Contents

1. A Short Introduction to DDA and Topology Optimization

Our DDA (discrete dipole approximation) implementation is based on methods described in previous literatures. [1–5]

The fundamental idea of DDA is to replace solid physical objects with a dense array of N dipoles, with equal intervals which are small compared to the wavelength and the object size.[1] With the index of the dipoles ranging from j=1…N, the polarization of each dipole is determined by the equation:

$$P_j = \alpha_j E_{ext,j} \text{ (Eq. 1)}$$

where P_j is the polarization of dipole j, α_j is the polarizability determined by the material at position j, $E_{ext,j}$ is the electric field at position j, equal to the sum of the input background field and the field from other N-1 dipoles. α_j has different forms based on different approximations. The most basic α_j is Clausius-Mossotti relation:

$$\alpha_j = \frac{3}{4\pi n}\frac{\varepsilon-1}{\varepsilon+2} \text{ (Eq. 2)}$$

where ε is the dielectric constant of the material. In our code, we utilized lattice dispersion relation (LDR) [2], which is a common choice of α_j in DDA implementations. For higher refractive index materials, another model called filtered coupled dipoles (FCD) is utilized.[5]

By separating the incident background field and the field from other dipoles, (Eq. 1) can be rewritten into:

$$P_j = \alpha_j(E_{inc,j} - \sum_{k\neq j} A_{jk}P_k) \text{ (Eq. 3)}$$

$$E_{inc,j} = E_0 \exp(ik \cdot r_j - i\omega t) \text{ (Eq. 4)}$$

$$A_{jk}P_k = \frac{\exp(ikr_{jk})}{r_{jk}^3}\left\{k^2 r_{jk} \times (r_{jk} \times P_k) + \frac{(1-ikr_{jk})}{r_{jk}^2} \times [r_{jk}^2 P_k - 3r_{jk}(r_{jk} \cdot P_k)]\right\} \ (j \neq k) \ \text{(Eq. 5)}$$

where $E_{inc,j}$ is the incident background field at position j, which is a plane wave and symmetric matrix A_{jk} represents the interaction between dipole j and k. For convenience, if we define $A_{jj} = \alpha_j^{-1}$ in addition to (Eq. 5), we can simplify (Eq. 5) for all dipole positions into a single matrix equation:

$$\boldsymbol{AP} = \boldsymbol{E} \ \text{(Eq. 6)}$$

where $\mathbf{P} = (P_1, P_2 \ldots P_N)$ and $\mathbf{E} = (E_{inc,1}, E_{inc,2} \ldots E_{inc,N})$ are 3N-dimensional vetors and $\boldsymbol{A}$ is $3N \times 3N$ symmetric matrix such that $A_{3j+l,3k+m} = (A_{jk})_{lm}$. The latter A_{jk} is the 3×3 matrix between dipole j and k in (Eq. 5). Thus, the problem of DDA simplifies to solving $\boldsymbol{P}$ in (Eq. 6) for initial condition $\boldsymbol{E}$. In our code, we used biconjugate gradient stabilized method (bicgstab) for the solution.

The CG (conjugrate-gradient) like methods, including bicgstab require matrix-vector products of the form $\cdot \mathbf{X}$, which is the most time-consuming bottleneck of the algorithm. As $\mathbf{A}$ has size of $3\mathrm{N} \times 3\mathrm{N}$ and $\mathbf{X}$ has the size of N, the product has a time complexity of $\mathrm{O}(N^2)$, which is extremely expensive, as N can be very big numbers.

Fortunately, fast-Fourier transform (FFT) can be implemented to reduce the time complexity to $O(Nln(N))$ because of the special form of matrix $\mathbf{A}$.[4] If we extract the diagonal elements of matrix $\mathbf{A}$ in Eq. 6, it can be written into:

$$\boldsymbol{A_{ij}} = \begin{cases} \boldsymbol{A'_{i-j}} & \boldsymbol{if\ i \neq j} \\ \mathbf{0} & \boldsymbol{if\ i = j} \end{cases} \quad \text{(Eq. 7)}$$

where the non-diagonal terms depend only on the difference of the indices. If we double the lattice in each dimension, regard A'_i and X_i as periodic in each dimension and set $X_i = 0$ for $N_x < i_x \leq 2N_x$, $N_y < i_y \leq 2N_y$, $N_z < i_z \leq 2N_z$, the product $Y_i \equiv (A \cdot X)_i$ is:

$$Y_i = \sum_{j_x=0}^{2N_x} \sum_{j_y=0}^{2N_y} \sum_{j_z=0}^{2N_z} A'_{i-j} \cdot X_j = \sum_{j}{}' A'_{i-j} \cdot X_j \ \text{(Eq. 8)}$$

which is a convolution. Thus, the result can be easily calculated with FFT. In practice, we implement FFT with CUDA library, using cuFFT on a NVIDIA GeForce RTX 2080 graphics card. Taking advantage of the superior parallel computing capability of GPU greatly boosts the speed of our DDA code, making it much faster than traditional commercial electromagnetic solvers based on CPU, such as Lumerical and COMSOL.

The space complexity of the algorithm seems to be $O(N^2)$, mainly due to the size of matrix **A**. However, just as what is shown in Eq. 7, the non-diagonal terms depend only on the relative position, **A** can be stored in $O(N)$. In practice, matrix **A** and the other vector **X** have to be first transmitted into GPU memory, and later transmitted back to memory after the convolution is finished. However, it is not necessary to transmit matrix **A** for every matrix vector product because the diagonal terms in matrix **A** depend on the material and the non-diagonal terms only depend on relative positions of the lattice points. When the geometry of the structure changes with the increase of iterations, material at each lattice point changes correspondingly to represent the correct geometry, but the relative spatial positions of the lattice points remain the same. Thus, most part of matrix **A** can be built and stored in GPU memory in the very beginning. For later matrix vector products, only the vector **X** needs to be transmitted.

The next step in inverse design after the electromagnetic problem is solved is to determine the gradient of the objective function to the design parameters ε:

$$\epsilon = \epsilon_1 + \varepsilon(\epsilon_2 - \epsilon_1) \text{ (Eq. 9)}$$

where ϵ_1 and ϵ_2 are the relative dielectric constant of the substrate and the designing material, while ϵ is the mixed virtue dielectric constant of the two. ε is the parameter between 0~1 that indicates the actual material and each lattice point has its individual ε. Principally, the final structure should be composed of parameters of 0 or 1 with no intermediate terms, but for the convenience of calculating the gradient, a continuous assumption has to be made between 0 and 1. In most of our simulations, intermediate terms reduce to a very small fraction with increasing iteration numbers. For the few intermediate terms left, we set values bigger than 0.5 to 1 while others to 0 to get the final structure. This can cause errors when iteration numbers are not sufficiently large.

The gradient of the objective function g relative to the parameters $\boldsymbol{\varepsilon}$ can be easily calculated with adjoint method:

$$\frac{\mathrm{dg}}{\mathrm{d}\boldsymbol{\varepsilon}} = \mathrm{g}_{\boldsymbol{\varepsilon}} - \boldsymbol{\lambda}^{\mathrm{T}}(\mathbf{A}_{\boldsymbol{\varepsilon}}\mathbf{P} - \mathbf{E}_{\boldsymbol{\varepsilon}}) \quad (\mathrm{Eq.}\,10)$$

$$\mathbf{A}^{\mathrm{T}}\boldsymbol{\lambda} = \mathrm{g}_{\mathbf{P}}^{\mathrm{T}} \quad (\mathrm{Eq.}\,11)$$

Eq. 11 is the adjoint problem of Eq. 6. Due to the symmetry of matrix $\mathbf{A}$, exactly the same matrix and method can be used to calculate $\boldsymbol{\lambda}$. It also preserves the same time complexity as the original problem. The gradients can then be further corrected by methods such as ADAM to achieve better convergence. Eventually, the gradients are timed with a fixed learning rate and added to $\boldsymbol{\varepsilon}$ to get the new structure. The full iteration is repeated until the object function converges.

2. How to use the parameter files (.ini)

Different .ini files are needed to perform different calculations. To designate which task to perform, type in the name of the model and the corresponding .ini file name. For example, if a 2D extruded geometry input into the model need to be topology optimized, type in 'EvoOpt 2D input' as 'name' and input '.\\tasks\\EvoOpt_2D_input.ini' as 'EvoOpt_2D_input_path'.

2.1. DDA input

This model calculates the electric field distribution with DDA for an arbitrary input geometry. The input geometry (grid) is always cubic, but the material distribution can vary as needed.

[Geometry]

pathCommonData=.\\sample\\grid_DDA_input.txt

This file contains the grid. The data in the file are organized as follow:

83	Maximum number of pixels in x dimension
83	Maximum number of pixels in y dimension
17	Maximum number of pixels s in z dimension
111537	Actual number of pixels in total
0	Coordinate x of first point
0	Coordinate y of first point
0	Coordinate z of first point
0	Coordinate x of second point
0	Coordinate y of second point
1	Coordinate z of second point

…	Etc.

The above structure has 81*81*17=111537 pixels. The coordinate goes from 0 to 80 and 0 to 16. No matter the geometry, the coordinate should start from the origin (0, 0, 0). Regardless of actual dimension, the coordinate should always be integer. 83, 83 and 17 must be bigger than the actual size of the grid in each dimension. These numbers represent the simulation space, but actual dipoles are built only for 111537 pixels.

pathPara=.\\sample\\para_DDA_input.txt

The material distribution that represents the actual structure input into the model. The size of the array is 334611=3*111537. Every 3 numbers represent the label of the material for the pixel whose coordinate is in grid_DDA_input.txt at the same position. If the material is isotropic, the 3 numbers for 1 pixel must be the same. As an example, for the first point in grid file, the coordinate is (0,0,0). If (1,1,1) is the first one in the para file, it means at point (0,0,0), the material is the one with label of 1.

The material label can be between 0 and 1 which means the material has a permittivity in between. (Eq. 9)

[Material]

material1=Air

The material with label 0, which is the background material.

material2=SiO2

The material with label 1, which is the material used for the structure if topology optimization is carried out.

[Grid]

d=25

Distance between neighboring dipoles. The unit is nm. The 81*81*17=111537 can be regarded to represent the actual space of the size of 2um*2um*400nm.

[Input field]

Currently, only linear polarized plane wave is supported to be as the background field. The amplitude of the input field is default as 1.0V/m so the output field distribution can always be regarded as local field enhancement.

lam=500

Unit: nm. The wavelength of the input wave.

nKx=0.0

x component of the unit vector of the wavenumber.

nKy=0.0

y component of the unit vector of the wavenumber.

nKz=1.0

z component of the unit vector of the wavenumber.

nEx=1.0

x component of the unit vector of polarization.

nEy=0.0

y component of the unit vector of polarization.

nEz=0.0

z component of the unit vector of polarization.

[DDA iteration]

MAX_ITERATION_DDA=10000

The maximum number of iterations the iterative solver goes to solver the equation before it is cut off even if the error is not smaller than the MAX_ERROR defined below.

MAX_ERROR=0.00001

The threshold that the iterative solver stops automatically when it is reached. Generally, smaller the MAX_ERROR value, more accurate the simulation, and more iterations needed. MAX_ERROR must be smaller than 1 because initially it is 1.

[Output]

saveDir=.\\sample

Where the folders for output data are located. The folders in the 'sample' directory are required for data being correctly stored and post-processed.

[Plot]

This is the part where instructions are given to the python script 'plot.py' as plotting options.

plot=True

True will enable the plot. False will disable the plot.

xPlot=True

Enable a 2D yz cross section plot for electric field distribution for the x coordinate given. True and False serve as switch

yPlot=True

Enable a 2D zx cross section plot for electric field distribution for the y coordinate given. True and False serve as switch

zPlot=True

Enable a 2D xy cross section plot for electric field distribution for the z coordinate given. True and False serve as switch

shapePlot=True

A plot of the geometry with both the structure and the background. True and False serve as switch.

shapeSolidPlot=True

A plot of the geometry with only the structure. True and False serve as switch.

x=10

x coordinate for the xPlot. x is unitless that represents the grid coordinate in the grid file.

y=10

y coordinate for the yPlot. y is unitless that represents the grid coordinate in the grid file.

z=20

z coordinate for the zPlot. z is unitless that represents the grid coordinate in the grid file.

ELimit=True

True will enable manual control of the maximum and minimum value for the legend and color of the electric field distribution plot. False vice versa.

ELimitLow=0.0

The minimum value for electric field distribution is 0.

ELimitHigh=3.0

The maximum value for electric field distribution is 3.

itStart=0

The start iteration for the plots. This value is always 0 for 'DDA input' since it only calculates one model.

itEnd=0

The end iteration for the plots. This value is always 0 for 'DDA input' since it only calculates one model.

colorMax=1

Maximum number of parameter for shapePlot and shapeSolidPlot. 1 if there are 2 materials. 2 if there are 3. Etc.

shapeBarrier=0.5

When plotting shapeSolidPlot, the threshold for the parameter to be regarded as background and not plotted. In this case, any material with parameter smaller than 0.5 will not be plotted.

plotDpi=100

The resolution of the plot. Smaller number decrease the resolution but makes the plotting process faster.

2.2. EvoOpt 2D input

This model topology optimizes the structure starting from an input initial geometry. The input geometry and the final results must be 2D extruded geometries, which means pixels have different parameters in x and y dimension but stays the same along the z direction. The input geometry (grid) is always cubic, but the material distribution can vary as needed.

The instructions not explained are the same as previous models.

[Geometry]

pathCommonData=.\\sample\\ grid_Evo_input.txt

pathPara=.\\sample\\para_fullslab.txt

[Material]

material1=Air

material2=SiO2

[Grid]

d=25

[Input field]

lam=500

nKx=0.0

nKy=0.0

nKz=1.0

nEx=1.0

nEy=0.0

nEz=0.0

[DDA iteration]

MAX_ITERATION_DDA=10000

MAX_ERROR=0.00001

[Evo Option]

MAX_ITERATION_EVO=200

The maximum number of iterations for topology optimization. In each iteration, the previous structure is simulated and local gradient is calculated. Then the local gradient is applied to the previous structure to generate a new structure.

epsilon=100

Fixed step that influences the optimization process. This step is applied to the calculated local gradient to update the previous parameters. Bigger step accelerates the optimization process but makes it easier for the algorithm to miss the local gradient.

[Filter Option]

For 2D extruded structure, filter can be applied to the parameters to control the robustness of the design. This option is used to get rid of fine structures and intermediate parameters and to generate more fabricable design.

filter=True

True will enable filter. False vice versa.

filterInfo=0,1.0,/100,2.0,/

0,1.0,/ means starting from iteration 0, a filter with radius of 1 pixel is applied. 100,2.0,/ means starting from iteration 100, a filter with radius of 2 pixel is applied. For each pixel, parameters of neighboring pixels inside the circle with the radius defined will be averaged to calculate the filtered parameter value. This can effectively control the minimum size of the fine structures in the final design.

For detail definition, please check equation 8 and equation 9 in [6].

betaType=piecewise

During the topology optimization process, intermediate materials with parameter between 0 and 1, which are physically unrealistic, are generated. These unwanted features should be eliminated because they can not be fabricated. Thus, a smooth function is needed to binarize the geometry. Please check equation 10 in [6].

β is the value that controls the sharpness of the smooth function. The smooth function is more close to a step function when β is bigger. (>50 generates an almost complete step function)

β can have different value at different iteration to control the extent of binarization throughout the design process, thus different functions can be defined.

These functions are defined in 'tools.cpp' and imported in 'FilterOption::update_beta()'.

betaType=piecewise: A piecewise function. For detail please check 'piecewise_update()' in 'tools.cpp'

betaType=exp: An exponential function. For detail please check 'exp_update ()' in 'tools.cpp'

betaType=linear: A linear function. For detail please check 'linear_update ()' in 'tools.cpp'

betaMin=0.0

Minimum β value.

betaMax=50.0

Maximum β value.

ita=0.5

Threshold for the smooth function. If the smooth function is a step function. Parameters bigger than ita will be 1. Parameters smaller than ita will be 0.

[Symmetry Option]

Symmetry condition can be applied to the optimization process. In DDA, no symmetry boundary condition can be applied so this option does not decrease the simulation space as in FEM. However, it can decrease the number of parameters. For example, a 4fold symmetric optimization has ¼ parameters compare to a non-symmetric optimization.

hasSym=False

True will enable symmetry. False vice versa.

symmetry=4fold

'4fold' means 4fold symmetry.

symAxis=9.5/9.5/

x and y coordinate for the symmetry axis.

[Obj Option]

objName=PointE

Name of the object function. Please check 4.1. for detail.

objPara=1000.0/1000.0/450.0/

Parameters of the object function. Please check 4.1. for detail.

[Output]

saveDir=.\\sample

[Plot]

plot=True

xPlot=False

yPlot=False

zPlot=True

shapePlot=True

shapeSolidPlot=False

x=40

y=40

z=15

ELimit=False

ELimitLow=0.0

ELimitHigh=3.0

itStart=151

itEnd=200

colorMax=1

shapeBarrier=0.5

plotDpi=100

3. Data Processing

3.1. Preprocessing

3.1.1. Input Material

All the dielectric functions are located in 'Topo-DDA-forWin64\\diel\\'. The real part is stored in file ending with 'Re_eps'. The imaginary part is stored in file ending with 'Im_eps'. If new material needs to be imported, name must be add into the dictionary in 'complex<double> Get_material()' in 'tools.cpp'.

Data in the first column of the file are the wavelengths in 'm', data in the second column are the corresponding permittivity.

3.1.1. Input Geometry from COMSOL

Please check https://github.com/Raznov/Generate-geometry-for-Topo-DDA-from-COMSOL.

3.2. Postprocessing

3.2.1. Output Geometry to COMSOL/Lumerical

Please check https://github.com/Raznov/Output-geometry-of-Topo-DDA-to-COMSOL-Lumerical

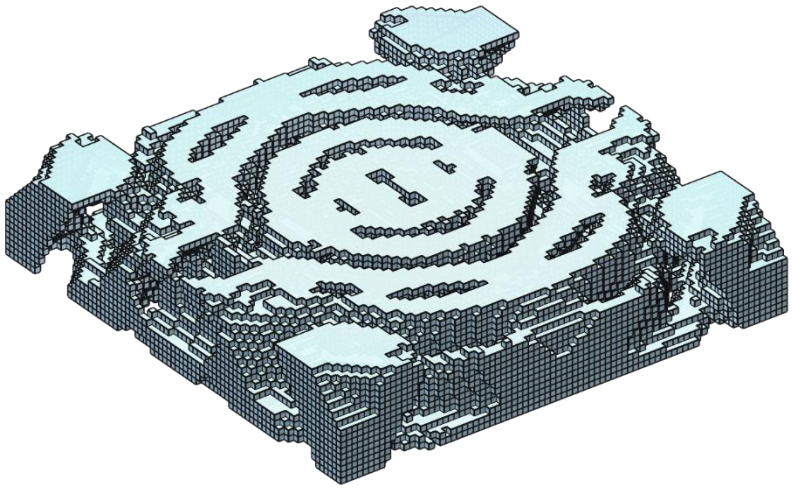
Lumerical FDTD:
Step 1: Run matlab script "genetrate_geo_for_lumerical" with the correct directory and file name.

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. The input file, however, is the previous file generated for Lumerical geometry input.

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 geometry plotted with COMSOL:



4. Data Structures

4.1. Object Functions

Object function is needed for the optimization process. The algorithm will try to optimize structures so as to maximize the object function value. Following object functions are currently supported:

objName=PointE:

This object function is the electric field amplitude at a designated coordinate. This coordinate does not necessarily need to be inside the simulation space, which means it can be a point far away from the structure. The input parameters are ‘x coordinate/y coordinate/z coordinate/’.

Reference

(1) B. T. Draine. THE DISCRETE-DIPOLE APPROXIMATION AND ITS APPLICATION TO INTERSTELLAR GRAPHITE GRAINS. *Astrophys. J.* **1988**.

(2) Draine, B. T.; Flatau, P. J. Discrete-Dipole Approximation For Scattering Calculations. *J. Opt. Soc. Am. A* **1994**, *11* (4), 1491. https://doi.org/10.1364/josaa.11.001491.

(3) Flatau, P. J.; Draine, B. T. Fast near Field Calculations in the Discrete Dipole Approximation for Regular Rectilinear Grids. *Opt. Express* **2012**. https://doi.org/10.1364/oe.20.001247.

(4) Goodman, J. J.; Draine, B. T. Application of Fast-Fourier-Transform Techniques to the Discrete-Dipole Approximation Flexible Technique for Calculating Scattering and { k [,, W. *Opt. Lett.* **1991**, *16* (15), 1198–1200.

(5) Yurkin, M. A.; Min, M.; Hoekstra, A. G. Application of the Discrete Dipole Approximation to Very Large Refractive Indices: Filtered Coupled Dipoles Revived. *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.* **2010**, *82* (3), 1–12. https://doi.org/10.1103/PhysRevE.82.036703.

(6) Wang, F.; Jensen, J. S.; Sigmund, O. Robust Topology Optimization of Photonic Crystal Waveguides with Tailored Dispersion Properties. *J. Opt. Soc. Am. B* **2011**, *28* (3), 387. https://doi.org/10.1364/josab.28.000387.