Contents

QuickStart

Linux

Step 1: Install CUDA library.

Step 2: Edit makefile in the directory.

Step3: Use build_file script to build the directory for results.

Step 4: “./make” and “./te” in the terminal

Windows

Step 1: Install CUDA and visual studio.

Step 2: Open the project file.

Step 3: Edit the links to the libraries in the project file setting. Eigen library is used for matrix manipulation and the distribution is contained in the Github repositories. You need to check out C/C++ option in the project property to locate the right position according to your local directory position. CUDA also needs to be located according to your own CUDA position.

Step 4: Build the directory (Change the name of the “sample” directory to what you want.)

Step 5: Compile and run.

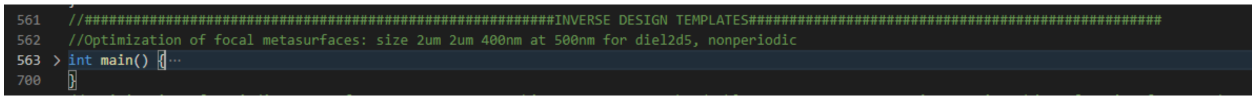
Test template

Various test templates for typical utilizations of the code are contained in the “testtemplate.cpp” file. User can copy paste whatever “int main(){……}” into the “test.cpp” file and test out. Edits can be made based on these templates to carry out users’ own simulation.

Topology Optimization of near-field focusing metasurfaces

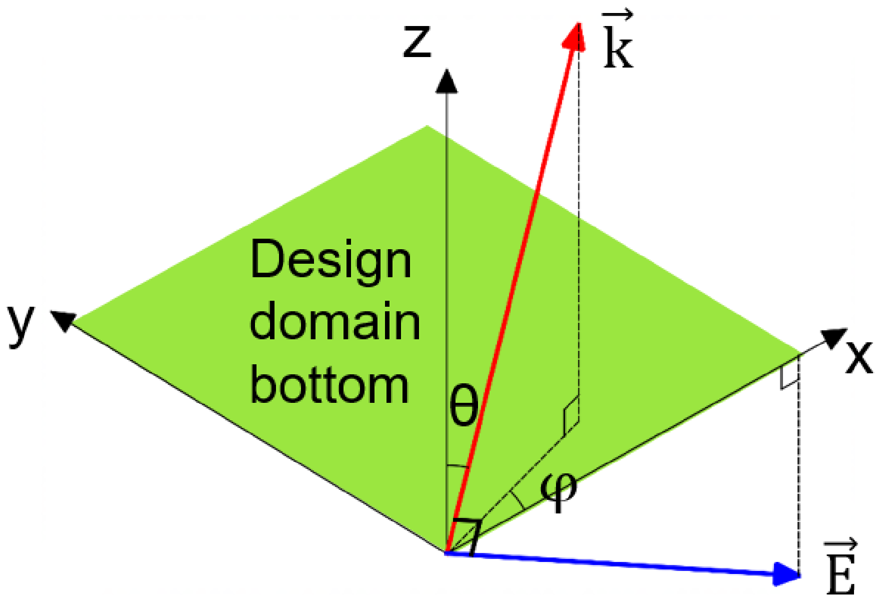
One example is to design a near-field focusing metasurface with topology optimization for random focal spot and wavelength from a dielectric slab. Due to the nature of DDA algorithm, material with refractive index below 1.6 is recommended for large structures.

The example in the testtemplate.cpp file is positioned as follow:



```
561   //################################################################INVERSE DESIGN TEMPLATES#################################################################
562   //Optimization of focal metasurfaces: size 2um 2um 400nm at 500nm for diel2d5, nonperiodic
563 > int main() {···
700   }
```

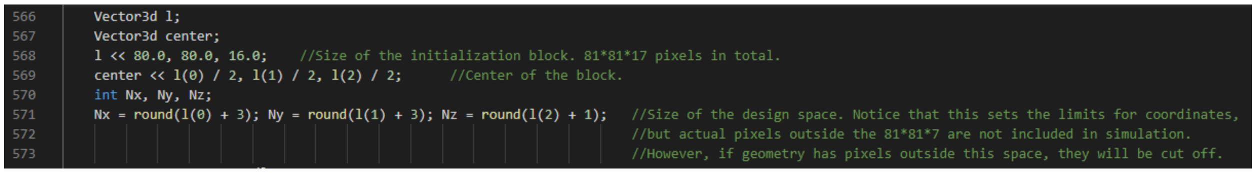
This template designs a metasurface of size 2um*2um*400nm (80*80*16 in terms of interval or 81*81*17 in terms of number of dipoles) with pixel size of 25nm. The input light is a plane wave at 500nm with angles defined as follows:



In this case, it is an input normal to the surface, x-polarized from below.

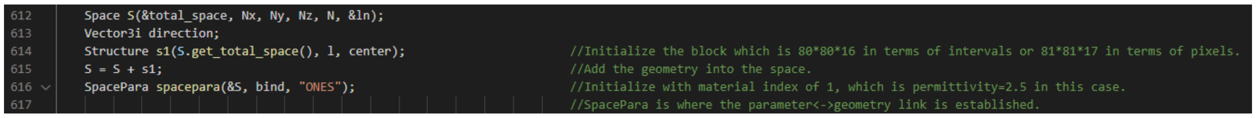
Various parameters can be changed to fulfill different purposes:

Size of the design space:



```
    Vector3d l;
    Vector3d center;
    l << 80.0, 80.0, 16.0;    //Size of the initialization block. 81*81*17 pixels in total.
    center << l(0) / 2, l(1) / 2, l(2) / 2;      //Center of the block.
    int Nx, Ny, Nz;
    Nx = round(l(0) + 3); Ny = round(l(1) + 3); Nz = round(l(2) + 1);   //Size of the design space. Notice that this sets the limits for coordinates,
                                                                        //but actual pixels outside the 81*81*7 are not included in simulation.
                                                                        //However, if geometry has pixels outside this space, they will be cut off.
```

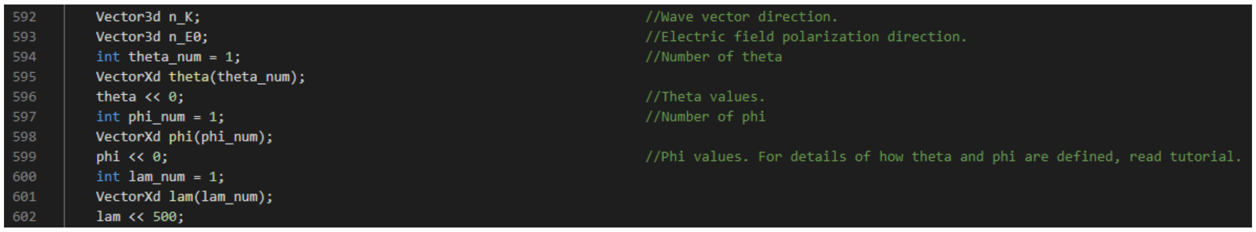
Only dipoles contained in the geometry will be simulated. The block defined by the Nx, Ny and Nz parameters serves as a coordinate system where the calculation takes place, but the void part not occupied by the geometries will not be simulated. This allows users to build several different geometries and push them one by one into the simulation space.



```
    Space S(&total_space, Nx, Ny, Nz, N, &ln);
    Vector3i direction;
    Structure s1(S.get_total_space(), 1, center);                    //Initialize the block which is 80*80*16 in terms of intervals or 81*81*17 in terms of pixels.
    S = S + s1;                                                      //Add the geometry into the space.
    SpacePara spacepara(&S, bind, "ONES");                           //Initialize with material index of 1, which is permittivity=2.5 in this case.
                                                                     //SpacePara is where the parameter<->geometry link is established.
```

In this case, as an example only one block is used as the design domain, so S=S+s1. If more geometries need to be add, users can repeat this process by doing S=S+s2… This can also be useful if topology optimization is not needed and DDA calculation for several different geometrical objects in the space need to be done, where users can build and push one by one the geometries and do the simulation.

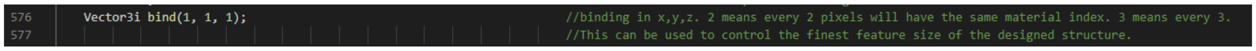
Input waves:



```
    Vector3d n_K;                                                    //Wave vector direction.
    Vector3d n_E0;                                                   //Electric field polarization direction.
    int theta_num = 1;                                               //Number of theta
    VectorXd theta(theta_num);
    theta << 0;                                                      //Theta values.
    int phi_num = 1;                                                 //Number of phi
    VectorXd phi(phi_num);
    phi << 0;                                                        //Phi values. For details of how theta and phi are defined, read tutorial.
    int lam_num = 1;
    VectorXd lam(lam_num);
    lam << 500;
```

Users can add multiple angles and wavelengths into the simulation and the object function will be averaged for all input parameters. In this case, a structure that optimizes the performance for an averaged value can be designed. (Notice that averaging wavelengths is much more memory consuming than averaging angles. Users can estimate the cost of adding one extra wavelength by double the memory of single wavelength simulation.)

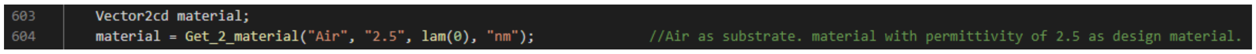
Binding:



```
    Vector3i bind(1, 1, 1);                                          //binding in x,y,z. 2 means every 2 pixels will have the same material index. 3 means every 3.
                                                                     //This can be used to control the finest feature size of the designed structure.
```

This allows the control of minimum feature size of the designed structure. 1 means each pixel changes its material index individually so that the minimum feature size should equal to the designated pixel size, which is d=25nm in this case. If 2 is defined, every two neighbouring pixels starting from the origin will have the same material index, so that the minimum feature size should be 2*d=50nm, etc. This practice does not change the accuracy of the DDA simulation since d is unchanged and the memory and time consumption also remains the same.

Materials:



```
    Vector2cd material;
    material = Get_2_material("Air", "2.5", lam(0), "nm");          //Air as substrate. material with permittivity of 2.5 as design material.
```

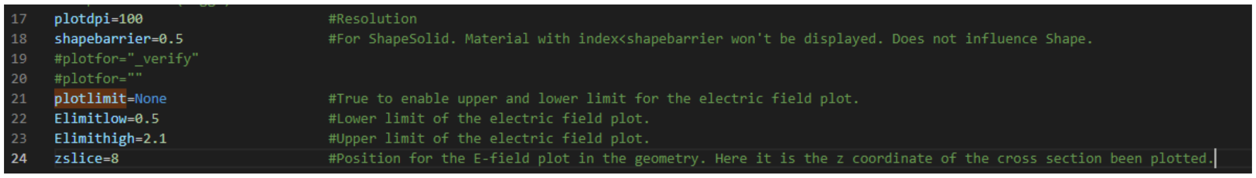
Air is for substrate and 2.5 is for the design material, which has the permittivity of 2.5. Some predefined materials are located in the "diel" directory. Users can define custom materials with the same format.

Visualization

Step 1: Install Anaconda if a python environment is not available.

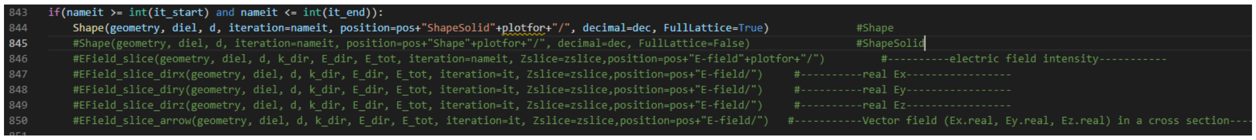
Step 2: Run plot.py in the terminal with "python3 plot.py name-of-result-directory iteration_start iteration_end". Plots of geometries and electric field distribution will be plot and saved in "Shape", "ShapeSolid" and "E-field". "Shape" is the structure plot with substrate. "ShapeSolid" is the same structure with the substrate hollowed out. “E-field” is the electric field distribution at a designated cross section of the space. Only results between iteration_start and iteration_end will be plotted.

The plot settings are as follows, but more setup can be down by editing the code directly.



```
plotdpi=100                         #Resolution
shapebarrier=0.5                    #For ShapeSolid. Material with index<shapebarrier won't be displayed. Does not influence Shape.
#plotfor="_verify"
#plotfor=""
plotlimit=None                      #True to enable upper and lower limit for the electric field plot.
Elimitlow=0.5                       #Lower limit of the electric field plot.
Elimithigh=2.1                      #Upper limit of the electric field plot.
zslice=8                            #Position for the E-field plot in the geometry. Here it is the z coordinate of the cross section been plotted.
```

Choose what plots need to be plot.



```
if(nameit >= int(it_start) and nameit <= int(it_end)):
    Shape(geometry, diel, d, iteration=nameit, position=pos+"ShapeSolid"+plotfor+"/", decimal=dec, FullLattice=True)            #Shape
    #Shape(geometry, diel, d, iteration=nameit, position=pos+"Shape"+plotfor+"/", decimal=dec, FullLattice=False)                #ShapeSolid
    #EField_slice(geometry, diel, d, k_dir, E_dir, E_tot, iteration=nameit, Zslice=zslice,position=pos+"E-field"+plotfor+"/")        #----------electric field intensity----------
    #EField_slice_dirx(geometry, diel, d, k_dir, E_dir, E_tot, iteration=it, Zslice=zslice,position=pos+"E-field/")     #----------real Ex-----------------
    #EField_slice_diry(geometry, diel, d, k_dir, E_dir, E_tot, iteration=it, Zslice=zslice,position=pos+"E-field/")     #----------real Ey-----------------
    #EField_slice_dirz(geometry, diel, d, k_dir, E_dir, E_tot, iteration=it, Zslice=zslice,position=pos+"E-field/")     #----------real Ez-----------------
    #EField_slice_arrow(geometry, diel, d, k_dir, E_dir, E_tot, iteration=it, Zslice=zslice,position=pos+"E-field/")   #-----------Vector field (Ex.real, Ey.real, Ez.real) in a cross section---
```

Validation

Lumerical FDTD

Step 1: Run matlab script “genetrate_geo_for_lumerical” with the correct directory and file name.

Step 2: Import the geometry into Lumerical FDTD with “binary import” option (choose nanometer for default settings).

COMSOL

The geometry can be generated with COMSOL-Matlab link by running “Build_geometry” matlab script. This allows better viewing of the geometry. If simulation needs to be done with COMSOL, using the binary file as material distribution is recommended compared to directly building up the geometry in COMSOL.

Example of COMSOL generated geometry: