reflector
coords = [curve,extrapoints];         % Full coords for RotPoly stored here

plot(x,z);                            % Plot parabola to check
axis equal                            % Scale axis equally for aspect ratio 1:1
set(gca, "linewidth",2, "fontsize", 14 )
xlabel( 'Radius in x Direction (mm)', 'FontSize', 14 );
ylabel( 'Dimension in z Direction (mm)', 'FontSize', 14 );
title( 'Parabola Profile', 'FontSize', 16 );

% Size of the simulation box, should be at least lambda/4 at lowest frequency
SimBox = [380 380 500];               % Size in x, y and z directions

% Initialise the FDTD structure
FDTD = InitFDTD( 'NrTS', 3500, 'EndCriteria', 0.5e-3 ); % End criteria -33 dB
FDTD = SetGaussExcite(FDTD,0.5*(f_start+f_stop),0.5*(f_stop-f_start));
BC = {'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8' 'PML_8'}; % PML boundary
%BC = {'MUR' 'MUR' 'MUR' 'MUR' 'MUR' 'MUR'}; % Simple MUR boundary
FDTD = SetBoundaryCond( FDTD, BC );   % Store FDTD and BC in FDTD structure

% Initialize the mesh with the "air-box" dimensions and key mesh lines for ports
max_res = c0 / (f_stop) / unit / 20; % cell size: lambda/20

mesh.x = [-SimBox(1)/2 -wg.radius 0 wg.radius SimBox(1)/2];
% create a smooth mesh between specified fixed mesh lines
mesh.x = SmoothMeshLines( mesh.x, max_res, 1.4);
mesh.y = mesh.x;                      % Structure is axis symmetric in x & y
mesh.z = [-FL-50 -FL-2 -FL -FL+depth 0 wg.length-5 wg.length wg.length+1 wg.length+20];
mesh.z = SmoothMeshLines(mesh.z, max_res, 1.4);

%% Setup CSXCAD geometry & mesh
```

```
CSX = InitCSX();
CSX = DefineRectGrid( CSX, unit, mesh );
CSX = AddMetal(CSX, 'Parabola');
% x is normal direction, "coords" is array to plot shape in 2D, z is the rotational
% axis direction.
CSX = AddRotPoly(CSX,'Parabola',10,'x',coords,'z');

% Waveguide horn is a rotational polygon
% The open aperture of the horn is at z=0 (at the reflector's focal point)
CSX = AddMetal(CSX, 'Circular_Waveguide');
p(1,1) = wg.radius;
p(2,1) = 0;
p(1,2) = wg.radius+wg.thickness;
p(2,2) = 0;
p(1,3) = wg.radius+wg.thickness;
p(2,3) = wg.length;
p(1,4) = wg.radius;
p(2,4) = wg.length;
CSX = AddRotPoly(CSX,'Circular_Waveguide',10,'x',p,'z');

% End cap to prevent the radiation coming out of the back of the "horn"
CSX = AddMetal(CSX,'Cap');
CSX = AddCylinder(CSX,'Cap',10,[0 0 wg.length],[0 0 wg.length+1],wg.radius+wg.thickness);
% End of model geometry

%% Apply the excitation %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
start=[-wg.radius -wg.radius wg.length ];
stop =[+wg.radius +wg.radius wg.length-5 ];
[CSX, port] = AddCircWaveGuidePort( CSX, 0, 1, start, stop, wg.radius*unit, 'TE11', 0,
1);

% Dump box for Electric field at Phi=0 (vertical cut)
CSX = AddDump(CSX,'Et_V_dump', 'SubSampling', '4,4,4');
start=[0 -380/2 -FL];
stop =[0 380/2 wg.length+20];
CSX = AddBox(CSX,'Et_V_dump',0,start,stop);

% Dump box for Electric field at Phi=90 (horizontal cut)
CSX = AddDump(CSX,'Et_H_dump', 'SubSampling', '4,4,4');
start=[-380/2 0 -FL];
stop =[380/2 0 wg.length+20];
CSX = AddBox(CSX,'Et_H_dump',0,start,stop);

% nf2ff calc
start = [mesh.x(10) mesh.y(10) mesh.z(10)];
stop  = [mesh.x(end-11) mesh.y(end-11) mesh.z(end-11)];
[CSX nf2ff] = CreateNF2FFBox(CSX, 'nf2ff', start, stop, 'Directions',
[1 1 1 1 1 1]),'OptResolution', max_res*2;

% Prepare simulation folder
Sim_Path = 'tmp';
Sim_CSX = 'Parabola.xml';
[status, message, messageid] = rmdir( Sim_Path, 's' ); % clear previous directory
[status, message, messageid] = mkdir( Sim_Path ); % create empty simulation folder
% Write openEMS compatible xml-file
WriteOpenEMS( [Sim_Path '/' Sim_CSX], FDTD, CSX );
% Show the structure
%CSXGeomPlot( [Sim_Path '/' Sim_CSX] );
CSXGeomPlot( [Sim_Path '/' Sim_CSX], ['--export-polydata-vtk=tmp'] );

% Run openEMS
%openEMS_opts = '--debug-PEC --no-simulation'; % uncomment to visualise mesh
%RunOpenEMS( Sim_Path, Sim_CSX, '--numThreads=3');

% Postprocessing & do the plots
freq = linspace(f_start,f_stop,201);
port = calcPort(port, Sim_Path, freq);

Zin = port.uf.tot ./ port.if.tot;
s11 = port.uf.ref ./ port.uf.inc;
```

```
% Plot reflection coefficient S11
figure
plot( freq/1e9, 20*log10(abs(s11)), 'k-', 'Linewidth', 2 );
ylim([-40 0]);
set(gca, "linewidth",2, "fontsize", 14 )
grid on
title( 'Reflection Coefficient S_{11}', 'FontSize', 16 );
xlabel( 'Frequency (GHz)','FontSize', 14 );
ylabel( 'Reflection Coefficient |S_{11}| (dB)','FontSize', 14 );
drawnow

% NFFF plots

% calculate the far field at phi=0, 45 and at phi=90 degrees
thetaRange = (0:0.2:359) - 180;
disp( 'calculating far field at phi=[0 45 90] deg...' );
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, [0 45 90]*pi/180);

Dlog=10*log10(nf2ff.Dmax);       % Calculate maximum Directivity in dB
G_a = 4*pi*A/(c0/f0)^2;          % Calculate theoretical gain for given aperture
e_a = nf2ff.Dmax/G_a;            % Calculate Efficiency

% Display some antenna parameters from above calculations
disp( ['radiated power: Prad = ' num2str(nf2ff.Prad) ' Watt']);
disp( ['directivity: Dmax = ' num2str(Dlog) ' dBi'] );
disp( ['aperture efficiency: e_a = ' num2str(e_a*100) '%'] );

% Directivity
figure
plotFFdB(nf2ff,'xaxis','theta','param',[1 3]);
ylim([-30 40]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40 )
title( 'Farfield Directivity @ 12GHz','FontSize', 16 );
xlabel( 'Frequency (GHz)','FontSize', 14 );
ylabel( 'Directivity (dBi)','FontSize', 14 );
drawnow

% Plot Ludwig3 cross polar
plotFFcocx(nf2ff,'xaxis','theta','param',[2]);
ylim([-30 40]);
xlim([-180 180]);
grid on
set(gca,"linewidth",2, "fontsize", 14, "XTick", -180:30:180, "YTick", -30:5:40 )
drawnow

% Polar plot
figure
leg=[]; %legend
polarFF(nf2ff,'xaxis','theta','param',[1 3],'logscale',[-30 35], 'xtics', 12);
title( 'Farfield Directivity @ 12GHz','FontSize', 16 );
xlabel( 'Frequency (GHz)','FontSize', 14 );
ylabel( 'Directivity (dBi)','FontSize', 14 );
drawnow

%% Calculate 3D pattern
phiRange = sort( unique( [-180:5:-100 -100:2.5:-50 -50:1:50 50:2.5:100 100:5:180] ) );
thetaRange = sort( unique([ 0:1:50 50:2.:100 100:5:180 ]));

disp( 'calculating 3D far field...' );
nf2ff = CalcNF2FF(nf2ff, Sim_Path, f0, thetaRange*pi/180, phiRange*pi/180,
'Verbose',2,'Outfile','nf2ff_3D.h5');

figure
plotFF3D( nf2ff, 'logscale', -40);        % plot 3D far field in dB

% Save far field in VTK to plot in ParaView
E_far_normalized = nf2ff.E_norm{1}/max(nf2ff.E_norm{1}(:));
DumpFF2VTK([Sim_Path '/Farfield.vtk'],E_far_normalized,thetaRange,phiRange,'scale',
0.005, 'logscale', -20, 'maxgain', 31);
```

END OF DOCUMENT