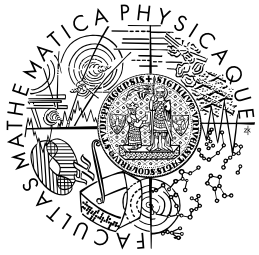


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Ondřej Novák

Numerical simulations of optical response of nanostructures using FDTD method

Institute of Physics of Charles University

Supervisor of the bachelor thesis: RNDr. Martin Veis, Ph.D.

Study programme: Physics

Study branch: General physics

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

My cordial gratitude belongs to my supervisor RNDr. Martin Veisov, Ph.D for his willingness to share his knowledge, for his guidance and priceless advice.

I would also like to heartily thank my family and my loved ones for their support.

Title: Numerical simulations of optical response of nanostructures using FDTD method

Author: Ondřej Novák

Institute: Institute of Physics of Charles University

Supervisor: RNDr. Martin Veis, Ph.D., Institute of Physics of Charles University

Abstract: Abstract: The aim of this thesis is to develope an efficient algorythm to compute optical response of nanostructures and to equip it with usefull tools for further data processing. Considered problem is reduced to two dimensions and the method used is the Finite Difference Time Domain (FDTD). This method operates on finite grid called Yee grid and is often called Yee algorithm. En extra emphasis is given on optimalization of the algorithm and writing the computer code efficiently. Evolution equations are written in tensor form and the core algorithm is moved to graphic card using CUDA. Various boundary conditions are introduced to reduce reflections on the edge of the grid. Representation of a real object on the Yee-grid is discussed with introduction of several smoothing methods to improve the shape convergence of simulated object. Useful post-processing methods are introduced - discrete fourier transform, from which the frequency response of simulated object can be computed and a way to compute the far field from the near field. Finaly, there is an attempt to simulate a surface plasmon.

Keywords: FDTD, CUDA, PML, CPML, Yee-grid

Contents

Introduction **2**

1 FDTD Method **4**
1.1 Maxwell's equations in 2D for FDTD 4
1.2 Finite difference derivatives approximation 6
1.3 Updating equations 7

2 Boundary conditions **12**
2.1 Perfectly matched layer 12
2.2 Complex Frequency-shifted PML 16
2.2.1 CPML parameters 19
2.3 Periodic boundary condition 20

3 Writing the code in Python **21**
3.1 Choosing Python as a programming language 22
3.2 Update equation in optimal form 22
3.3 Parallelization 25
3.4 Computation speed 26

4 Preparing computation input **27**
4.1 Grid parameters and resolution 27
4.2 Building an object on the Yee-grid 28
4.3 Staircase smoothing 29
4.4 Source generation 31

5 Post processing **34**
5.1 Fourier Transform 34
5.2 Near-field to Far-field Transformation 35
5.2.1 Surface Equivalence Theorem 36
5.2.2 Surface Currents 36
5.2.3 Far-field expression 37

6 Surface plasmons **38**
6.1 Dielectric model of noble metals 38
6.1.1 Drude–Sommerfeld theory 38
6.1.2 Interband transitions 39
6.2 Simulation of plasmon 39

Conclusion **42**

Bibliography **43**

List of Figures **44**

List of Abbreviations **45**

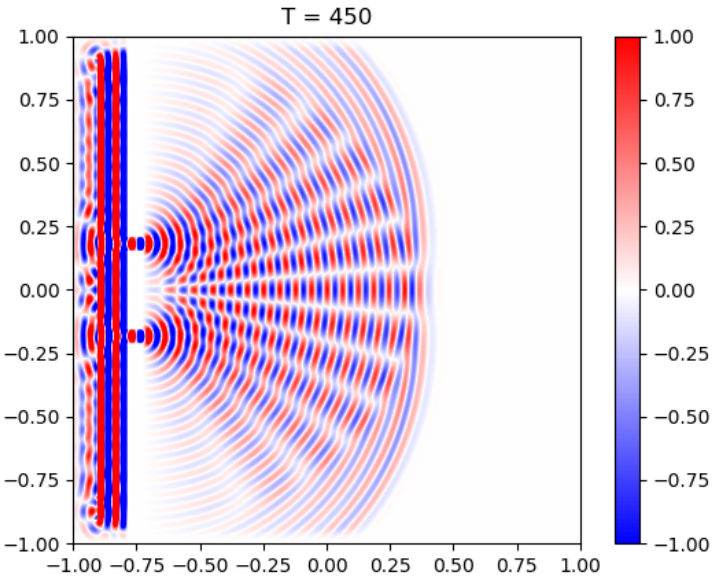
Introduction

From time immemorial, people aimed to understand physical phenomena so they could exploit it to they'r own benefit and further advance. To do so, physics was slowly developed over the millennia, the sophisticated tool to predict the future. As the understanding of nature prolonged, the complexity of our equation grew. Nowadays, analytic solutions of current problems appears only rarely. Fortunately, we live in an era of silicon, material, that is responsible for probably the greatest advance of human spieces so far. The computing power of modern machines opened a window to new branch of mathematics, the numerics.

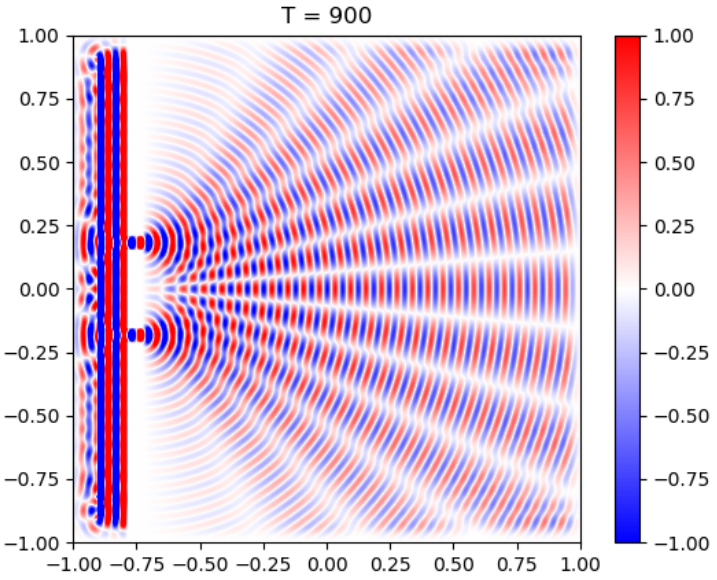
The interest of this thesis is given to *Maxwell's equations*, which is a system of four partial differential vector equations. Even tho, there are some anylytic solutions to simple problems, in most of the engineering problems, the numerical simulation is inevitable. There are several approaches in this field: Transmit matrix, for solving the thin layers problem, Finite element method, for field problem, etc.

The method, described in this thesis, is called the Finite difference time domain method, (FDTD) for short. This method allows user to solve an evolution problem. In typical situation, user models a device on discrete grid, called *Yee grid* and defines the outer field. The algorithm compute the value of the field on every single point on grid in each time step required. The advantage of using this approach is, that it is simple - Maxwell's equations are broken into evolution equations, using the difference approximation, it is universal - It can be used in many other problems, such as heat propagation, Schrodinger equation etc. It is intuitive, because it solves Maxwell's equations directly, while using only difference approximation. It is a great learning tool for electrodynamics, because the output of the simulation can be converted into a video of the field. It is scalable, the time required, while extending the calculation, grows linearly, while with similar methods, it grows exponentially. It is easily parallelizable. It can be simply modified to handle nonlinear behaviour. It is a well developed algorythm and so it become robust and accurate. The downside of this algorithm is, that it needs some additional layers around it's problem space to be incorporated. It provides only the information about field, so that other post processing methods has to be used to extract desired information, such as frequency response. It's grid does not efficiently represents curved surfaces and it is very inefficient for highly resonant devices.

The aim of this thesis is to explain the mechanism behind FDTD algorithm, to develop code in two dimensions with emphasis on efficiency, to provide solution for some of the flaws mentioned above, namely Perfectly matching layer, curved surfaces, spectrum extraction and to provide some understanding about limits of this computation method.



(a) Monochromatic plane wave hitting the two slide barrier after 450 time steps.



(b) Monochromatic plane wave hitting the two slide barrier after 900 time steps.

Figure 1: An axample of the FDTD simulation.

1. FDTD Method

FDTD stands for *Finite-difference time-domain*. It is a robust numerical technique, used to compute approximate solutions of *Maxwell's equations*, using the finite difference in spatial steps and time steps. This method is also called *Yee's method* after Chinese American applied mathematician *Kane S. Yee*.

1.1 Maxwell's equations in 2D for FDTD

In order to write the FDTD algorythm, lets have a look at general *Maxwell's equations*:

$$\begin{aligned} \operatorname{rot} \boldsymbol{H} - \partial_t \boldsymbol{D} &= \boldsymbol{j} \\ \operatorname{div} \boldsymbol{D} &= \rho_e \\ \operatorname{rot} \boldsymbol{E} + \partial_t \boldsymbol{B} &= -\boldsymbol{M} \\ \operatorname{div} \boldsymbol{B} &= \rho_m \end{aligned} \tag{1.1}$$

Here $\boldsymbol{H}$ stands for the magnetic field strength vector, $\boldsymbol{D}$ is the electric displacement vector, $\boldsymbol{j}$ is the electric current density vector, ρ_e is the electric charge density, $\boldsymbol{E}$ is the electric field strength vector, $\boldsymbol{B}$ is the magnetic flux density vector, $\boldsymbol{M}$ is the magnetic current density vector and finally ρ_m is the magnetic charge density. We chose this form of *Maxwell's equations*, with $\boldsymbol{M}$ and ρ_m, because even tho these quantities does not represent anything in a real world, their presence in *Maxwell's equations* can simplify some engineering problems.

Maxwell's equations alone can not fully describe the Electromagnetism in materials, beacue it is a set of 8 equations for 12 variables we need to add *Constitutive relations* first. This means, that we can think about *Maxwell's equations* as about equations that describes fields production mechanism, and *Constitutive relations*:

$$\boldsymbol{D}(t) = \varepsilon(t) * \boldsymbol{E}(t), \tag{1.2}$$

$$\boldsymbol{B}(t) = \mu(t) * \boldsymbol{H}(t), \tag{1.3}$$

as about equations, that describes, how fields interact with materials. Note, that $\varepsilon(t)$ and $\mu(t)$ are time-dependant tensors, but that is only the most general case. In this thesis, we will focus primarily on linear materials constant in time.

Expanding curl *Maxwell's equations*, we obtain:

$$\begin{aligned} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \frac{\partial D_x}{\partial t} &= j_x \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \frac{\partial D_y}{\partial t} &= j_y \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \frac{\partial D_z}{\partial t} &= j_z \end{aligned} \tag{1.4}$$

$$\begin{aligned}
\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} + \frac{\partial B_x}{\partial t} &= -M_x \\
\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} + \frac{\partial B_y}{\partial t} &= -M_y \\
\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} + \frac{\partial B_z}{\partial t} &= -M_z
\end{aligned} \tag{1.5}$$

In this thesis, we focun only on problems, that are homogenou in one dimension. This means, that fields does not change in this dmension and so we are left with two-dimensional problem. Without loss of generality, we can assume this direction to be the Z direction. Hence, we can substitute $\frac{\partial}{\partial z} = 0$ to sets of equations above, receiving the following:

$$\begin{aligned}
\frac{\partial H_z}{\partial y} - \frac{\partial D_x}{\partial t} &= j_x \\
-\frac{\partial H_z}{\partial x} - \frac{\partial D_y}{\partial t} &= j_y \\
\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \frac{\partial D_z}{\partial t} &= j_z
\end{aligned} \tag{1.6}$$

$$\begin{aligned}
\frac{\partial E_z}{\partial y} + \frac{\partial B_x}{\partial t} &= -M_x \\
-\frac{\partial E_z}{\partial x} + \frac{\partial B_y}{\partial t} &= -M_y \\
\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} + \frac{\partial B_z}{\partial t} &= -M_z
\end{aligned} \tag{1.7}$$

Having a closer look at these equations, we can see, that these equations can be split into two separate sets, called TM and TE modes respectively:

$$\begin{aligned}
\frac{\partial D_z}{\partial t} &= \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - j_z \\
\frac{\partial B_x}{\partial t} &= -\frac{\partial E_z}{\partial y} - M_x \\
\frac{\partial B_y}{\partial t} &= \frac{\partial E_z}{\partial x} - M_y
\end{aligned} \tag{1.8}$$

$$\begin{aligned}
\frac{\partial B_z}{\partial t} &= \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - M_z \\
\frac{\partial D_x}{\partial t} &= \frac{\partial H_z}{\partial y} - j_x \\
\frac{\partial D_y}{\partial t} &= -\frac{\partial H_z}{\partial x} - j_y
\end{aligned} \tag{1.9}$$

Since these sets of equations are decoupled and does not contain any common terms, we can focus on deriving updating equations from single mode, TM in our case.

Another step is to add some form of *Constitutive relations* (1.2, 1.3) to our TM mode equations (1.8). We start with the simple case, where *Constitutive relations* can be described by diagonal tensor:

$$\varepsilon = \begin{bmatrix} \varepsilon_x & 0 & 0 \\ 0 & \varepsilon_y & 0 \\ 0 & 0 & \varepsilon_z \end{bmatrix} \tag{1.10}$$

$$\mu = \begin{bmatrix} \mu_x & 0 & 0 \\ 0 & \mu_y & 0 \\ 0 & 0 & \mu_z \end{bmatrix} \tag{1.11}$$

This way we can replace $\boldsymbol{B}$ with $\boldsymbol{H}$ and $\boldsymbol{D}$ with $\boldsymbol{E}$. Terms $\boldsymbol{j}$ and $\boldsymbol{M}$ can be written as a sum of conduction current density and impressed current density The conduction current density can be expresed by using the *Ohm's law*:

$$\begin{aligned} \boldsymbol{j} &= \boldsymbol{j}_c + \boldsymbol{j}_i = \sigma^e \boldsymbol{E} + \boldsymbol{j}_i \\ \boldsymbol{M} &= \boldsymbol{M}_c + \boldsymbol{M}_i = \sigma^m \boldsymbol{H} + \boldsymbol{M}_i \end{aligned} \tag{1.12}$$

Where σ^e (σ^m) is electric (magnetic) conductivity. Once again we expect them in diagonal form:

$$\sigma^e = \begin{bmatrix} \sigma^e_x & 0 & 0 \\ 0 & \sigma^e_y & 0 \\ 0 & 0 & \sigma^e_z \end{bmatrix} \tag{1.13}$$

$$\sigma^m = \begin{bmatrix} \sigma^m_x & 0 & 0 \\ 0 & \sigma^m_y & 0 \\ 0 & 0 & \sigma^m_z \end{bmatrix} \tag{1.14}$$

These substitutions leaves us with following set of equations:

$$\begin{aligned} \frac{\partial E_z}{\partial t} &= \frac{1}{\varepsilon_z}\Big(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma^e_z E_z - j_{iz}\Big) \\ \frac{\partial H_x}{\partial t} &= \frac{1}{\mu_x}\Big(-\frac{\partial E_z}{\partial y} - \sigma^m_x H_x - M_{ix}\Big) \\ \frac{\partial H_y}{\partial t} &= \frac{1}{\mu_y}\Big(\frac{\partial E_z}{\partial x} - \sigma^m_y H_y - M_{iy}\Big) \end{aligned} \tag{1.15}$$

In the next step, we are going to move to discrete world of *Yee's grid*. In order to do so, we have to substitute partial derivatives with discrete differences.

1.2 Finite difference derivatives approximation

In FDTD we take a continuous function, sample it by defined sampling rate and then replace our continuous operators with discrete operator. Choosing the right sampling rate is a question of stability and will be discussed later. The essential step is choosing the right discrete operators.

Since we are using differential form of Maxwell's equations, the most often used operator is derivative (difference in discrete case). When we have a look at the definition of derivation:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{1.16}$$

Having a Δx small, but finite, the difference approximation, made from this definition would be a forward difference:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{1.17}$$

This is usable, but to reduce numerical error, we can find even better approximation, the central difference:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x}. \tag{1.18}$$

The error caused by this approximation is discussed in [1].

In this case, the function value in position, where the difference is calculated, is not actually used, but points around are used instead. Also the interval Δx is double, which is actually not an issue, we will be able to take an advantage of it later.

We could even use aproximations of higher order, which uses more points to calculate the difference, but that would rapidly increase the computation time and the benefit is not noticeable. Using smaller spatial and time steps is more beneficial.

1.3 Updating equations

Acording to [1], page 14, taking the *Central difference* (1.18) and replacing both spatial and time derivatives in (1.15), we obtain a discrete form of update equations. Filed components are now sampled at discrete position in time and space and forms an orthogonal grid formed of rectangular cells called *Yee cells*. Having a closer look at the position of field components in *Yee cell*, one can realize, that electric field vector components are placed at the centers of the edges of the *Yee cells* and are oriented parallel to these edges. Magnetic field vector components however are place at the centers of the sides of these cells and are parallel to it's normals:

$$\begin{aligned} E_z(i,\, j) &\Rightarrow \Big[i\Delta x,\, j\Delta y\Big] \\ H_x(i,\, j) &\Rightarrow \Big[i\Delta x,\, (j + 0.5)\Delta y\Big] \\ H_y(i,\, j) &\Rightarrow \Big[(i + 0.5)\Delta x,\, j\Delta y\Big]. \end{aligned} \tag{1.19}$$

If we divide the *Yee cell* into 8 sub-cells with each side half length of the original, we end up with cells that have some field vector components on its vertices, but only half of these vertices will be filled. This allows us to reduce the memory usage of future code and use only half of the double step in (1.18) approximation. With this step, we have to keep in mind, that now both electric

and magnetic field lives on its own grid that is shifted to the other one by half of the cell length in all directions. It is a thing that we are going to have to address later when building an object in *Yee grid*.

From now, time (step) index is going to be written as an upper index so it is easier distinguished from spatial indexes.

$$\begin{aligned}\frac{E_z^{n+1}(i,\,j)-E_z^{n}(i,\,j)}{\Delta t} = &\frac{1}{\varepsilon_z(i,\,j)}\frac{H_y^{n+\frac{1}{2}}(i,\,j)-H_y^{n+\frac{1}{2}}(i-1,\,j)}{\Delta x}\\ &-\frac{1}{\varepsilon_z(i,\,j)}\frac{H_x^{n+\frac{1}{2}}(i,\,j)-H_x^{n+\frac{1}{2}}(i,\,j-1)}{\Delta y}\\ &-\frac{\sigma_z^{e}(i,\,j)}{\varepsilon_z(i,\,j)}E_z^{n+\frac{1}{2}}(i,\,j)-\frac{j_{iz}^{n+\frac{1}{2}}(i,\,j)}{\varepsilon_z(i,\,j)}\end{aligned} \tag{1.20}$$

$$\begin{aligned}\frac{H_x^{n+\frac{1}{2}}(i,\,j)-H_x^{n-\frac{1}{2}}(i,\,j)}{\Delta t} = &-\frac{1}{\mu_x(i,\,j)}\frac{E_z^{n}(i,\,j+1)-E_z^{n}(i,\,j)}{\Delta y}\\ &-\frac{\sigma_x^{m}(i,\,j)}{\mu_x(i,\,j)}H_x^{n}(i,\,j)-\frac{M_{ix}^{n}(i,\,j)}{\mu_x(i,\,j)}\end{aligned} \tag{1.21}$$

$$\begin{aligned}\frac{H_y^{n+\frac{1}{2}}(i,\,j)-H_y^{n-\frac{1}{2}}(i,\,j)}{\Delta t} = &\frac{1}{\mu_y(i,\,j)}\frac{E_z^{n}(i+1,\,j)-E_z^{n}(i,\,j)}{\Delta x}\\ &-\frac{\sigma_y^{m}(i,\,j)}{\mu_y(i,\,j)}H_y^{n}(i,\,j)-\frac{M_{iy}^{n}(i,\,j)}{\mu_y(i,\,j)}\end{aligned} \tag{1.22}$$

Since we want to write a computer code, dealing with half size indexes would not be practical and so we also replaced interval $(n-\frac{1}{2},\,n+\frac{1}{2})$ with interval $(0,\,1)$ so the length stay the same and the spatial indexes become natural numbers.

The problem is that now we have Electric field vector component with half size step index and full size step index. To fix this, assuming the field vector component change in between time steps is small are going to approximate the half step size index with full step size index:

$$E_z^{n+\frac{1}{2}} \approx \frac{E_z^{n}+E_z^{n+1}}{2} \tag{1.23}$$

The same can be done for magnetic field vector component, but instead of changing half size step index with full size step index, we change them in the other way:

$$\begin{aligned}H_x^{n} &\approx \frac{H_x^{n+\frac{1}{2}}+H_x^{n-\frac{1}{2}}}{2}\\ H_y^{n} &\approx \frac{H_y^{n+\frac{1}{2}}+H_y^{n-\frac{1}{2}}}{2}\end{aligned} \tag{1.24}$$

Substituting (1.23) and (1.24) into (1.20), (1.21) and (1.22) and rearranging terms so that the future time steps are on the left hand side of the equations, while previous are on the right hand side, we recive following:

$$\begin{aligned} E_z^{n+1}(i,\,j)\Big(\frac{1}{\Delta t}+\frac{\sigma_z^e(i,\,j)}{2\varepsilon_z(i,\,j)}\Big) = \Big(\frac{1}{\Delta t}-\frac{\sigma_z^e(i,\,j)}{2\varepsilon_z(i,\,j)}\Big)E_z^n(i,\,j) \\ + \frac{1}{\varepsilon_z(i,\,j)}\frac{H_y^{n+\frac{1}{2}}(i,\,j)-H_y^{n+\frac{1}{2}}(i-1,\,j)}{\Delta x} \\ - \frac{1}{\varepsilon_z(i,\,j)}\frac{H_x^{n+\frac{1}{2}}(i,\,j)-H_x^{n+\frac{1}{2}}(i,\,j-1)}{\Delta y} \\ - \frac{j_{iz}^{n+\frac{1}{2}}(i,\,j)}{\varepsilon_z(i,\,j)} \end{aligned} \tag{1.25}$$

$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,\,j)\Big(\frac{1}{\Delta t}+\frac{\sigma_x^m(i,\,j)}{2\mu_x(i,\,j)}\Big) = \Big(\frac{1}{\Delta t}-\frac{\sigma_x^m(i,\,j)}{2\mu_x(i,\,j)}\Big)H_x^{n-\frac{1}{2}}(i,\,j) \\ - \frac{1}{\mu_x(i,\,j)}\frac{E_z^n(i,\,j+1)-E_z^n(i,\,j)}{\Delta y} \\ - \frac{M_{ix}^n(i,\,j)}{\mu_x(i,\,j)} \end{aligned} \tag{1.26}$$

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i,\,j)\Big(\frac{1}{\Delta t}+\frac{\sigma_y^m(i,\,j)}{2\mu_y(i,\,j)}\Big) = \Big(\frac{1}{\Delta t}-\frac{\sigma_y^m(i,\,j)}{2\mu_y(i,\,j)}\Big)H_y^{n-\frac{1}{2}}(i,\,j) \\ + \frac{1}{\mu_y(i,\,j)}\frac{E_z^n(i+1,\,j)-E_z^n(i,\,j)}{\Delta x} \\ - \frac{M_{iy}^n(i,\,j)}{\mu_y(i,\,j)} \end{aligned} \tag{1.27}$$

Now we are almost done with evolution equations. In last few steps we will add a relationship between spatial and time steps and evaluate the $n+1$ or $(n+\frac{1}{2}$ vector component of the electric (magnetic) field, using the previous time steps.

The relation between time and space step we choose is so that it takes at least two time steps to propagate trought one spatial cell:

$$\begin{aligned} \Delta x &= 2\,c_0\Delta t, \\ \Delta y &= \frac{2}{k}c_0\Delta t, \end{aligned} \tag{1.28}$$

where c_0 is the speed of light in vacuum and k is the ratio between Δx and Δy:

$$c_0 = \frac{1}{\sqrt{\varepsilon_0\mu_0}}, \tag{1.29}$$

$$\Delta x = k\Delta y \tag{1.30}$$

we obtain evolution equations:

$$\begin{aligned} E_z^{n+1}(i,\,j) &= \frac{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 - \sigma_z^e(i,\,j)\Delta x}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} E_z^n(i,\,j) \\ &+ \frac{2}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x}\Big[H_y^{n+\frac{1}{2}}(i,\,j) - H_y^{n+\frac{1}{2}}(i-1,\,j)\Big] \\ &- \frac{2k}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x}\Big[H_x^{n+\frac{1}{2}}(i,\,j) - H_x^{n+\frac{1}{2}}(i,\,j-1)\Big] \\ &- \frac{2\Delta x}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} j_{iz}^{n+\frac{1}{2}}(i,\,j) \end{aligned} \tag{1.31}$$

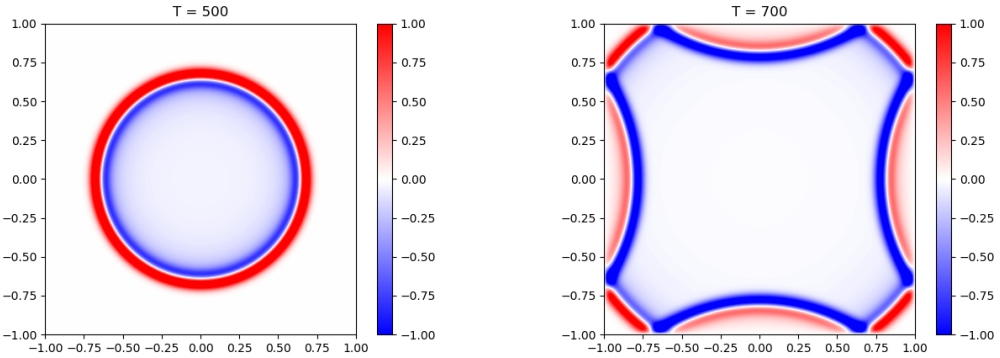
$$\begin{aligned} H_x^{n+\frac{1}{2}}(i,\,j) &= \frac{4c_0\mu_0\mu_{rx}(i,\,j) - \Delta x\sigma_x^m(i,\,j)}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)} H_x^{n-\frac{1}{2}}(i,\,j) \\ &- \frac{2k}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)}\Big[\tilde{E}_z^n(i,\,j+1) - \tilde{E}_z^n(i,\,j)\Big] \\ &- \frac{2\Delta x}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)} M_{ix}^n(i,\,j) \end{aligned} \tag{1.32}$$

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i,\,j) &= \frac{4c_0\mu_0\mu_{ry}(i,\,j) - \Delta x\sigma_y^m(i,\,j)}{4c_0\mu_0\mu_r(i,\,j) + \Delta x\sigma_y^m(i,\,j)} H_y^{n-\frac{1}{2}}(i,\,j) \\ &+ \frac{2}{4c_0\mu_0\mu_{ry}(i,\,j) + \Delta x\sigma_y^m(i,\,j)}\Big[\tilde{E}_z^n(i+1,\,j) - \tilde{E}_z^n(i,\,j)\Big] \\ &- \frac{2\Delta x}{4c_0\mu_0\mu_{ry}(,\,j) + \Delta x\sigma_y^m(i,\,j)} M_{iy}^n(i,\,j) \end{aligned} \tag{1.33}$$

The last operation with these equations, we substitute time-independatn factors with constants:

$$\begin{aligned} \tilde{E}_z^{n+1}(i,\,j) &= C_{ezez}(i,\,j)\tilde{E}_z^n(i,\,j) \\ &+ C_{ezhy}(i,\,j)\Big[H_y^{n+\frac{1}{2}}(i,\,j) - H_y^{n+\frac{1}{2}}(i-1,\,j)\Big] \\ &- C_{ezhx}(i,\,j)\Big[H_x^{n+\frac{1}{2}}(i,\,j) - H_x^{n+\frac{1}{2}}(i,\,j-1)\Big] \\ &- C_{ezjiz}(i,\,j) j_{iz}^{n+\frac{1}{2}}(i,\,j) \\ H_x^{n+\frac{1}{2}}(i,\,j) &= C_{hxhx}(i,\,j)H_x^{n-\frac{1}{2}}(i,\,j) \\ &- C_{hxez}(i,\,j)\Big[\tilde{E}_z^n(i,\,j+1) - \tilde{E}_z^n(i,\,j)\Big] \\ &- C_{hxmix}(i,\,j)M_{ix}^n(i,\,j) \\ H_y^{n+\frac{1}{2}}(i,\,j) &= C_{hyhy}(i,\,j)H_y^{n-\frac{1}{2}}(i,\,j) \\ &+ C_{hyez}(i,\,j)\Big[\tilde{E}_z^n(i+1,\,j) - \tilde{E}_z^n(i,\,j)\Big] \\ &- C_{hymiy}(i,\,j)M_{iy}^n(i,\,j) \end{aligned} \tag{1.34}$$

As we can see, some of these coefficients differ only by factor k, which means that in case, we are using equidistant grid in both directions ($k = 1$), we can



(a) Gaussian pulse evolution at 500th time step

(b) Gaussian pulse evolution at 700th time step

Figure 1.1: Demonstration of reflection on boundary without using Absorbing boundary condition. The source is a gaussian pulse, centered in the middle of the grid. The grid represents a free space.

use fewer coefficients and save computer memory. This may not seem significant in two dimensions, but these small facts quickly add up and moving into three dimensions would become realy problematic.

Let's briefly summarize this chapter. We took differential *Maxwell's equations*, expanded by magnetic current density and magnetic charge density and had detailed look at curl equations. We substituted *Constitutive relations* into these equations and rewrote these vector equations into vector component equations. Assuming that our problem is homogeneous in z direction, we eliminated partial derivatives along the z axis. We then separated these equations into two independent modes, TM and TE. Simplifying our case to have only time-independent linear material property, described by diagonal tensor permittivity and permeability, we obtained our final analytical equations (1.15).

We then moved into discrete *Yee's grid* where we approximated partial derivatives with central differences, which is a second order error approximation.

In the last section of this chapter, we used numerous substitutions and grid shifts, to simplify these equations for computer code as much as possible. We ended up with formula to compute field strength vector component only from previous time steps field strength vector components.

Problem with these formulas is that our grid size is finite and cells at the edges of our grid has one neighbor missing and so the field, lying in them, can not be properly computed. This means that wave, propagating towards the problem boundary, would reflect back to our problem space and cause huge error. This phenomena is demonstrated on figure 1.1. The profile of used gaussian pulse is $200\,exp(-(t-100)^2/500)$, which reaches it's maximum at 100th time step.

We can not fully avoid this problem in two dimensions, but we can minimalize it by introducing fictitious material around our grid as an absorbing layer, so that outgoing waves mostly die out. This method is called Absorbing boundary condition (ABC), or in our case, more specifically, Perfectly matching layer (PML), which is a subject of the next chapter.

2. Boundary conditions

A goal of this chapter is to develop *Absorbing layer* around our problem space, that does not cause any reflections and is loosy so that any wave, coming to this layer, would die out before hitting it's end. There are several approaches available to solve this problem. The one, discussed in this chapter, is called *Perfectly Matched Layer*.

The idea of this approach is to surround our grid with additional layer of fictitious lossy material, that reduce the amplitude of outgoing wave by several orders of magnitude and so the reflection, caused on the outer boundary of the grid, would be less significant.

The amount of reflection on the interface of two materials A and B, caused by a wave, propagating from medium A into medium B, is dictated by reflective coefficient:

$$\Gamma = \frac{\eta_A - \eta_B}{\eta_A + \eta_B}, \tag{2.1}$$

where η is the impedance of respective medium:

$$\eta = \sqrt{\frac{\mu}{\varepsilon}}. \tag{2.2}$$

We have to make that reflective coefficient between our problem space and the PML layer to be equal to zero.

2.1 Perfectly matched layer

The first introduced approach is the PML, which is efficient in simple cases and does not cause a drastic increase in computaton time. Since working with anisotropic equations would be rather difficult, we are going to continue with isotropic version of Maxwell's equations without any magnetic charge or conductivity and move it to the frequency domain (these components can be added later on):

$$\begin{aligned} j\omega D_z &= c_0\Big(\frac{\partial H_y}{\partial}x - \frac{\partial H_x}{\partial y}\Big) \\ D_z(\omega) &= \varepsilon_r(\omega)E_z(\omega) \\ j\omega H_x &= -c_0\frac{\partial E_z}{\partial y} \\ j\omega H_y &= c_0\frac{\partial E_z}{\partial x} \end{aligned} \tag{2.3}$$

Note, that in these equations, we already substituted actual electric field with rescaled electric field:

$$\tilde{E} = \sqrt{\frac{\varepsilon_0}{\mu_0}}E, \tag{2.4}$$

and we are going to note it E for simplification. Also time step was chosen based on the cell size:

$$\Delta t = \frac{\Delta x}{2c_0} \tag{2.5}$$

We are going to introduce new fictitious complex dielectric constants and permeabilities ε^*_{Fz}, μ^*_{Fx} and μ^*_{Fy}, that are equal to 1 in our problem space, so that evolution equations are changed only in the new layer:

$$\begin{aligned} j\omega D_z \varepsilon^*_{Fz}(x)\varepsilon^*_{Fz}(y) &= c_0\Big(\frac{\partial H_y}{\partial} x - \frac{\partial H_x}{\partial y}\Big) \\ D_z(\omega) &= \varepsilon_r(\omega) E_z(\omega) \\ j\omega H_x \mu^*_{Fx}(x)\mu^*_{Fx}(y) &= -c_0 \frac{\partial E_z}{\partial y} \\ j\omega H_y \mu^*_{Fy}(x)\mu^*_{Fy}(y) &= c_0 \frac{\partial E_z}{\partial x} \end{aligned} \tag{2.6}$$

[2] suggests form of these new variables to be:

$$\begin{aligned} \varepsilon^*_{Fm} &= \varepsilon_{Fm} + \frac{\sigma_{Dm}}{j\omega\varepsilon_0} \\ \mu^*_{Fm} &= \mu_{Fm} + \frac{\sigma_{Hm}}{j\omega\mu_0} \\ m &= x \; or \; y \end{aligned} \tag{2.7}$$

with conditions to ensure zero reflectivity:

$$\begin{aligned} \varepsilon_{Fm} &= \mu_{Fm} = 1 \\ \frac{\sigma_{Dm}}{\varepsilon_0} &= \frac{\sigma_{Hm}}{\mu_0} = \frac{\sigma_D}{\varepsilon_0} \\ \varepsilon^*_{Fx} &= \frac{1}{\varepsilon^*_{Fy}} \\ \mu^*_{Fx} &= \frac{1}{\mu^*_{Fy}} \end{aligned} \tag{2.8}$$

Using this condition, we obtain zero reflection index, while going from problem space into PML. Now we only need the complex part of (2.7) to be gradually increasing, as it goes into the PML. In order to do this, σ_D has to be increasing.

Notice, that the new parameters, we added in (2.6), are dependant only on 1 coordinate. It is because in this form, we can simply decide, if we want to implement the PML in both directions, or just on only one and use some other form of boundary condition on the other one. (Periodic boundary condition for modeling periodic structures can be an example).

To make the further calculation simpler, let's focus on implementing PML only in one direction, compute prefactors before field variables and then guess a

form of these prefactors in the other direction. In the x direction we obtain:

$$\begin{aligned} j\omega\Big(1+\frac{\sigma_D(x)}{j\omega\varepsilon_0}\Big)D_z &= c_0\Big(\frac{\partial H_y}{\partial y}-\frac{\partial H_x}{\partial y}\Big) \\ j\omega\Big(1+\frac{\sigma_D(x)}{j\omega\varepsilon_0}\Big)^{-1}H_x &= -c_0\frac{\partial E_z}{\partial y} \\ j\omega\Big(1+\frac{\sigma_D(x)}{j\omega\varepsilon_0}\Big)H_y &= c_0\frac{\partial E_z}{\partial x} \end{aligned} \tag{2.9}$$

The next step is to move back into the time domain and apply the difference approximation. Doing so for the D_z and Hy we obtain:

$$\begin{aligned} D_z^{n+1/2}(i,j) &= gij3(i)\,D_z^{n-1/2}(i,j) + gi2(i)\,0.5\left[H_y^n(i+1/2,\,j)\right. \\ &\quad \left. - H_y^n(i-1/2,\,j - H_x^n(i,\,j+1/2) - H_x^n(i,\,j-1/2)\right] \\ H_y^{n+1}(i+1/2,\,j) &= fi3(i+1/2)H_y^n(i+1/2,\,j) \\ &\quad + fi2(i+1/2)\,0.5\left[E_z^{n+1/2}(i+1,\,j) - E_z^{n+1/2}(i,\,j)\right] \end{aligned} \tag{2.10}$$

The equation, evolving H_x, is a bit different from previous equations. It contains $(j\omega)^{-1}$, which relates to integration in time domain. Since we work with discretized entities, integration naturally becomes summation. To avoid an increasing summation in every single time step, which is a time demanding operation, we are going to define a new variable that holds the current value of that summation and which is updated with each time step. This can be done by implementing the following series of equations:

$$\begin{aligned} curl_e &= \left[E_z^{n+1/2}(i,\,j) - E_z^{n+1/2}(i,\,j+1)\right] \\ I_{Hx}^{n+1/2}(i,\,j+1/2) &= I_{Hx}^{n-1/2}(i,\,j+1/2) + curl_e \\ H_x^{n+1}(i,\,j+1/2) &= H_x^n(i,\,j+1/2) + 0.5\,curl_e + fi1(i)\,I_{Hx}^{n+1/2}(i,\,j+1/2) \end{aligned} \tag{2.11}$$

Before explicitly stating each coefficient, lets write down the whole set of equations, with PML implemented in both directions and having the conductivity

implemented:

$$\begin{aligned}
D_z[i,\,j] &= gi3\,gj3[j]\,D_z[i,\,j] \\
&\quad + gi2[i]\,gj2[j]\,\frac{0.5}{1+\sigma[i,\,j]}\Big(H_y[i,\,j] - H_y[i-1,j]\Big) \\
&\quad + gi2[i]\,gj2[j]\,\frac{1}{2k}\frac{1}{1+\sigma[i,\,j]} \\
E_z[i,\,j] &= \frac{1}{\varepsilon_r[i,\,j]}\,D_z[i,\,j] \\
curl_e[i,\,j] &= E_z[i,\,j] - E_z[i,\,j+1] \\
I_{Hx}[i,\,j] &= I_{Hx}[i,\,j] + fi1[i]\,curl_e[i,\,j] \\
H_x[i,\,j] &= fj3[j]\,H_x[i,\,j] + fj2[j]\,\frac{1}{2\mu_r}\Big(curl_e[i,\,j] + \frac{1}{k}I_{Hx}[i,\,j]\Big) \\
curl_e[i,\,j] &= E_z[i+1,\,j] - E_z[i,\,j] \\
I_{Hy}[i,\,j] &= I_{Hy}[i,\,j] + fi1[j]\,curl_e[i,\,j] \\
H_y[i,\,j] &= fj3[i]\,H_y[i,\,j] + fj2[i]\,\frac{1}{2\mu_r}\Big(curl_e[i,\,j] + I_{Hy}[i,\,j]\Big)
\end{aligned} \tag{2.12}$$

We did another simplification to the notation. Time steps are not noted anymore, because each new step is calculated from the previous values and so the left-hand side of the equation denotes the new time step and every component, standing on the right-hand side, denotes previous steps. Since there are no half indexes of array elements in the computer code, field indexes were shifted as well to reflect the future code. Also, here $\sigma[i,\,j]$ stands for actual conductivity, not just the imaginary one, we defined inside PML coefficients.

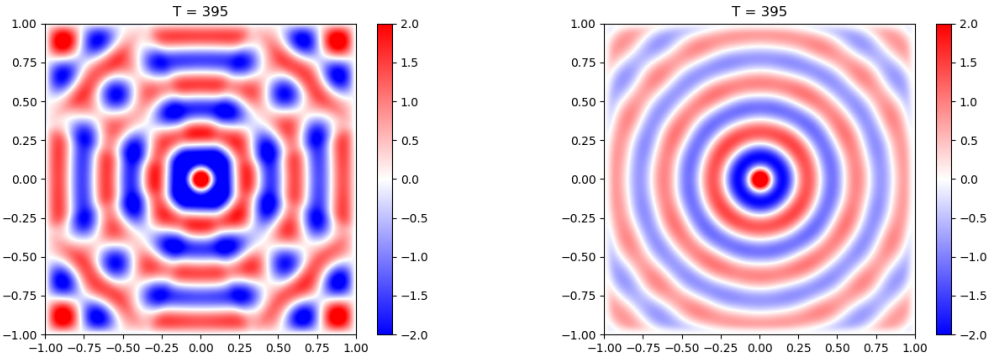
Introduced parameters can be written as following:

$$\begin{aligned}
fi1(i) &= xn(i) \\
fi2(i) &= \frac{1}{1+xn(i)} \\
fi3(i) &= \frac{1-xn(i)}{1+xn(i)} \\
gi2(i) &= \frac{1}{1+xn(i)} \\
gi3(i) &= \frac{1-xn(i)}{1+xn(i)},
\end{aligned} \tag{2.13}$$

and the same goes for prefactors, depending on the j parameter. Parameter xn can vary in the f and g sets of parameters. In [2], it takes a form of:

$$\begin{aligned}
f: &\quad xn(i) = \frac{1}{3}\Big(\frac{i}{npml}\Big)^3 \\
g: &\quad xn(i) = \frac{1}{4}\Big(\frac{i}{npml}\Big)^3
\end{aligned} \tag{2.14}$$

Important note is, that i starts at the edge of our problem space and grows into the PML layer, so it's value ranges between 0 and $npml$, where $npml$ is



(a) Propagation of source in free air without PML around the boundary

(b) Propagation of source in free air with 10 PML layers around the boundary

Figure 2.1: Demonstration of reflection on boundary with additional 10 layers of PML. The source is a sinusoidal pulse with profile $10\,sin(n/10)$, centered in the middle of the grid. The grid represents a free space.

the thickness of PML layers in cells. This ensures that newly added parameters are equal to one in our problem space and so they cause no effect on original equations there.

The implementation of this algorithm can be demonstrated on picture 2.1. While there is still some reflection noticeable, it's magnitude is way lower, then any other field, existing inside our problem space. The amount of reflection can be reduced even further by changing coefficients in equation (2.14), but that is specific for every problem.

2.2 Complex Frequency-shifted PML

CPML should be more robust, while dealing with evanescent waves, but other then that, it is a much more time demanding implementation of the boundary conditions.

Following the procedure, described in [1], the first step is taking the (1.15) to frequency domain by substituting the time derivative $\frac{\partial}{\partial t}$ by $j\omega$, where j is the imaginary unit and instead of using just *Constitutive relations*, we are going to introduce *Stretched coordinate metrics* S_{ex}, S_{ey}, S_{mx} and S_{my}:

$$\begin{aligned} j\omega\varepsilon_z E_z + \sigma_z^e E_z + j_{iz} &= \frac{1}{S_{ex}}\frac{\partial H_y}{\partial x} - \frac{1}{S_{ey}}\frac{\partial H_x}{\partial y} \\ j\omega\mu_x H_x + \sigma_x^m H_x + M_{ix} &= -\frac{1}{S_{my}}\frac{\partial E_z}{\partial y} \\ j\omega\mu_y H_y + \sigma_y^m H_y + M_{iy} &= \frac{1}{S_{mx}}\frac{\partial E_z}{\partial x} \end{aligned} \tag{2.15}$$

We do not actually need to use impressed current density, because these equations describes the behaviour of CPML areas, which is outside of our problem space, but we can also include them and set them to zero on the outside of the problem space. This allows us to have just a single set of equations for whole grid

with several parameters that holds only on certain areas. This may cause some increase in complexity of our equations, but it simplifies the writing of the code, since we would not have to check which set of equations, we are supposed to use.

The form of *Stretched coordinate metrics* is described in [3]:

$$\begin{aligned} S_{ei} &= \kappa_{ei} + \frac{\sigma_{pei}}{\alpha_{ei} + j\omega\varepsilon_0} \\ S_{mi} &= \kappa_{mi} + \frac{\sigma_{pmi}}{\alpha_{mi} + j\omega\varepsilon_0} \\ i &= x, y \end{aligned} \tag{2.16}$$

σ_{pex}, σ_{pey}, σ_{pmx} and σ_{pmy} are new fictious conductivities, κ_{ei}, κ_{mi}, α_{ei} and α_{mi} are new CPML parameters.

Since we want to develop new evolution eqautions, we have to move back to time domain. Using the inverse Laplace transform, we obtain:

$$\begin{aligned} \varepsilon_z \frac{\partial E_z}{\partial_t} + \sigma_z^e E_z + j_{iz} &= \zeta_{ex} * \frac{\partial H_y}{\partial x} - \zeta_{ey} * \frac{\partial H_x}{\partial y} \\ \mu_x \frac{\partial H_x}{\partial t} + \sigma_x^m H_x + M_{ix} &= -\zeta_{my} * \frac{\partial E_z}{\partial y} \\ \mu_y \frac{\partial H_y}{\partial t} + \sigma_y^m H_y + M_{iy} &= \zeta_{mx} * \frac{\partial E_z}{\partial x}, \end{aligned} \tag{2.17}$$

where ζ is the inverse Laplace transform of $\frac{1}{S}$.

$$\begin{aligned} \zeta_{ei}(t) &= \frac{\delta(t)}{\kappa_{ei}} - \frac{\sigma_{pei}}{\varepsilon_0 \kappa_{ei}^2} e^{-\left(\frac{\sigma_{pei}}{\varepsilon_0 \kappa_{ei}} + \frac{\sigma_{pei}}{\varepsilon_0}\right)t} u(t) = \frac{\delta(t)}{\kappa_{ei}} + \xi_{ei}(t) \\ \zeta_{mi}(t) &= \frac{\delta(t)}{\kappa_{mi}} - \frac{\sigma_{pmi}}{\varepsilon_0 \kappa_{mi}^2} e^{-\left(\frac{\sigma_{pmi}}{\varepsilon_0 \kappa_{mi}} + \frac{\sigma_{pmi}}{\varepsilon_0}\right)t} u(t) = \frac{\delta(t)}{\kappa_{mi}} + \xi_{mi}(t) \end{aligned} \tag{2.18}$$

As we can see, multiplication become convolution. Before applying central difference approximation (1.18), we have to have a look at discrete convolution first. These time convolutions takes form of:

$$\begin{aligned} \xi_{ex} * \frac{\partial H_y}{\partial x} &= \int_{\tau=0}^{\tau=t} \xi_{ex}(\tau) \frac{\partial H_y(t-\tau)}{\partial x} d\tau \\ &\approx \sum_{m=0}^{m=n-1} Z_{0ex}(m) \Big(H_y^{n-m+\frac{1}{2}}(i,\, j) - H_y^{n-m+\frac{1}{2}}(i-1,\, j) \Big), \end{aligned} \tag{2.19}$$

where

$$Z_{0ex}(m) = \frac{\rho_{pex}}{\Delta x(\rho_{pex}\kappa_{ex} + \alpha_{ex}\kappa_{ex}^2} \left[e^{-(\frac{\rho_{pex}}{\kappa_{ex}} + \alpha_{pex})\frac{\Delta t}{\varepsilon_0}} - 1 \right] e^{-(\frac{\rho_{pex}}{\kappa_{ex}} + \alpha_{pex})\frac{m\Delta t}{\varepsilon_0}} \tag{2.20}$$

In order to shorter our equations, lets use following substitution:

$$\Psi_{ezx}^{n+\frac{1}{2}}(i,\, j) = \sum_{m=0}^{m=n-1} Z_{0ex}(m) \Big(H_y^{n-m+\frac{1}{2}}(i,\, j) - H_y^{n-m+\frac{1}{2}}(i-1,\, j) \Big) \tag{2.21}$$

The subscript ezx means, that this term belongs to updating equation of E_z and it contains derivation in x direction.

Examining the expression (2.19), we realize, that it is calculated from every previous time step which means that we would have to keep every previous time step stored, which would quickly overflow our available memory. However, the time appears only in one exponential and in the sum of the deference terms, so we can simplify the computation to use only the previous time step:

$$\begin{aligned}
\Psi_{ezx}^{n+\frac{1}{2}}(i,\,j) &= b_{ex}\Psi_{ezx}^{n-\frac{1}{2}}(i,\,j) + a_{ex}\Big(H_y^{n+\frac{1}{2}}(i,\,j) - H_y^{n+\frac{1}{2}}(i-1,\,j)\Big) \\
a_{ex} &= \frac{\sigma_{pex}}{\Delta x(\sigma_{pex}\kappa_{ex} + \alpha_{ex}\kappa_{ex}^2)}\Big[b_{ex} - 1\Big] \\
b_{ex} &= e^{-(\frac{\sigma_{pex}}{\kappa_{ex}} + \alpha_{ex})\frac{\Delta t}{\varepsilon_0}}
\end{aligned} \tag{2.22}$$

This is called The Recursive Convolution Method and it is derived in detail in [1], page 235.

We can do the same for magnetic version of this term:

$$\begin{aligned}
\Psi_{mxy}^{n+\frac{1}{2}}(i,\,j) &= b_{my}\Psi_{mxy}^{n-\frac{1}{2}}(i,\,j) + a_{my}\Big(E_z^{n+1}(i,\,j) - E_z^{n+1}(i-1,\,j)\Big) \\
a_{my} &= \frac{\sigma_{pmy}}{\Delta x(\sigma_{pmy}\kappa_{my} + \alpha_{my}\kappa_{ex}^2)}\Big[b_{my} - 1\Big] \\
b_{my} &= e^{-(\frac{\sigma_{pmy}}{\kappa_{my}} + \alpha_{my})\frac{\Delta t}{\mu_0}}
\end{aligned} \tag{2.23}$$

Using this technique and following the same procedure, as derived in the first chapter, we obtain:

$$\begin{aligned}
\tilde{E}_z^{n+1}(i,\,j) &= C_{ezez}(i,\,j)\tilde{E}_z^n(i,\,j) - C_{ezjiz}(i,\,j)j_{iz}^{n+\frac{1}{2}}(i,\,j) \\
&\quad + \frac{1}{\kappa_{ex}(i,\,j)}C_{ezhy}(i,\,j)\Big[H_y^{n+\frac{1}{2}}(i,\,j) - H_y^{n+\frac{1}{2}}(i-1,\,j)\Big] \\
&\quad - \frac{1}{\kappa_{ey}(i,\,j)}C_{ezhx}(i,\,j)\Big[H_x^{n+\frac{1}{2}}(i,\,j) - H_x^{n+\frac{1}{2}}(i,\,j-1)\Big] \\
&\quad - \Delta y C_{ezhx}(i,\,j)\Psi_{ezy}^{n+\frac{1}{2}}(i,\,j) + \Delta x C_{ezhy}(i,\,j)\Psi_{ezx}^{n+\frac{1}{2}}(i,\,j) \\
H_x^{n+\frac{1}{2}}(i,\,j) &= C_{hxhx}(i,\,j)H_x^{n-\frac{1}{2}}(i,\,j) - C_{hxmix}(i,\,j)M_{ix}^n(i,\,j) \\
&\quad - \frac{1}{\kappa_{my}(i,\,j)}C_{hxez}(i,\,j)\Big[\tilde{E}_z^n(i,\,j+1) - \tilde{E}_z^n(i,\,j)\Big] \\
&\quad - \Delta y C_{hxez}(i,\,j)\Psi_{mxy}(i,\,j) \\
H_y^{n+\frac{1}{2}}(i,\,j) &= C_{hyhy}(i,\,j)H_y^{n-\frac{1}{2}}(i,\,j) - C_{hymiy}(i,\,j)M_{iy}^n(i,\,j) \\
&\quad + \frac{1}{\kappa_{mx}(i,\,j)}C_{hyez}(i,\,j)\Big[\tilde{E}_z^n(i+1,\,j) - \tilde{E}_z^n(i,\,j)\Big] \\
&\quad + \Delta x C_{hyez}\Psi_{myx}(i,\,j)
\end{aligned} \tag{2.24}$$

These are our updating equations for our FDTD code with CPML implemented. We defined a lot of new parameters in this chapter, but hadn't discussed it's values yet.

2.2.1 CPML parameters

The first set of parameters, we are going to have a look at, is the set of CPML σ parameters. [1] suggests using polynomial scaling of σ_{pei}. Starting from 0 at problem interface boundary and inside of the problem space, scaling to σ_{max} at the external boundary:

$$\sigma_{pei}(\rho) = \sigma_{max}\left(\frac{\rho}{\delta}\right)^{n_{pol}} \tag{2.25}$$

Where ρ is the distance from inner CPML boundary, δ is CPML thickness and n_{pol} is the polynomial order used. [2] empirically found $n_{pol} = 3$ to be the most effective stable variation, but [1] is not so scrict and accepts any value between 2 and 4. σ_{max} is calculated as:

$$\sigma_{max} = \sigma_{sc}\frac{n_{pol}+1}{150\pi\sqrt{\varepsilon_r}\Delta i} \tag{2.26}$$

Optimal value for σ_{sc} is between 0.7 and 1.5, Δi means Δx or Δy, which is a size of one cell in corresponding dimension. As we can see, σ_{max} contains information about material inside it. This is, where PML and CPML differs, CPML is able to simulate objects, that are not fully contained in problem space.

Since we want to avoid reflections at grid-CPML interface, the following equation has to hold for our parameters:

$$\frac{\sigma_{pei}}{\varepsilon_0} = \frac{\sigma_{pmi}}{\mu_0} \tag{2.27}$$

To satisfy this condition, we simply chose our other paramter to be:

$$\sigma_{pmi}(\rho) = \frac{\mu_0}{\varepsilon_0}\sigma_{max}\left(\frac{\rho}{\delta}\right)^{n_{pol}} \tag{2.28}$$

Another parameter, we defined in this chapter, is κ. This parameter has to be equal to 1 in our problem space, while scaling towards κ_{max} in the CPML area. We can once again use polynomial scaling:

$$\kappa_{ei}(\rho) = 1 + (\kappa_{max} - 1)\left(\frac{\rho}{\delta}\right)^{n_{pol}} \tag{2.29}$$

$$\kappa_{mi}(\rho) = 1 + (\kappa_{max} - 1)\left(\frac{\rho}{\delta}\right)^{n_{pol}} \tag{2.30}$$

Difference between (2.29) and (2.30) is that electric and magnetic fields are calculated in different positions, therefore ρ is not the same in these relations, but rather shifted by half of the cell size. The recomanded value of κ_{max} is in range $(5 - 11)$.

The last set of parameters, we defined, is the set of α parameters. These parameters takes the maximum value at the inner domain-CPML interface and scales linearly to minimum value at the outer edge:

$$\alpha_{ei}(\rho) = \alpha_{min} + (\alpha_{max} - \alpha_{min})\left(1 - \frac{\rho}{\delta}\right) \tag{2.31}$$

The magnetic version of α is once again rescaled by ε_0 and μ_0:

$$\alpha_{mi}(\rho) = \frac{\mu_0}{\varepsilon_0}\left(\alpha_{min} + (\alpha_{max} - \alpha_{min})\left(1 - \frac{\rho}{\delta}\right)\right) \tag{2.32}$$

The α_{max} should be chosen in range from 0 to 0.05.

2.3 Periodic boundary condition

In contrast to previously discussed boundary condition, periodic boundary condition does not add any additional layers to our grid. It is used for modeling periodic structures in a steady state.

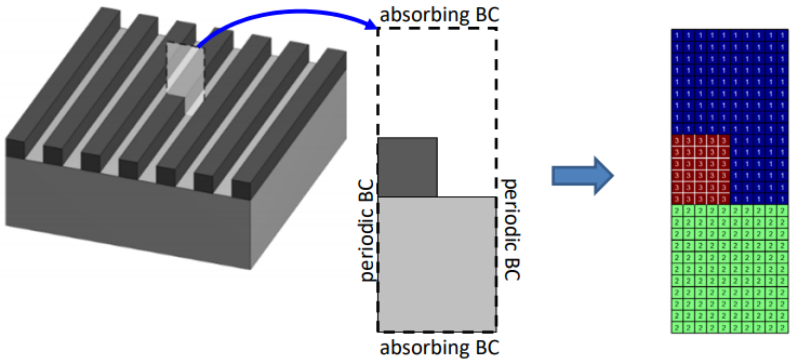


Figure 2.2: Example of periodic boundary condition implementation. This picture is taken from [4]

As shown on picture 2.2, periodic boundary condition has to be implemented in certain direction/s, not just some side, as PML/CPML. Previous boundary conditions had to be implemented, because cells on the edge of the grid were missing neighboring cells to compute a new time step, which caused reflection as shown on picture (1.1). This time however, the missing cell is substituted by the cell on the other side of the grid and no reflection is caused. Let us suppose we have a grid as shown on picture (2.2) and a wave, propagating towards the left edge of the grid. Once it hits the edge, it does not reflect, but it appears on the right side of the grid instead:

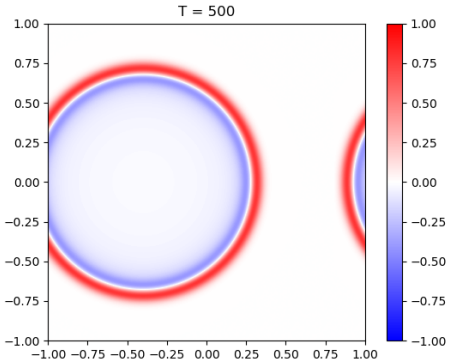


Figure 2.3: Cylindrical wave with gausian profile propgating trought periodic boundary condition

3. Writing the code in Python

We developed a way to compute *Maxwell's equations* on a finite grid and a way to deal with waves, propagating towards the edge of the grid. The task that lies ahead is to write these obtained formulas in a form, that computer would understand and could solve them as fast as possible.

As mentioned before, FDTD simulations can be really demanding on the computation time and hardware used. As and example, imagine, we have a (400 x 400) grid of cells, where we want to compute 10000 time steps. Having a brief look at (2.12) or (2.24), we realize, that we have to compute 5 or 7 parameters in each cell during every single time step. This mean, that we face a problem of computing roughly $8 - 10 \cdot 10^9$ variables, still in two dimensions.

Lets start this chapter with one of the flowchart of the code, we are supposed to write:

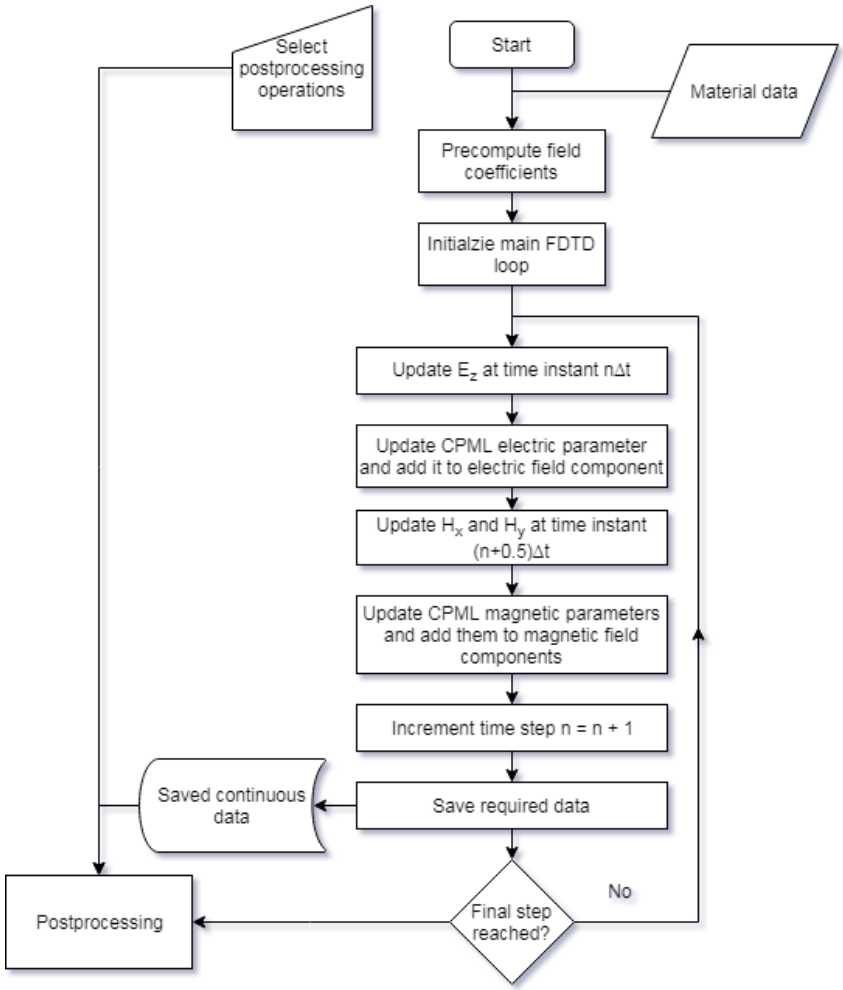


Figure 3.1: Flowchart of FDTD main loop with CPML implementation

What is important on this flowchart, is the order of operations in main loop, which is where the field computation together with border conditions is being computed. Additional post processing, such as far field computation or Fourier transformation, is computed as an post processing operation, so it does not in-

terfere with the main loop except adding additional requirement on data saving. Field coeffitients are precomputed before the main loop starts, so that the computation can run as fast as possible.

3.1 Choosing Python as a programming language

There are many examples of FDTD code, written predominantly in *MATLAB* and *C*, but for this task, *Python* was chosen thanks to a wide range of external advanced libraries and because an advanced code can be written in *Python* with ease.

The largest advantage, of choosing *Python* for this task, is that it supports *Torch* via *PyTorch* library. *PyTorch* is an open source deep learning platform, developed by *Facebook*, with a powerhouse of functions for image processing. *PyTorch* supports *CUDA* (Compute Unified Device Architecture), which allows software to be processed by graphic card. Since we are dealing mostly with 2-dimensional arrays, we can think about these arrays as about a bitmaps, that has to be processed and so using a graphic card would significantly increase the program performance. Another bonus is, that the parallelism between *CUDA* cores is already pre-written in *PyTorch* functions, which simplify the code writing significantly.

3.2 Update equation in optimal form

Having a closer look at equations (2.12) or (2.24), one quickly realize, that the field at the following time step depends on neighboring cells of the current time step. We can take an advantage of this and rewrite these operations, using the convolution operator, which is deeply implemented in many programming languages.

There is a part in our updating equations (2.24 that repeat itself in various forms and so our program spends most of the time computing this formula:

$$C(i,\, j)\Big(F(i,\, j) + F(i+1,\, j)\Big) \tag{3.1}$$

Where $C(i,\, j)$ stands for general constant, standing before additive bracket and $F(i,\, j)$ is some time dependant field element.

Instead of rewriting each cell of the grid individually, we can define an operation that proceeds the whole time step as a single variable. That operation would be the discrete convolution. Term (3.1) can be written as:

$$\begin{aligned} &C(i,\, j)\Big(F * K_{+i+}\Big)(i,\, j) \\ &K_{+i+} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \end{aligned} \tag{3.2}$$

where F is a field with elements $F(i,\, j)$ and K_{+i+} is a convolution kernel. The subscript of the kernel is written according to following rules: First symbol (+

or -) means, that we add/subtract another element to/from the central element respectively. Second symbol means, in which direction lies the other element. The last symbol means, in which on which $(+i+)$ means, that we realizing addition of central and following element in i (x) direction.

Using this, we can rewrite update equations (2.24) into computer code.

$$\begin{aligned}
\tilde{E}_z &= C_{ez1}(i,\,j)\tilde{E}_z - C_{ez2}j_{iz}^{n+\frac{1}{2}} \\
&\quad + C_{ez3}\Big(H_y * K_{-i-}\Big) - C_{ez4}\Big(H_x * K_{-j-}\Big) \\
&\quad - C_{ez5}\Psi_{ezy} + C_{ez6}\Psi_{ezx} \\
\Psi_{mxy} &= b_{my}\Psi_{mxy} + a_{my}\Big(E_z * K_{-j-}\Big) \\
\Psi_{myx} &= b_{mx}\Psi_{myx} + a_{mx}\Big(E_z * K_{-i-}\Big) \\
H_x &= C_{hx1}(i,\,j)H_x - C_{hx2}M_{ix}^n(i,\,j) \\
&\quad + C_{hx3}\Big(\tilde{E}_z * K_{-j+}\Big) - C_{hx4}\Psi_{mxy} \\
\Psi_{ezy} &= b_{ey}\Psi_{ezy} + a_{ey}\Big(H_x * K_{-j-}\Big) \\
H^y &= C_{hy1}H_y - C_{hy2}M_{iy}^n \\
&\quad - C_{hy3}\Big(\tilde{E}_z * K_{-i+}\Big) \\
&\quad + C_{hy4}\Psi_{myx}(i,\,j) \\
\Psi_{ezx} &= b_{ex}\Psi_{ezx} + a_{ex}\Big(H_y * K_{-i-}\Big),
\end{aligned} \tag{3.3}$$

where constant parameters and kernels can be written by components:

$$\begin{aligned}
K_{-i+} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix} \\
K_{-i-} &= \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
K_{-j+} &= \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \\
K_{-j-} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}
\end{aligned} \tag{3.4}$$

$$
\begin{aligned}
C_{ez1}(i,\,j) &= \frac{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 - \sigma_z^e(i,\,j)\Delta x}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} \\
C_{ez2}(i,\,j) &= \frac{2\Delta x}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} \\
C_{ez3}(i,\,j) &= \frac{2\kappa_{ex}^{-1}(i,\,j)}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} \\
C_{ez4}(i,\,j) &= \frac{2k\kappa_{ey}^{-1}(i,\,j)}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} \\
C_{ez5}(i,\,j) &= \frac{2\Delta x}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} \\
C_{ez6}(i,\,j) &= \frac{2\Delta x}{4\varepsilon_0\varepsilon_{rz}(i,\,j)c_0 + \sigma_z^e(i,\,j)\Delta x} \\
C_{hx1}(i,\,j) &= \frac{4c_0\mu_0\mu_{rx}(i,\,j) - \Delta x\sigma_x^m(i,\,j)}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)} \\
C_{hx2}(i,\,j) &= \frac{2\Delta_x}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)} \\
C_{hx3}(i,\,j) &= \frac{2k\kappa_{my}^{-1}(i,\,j)}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)} \\
C_{hx4}(i,\,j) &= \frac{2\Delta x}{4c_0\mu_0\mu_{rx}(i,\,j) + \Delta x\sigma_x^m(i,\,j)} \\
C_{hy1}(i,\,j) &= \frac{4c_0\mu_0\mu_{ry}(i,\,j) - \Delta x\sigma_y^m(i,\,j)}{4c_0\mu_0\mu_r(i,\,j) + \Delta x\sigma_y^m(i,\,j)} \\
C_{hy2}(i,\,j) &= \frac{2\Delta x}{4c_0\mu_0\mu_{ry}(,\,j) + \Delta x\sigma_y^m(i,\,j)} \\
C_{hy3}(i,\,j) &= \frac{2\kappa_{mx}^{-1}(i,\,j)}{4c_0\mu_0\mu_{ry}(i,\,j) + \Delta x\sigma_y^m(i,\,j)} \\
C_{hy4}(i,\,j) &= \frac{2\Delta x}{4c_0\mu_0\mu_{ry}(i,\,j) + \Delta x\sigma_y^m(i,\,j)}
\end{aligned}
\tag{3.5}
$$

Parameters a and b were already discussed in the second chapter. (2.22), (2.23)

One can notice, that upper indexes, indicating time steps, disappeared at parameters, that are not given. In fact, right hand side of the equations contains the following time step, which is yet to be computed from the current ones, standing on the right hand side of these equations. This means, that previous time steps are being overwritten each time the new one is being computed. We are allowed to do so, because equations are in such order, that once the new time step is computed, the old one is no longer needed in the same step of the loop. The advantage is, that we can save a lot of computer memory. Another change, compared to equations (2.24) is, that we no longer use spatial indexes, because we now have our variables on form of bitmaps and so we do not have to use element operations, as we did before.

Another note is, that we can actually reduce the amount of coefficients because in (3.5) we can see that $C_{ez2} = C_{ez6}$. It is not much, but once again, it saves the

memory used.

3.3 Parallelization

Parallelization is an important tool in numerical simulations. Since we live in an era, when almost every commercial processor has more than one core, exploiting it could potentially lead to huge increase in raw computation power.

This, however is easier said then done. There are generally two approaches when it comes to parallelism in Python: Multithreading and Multiprocessing. First problem with these is, that we are already exploiting external, well-written libraries, written in *C* (*NumPy*, *SciPy*), because they offer a very effective solution when it comes to computing multidimensional convolution. Applying *Multithreading* or *Multiprocessing* to these would mean slicing our multidimensional arrays into slices of one dimension lower. Only then we could slit the convolution operator between multiple threads or processes, but then we would have to staple the results. Adding these extra steps into our algorithm does not really healp, because even if we do so, there is another problem with this approach: *Global interpreter lock* (GIL). More information can be found in [5].

Even tho, there are some reasonable way to parallelize NumPy functions, we decided not to do so, bceause instead of using CPU, we can move this problem to GPU. For this reason we decided to use already parallelized library called *PyTorch*, which can run on GPU using CUDA. We import this library using the standart importing command:

```
import torch
import torch.nn.funkcional as F
```

In order to use the GPU, we have to add the device selecting command. Since CUDA is not available on every device, we implement test for availability:

```
device = torch.device("cuda:0" if
                      torch.cuda.is_available()
                      else "cpu")
```

In the *cuda:0* command the number selects, which graphic card we want to use. This allows us to fully exploit multi-GPU devices. For example, we could compute TE mode on one GPU and TE mode on the other one.

Another step is to convert our variables into PyTorch data frame, which is a tensor. Variables, that are not processed in the main loop, are created as *NumPy* arrays. NumPy array a can be easyly converted into Torch tensor a by the following instruction:

```
a = torch.as_tensor(a),
```

however, we also have to fix the issue with dimensional compatibility. Since there are some extra channels in Torch 2D convolution, which are not interested for our use, our two dimensional Numpy array has to be converted into four dimensional array. To do so, a function "array_to_tensor()" is introduced within provided library. The used data type is Float, but it can be converted into double. There is also a function to inverse conversion called "tensor_to_array()" because the post processing is managed back on CPU and the Numpy array is more versatile from the function point of view.

Once we have our tensors properly converted, we have to specify, on which device they should be processed. In order to move the tensor "a" to the GPU, the following command has to be used:

```
a = a.to(device)
```

Having the tensors are moved to GPU, the actual computation of evolution equation derived in the previous chapter is done by class called "torch.nn.Conv2d" which handles the convolution operation. When the main loop is finished, tensors can be moved back to CPU and then back to Numpy arrays for further post processing.

3.4 Computation speed

Moving the computation from CPU to GPU, using the torch tensors increases the computation speed significantly. To measure how exactly, the testing was performed on GPU *Nvidia GTX 1050 Ti*. The simulation used for this purpose is a monochromatic plain wave propagating trough free space with PML around computation grid. Since amount of operation during each simulation with fixed grid size and amount of time steps is the same, we expect the a similar result for more complex problems. The behavior of computation speed is shown on graph 3.2. One can clearly notice that this method prefer using larger grid size. This has to do with GPU cores saturation. The maximum saturation was reached at around grid size of $4000 \cdot 4000$ cells with computation speed of 120 milions cells per second.

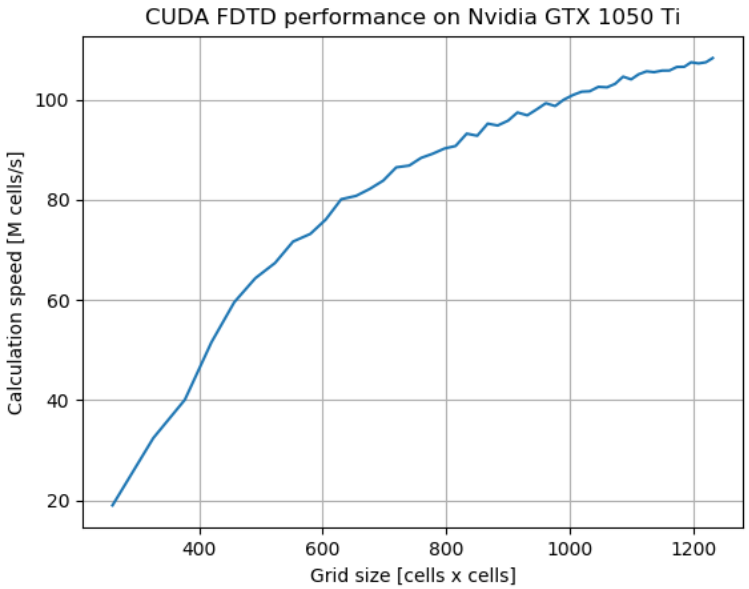


Figure 3.2: Graph of the calculation speed, using different size of grid. Grid size already include PML layers.

4. Preparing computation input

As described by the flowchart at the beginning of previous chapter (3.1), the computation require Material data. In this chapter, we are going to discuss how to select the right grid resolution, how to model a real world device on an orthogonal Yee-grid, how to improve the convergence of our simulation and what are the limits for the spatial and frequency resolution.

4.1 Grid parameters and resolution

In the first chapter, we established relation between spatial and time steps (1.28). The time step can be properly determined by condition that electromagnetic wave, propagating at maximum speed (speed of light), can propagate only trough one cell per each time step. Slowing the propagation trough cell would lead to more stable solution, but also to increase in length of the simulation and so a reasonable solution is the following formula:

$$\Delta t \leq \frac{\Delta x}{2c_0}, \tag{4.1}$$

which is where (1.28) came from.

Having the time step determined by the spatial step, the spatial step is the only grid parameter we have yet to define. Within this task, we are supposed to choose the right sampling of used wavelength. This sampling depends on many factors, as discussed in [6] and [7], but it is known to use at least 10 steps per wavelength is enough. It is important to remember that we have to take in count that this has to be held in every cell of the *Yee-grid*. This means that we have to take the shortest wavelength, we want to simulate and divide it by the largest refractive index in our simulation. Then we can properly determine the the adequate cell by "divide by ten" rule.

Another parameter we have to discuss, is the length of the simulation. This is important in case of frequency response of our device, because calculating the Fourier transform of our output require infinite amount of time steps, which we can not provide. An intuitive example is to look at this problem from the other way around. Let us suppose that we have a source, described by itš spectrum. We run a simulation of this source in our program and measure it to get the output. In theory, if we calculate the Fourier transform of our output, we should get the spectrum of the source we used, but that holds only if our simulation goes to infinity, but since it does not, we get our source spectrum blurred.

As an example from [4], let $H(\omega)$ be our source spectrum:

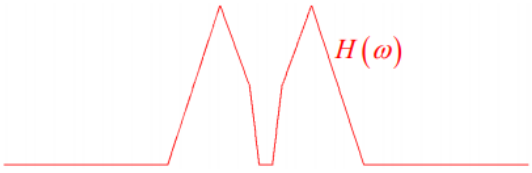


Figure 4.1: Spectrum of the example source. This picture is taken from [4].

The field, generated by this source, would be $h(t)$:

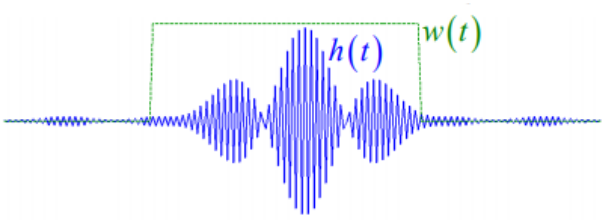


Figure 4.2: Generated field vs measured field. This picture is taken from [4].

But since our simulation is finite, we do not measure the $h(t)$ function, but rather $h(t) \cdot w(t)$, where $w(t)$ is function, which is equal to 1 within our simulation and zero otherwise. Having a look at spectrum based on this result, we obtain blurred spectrum:

$$H_{blurred}(\omega) = F\Big\{h(t) \cdot w(t)\Big\} = F\Big\{h(t)\Big\} * F\Big\{w(t)\Big\} = H(\omega) * W(\omega), \tag{4.2}$$

where $W(\omega)$ is rescaled sinc function. Extending the simulation to infinity, the sinc function $W(\omega)$ would limit to delta function.

This means, that our spectrum, calculated by the program, is going to be blurred and the amount of blurring depends on the number of time steps, which we are going to compute. Having a look at the *Nyquist sampling theorem* [[8]]:

$$f_{max} = \frac{1}{2\Delta t}, \tag{4.3}$$

we have some relation between the maximal frequency, we are going to simulate and the time step. (Even tho that we are using factor 10 instead of 2, but that is just because of numerical stability requirement. According to [4], the relation between spectral resolution and the amount of steps is:

$$\Delta f = \frac{2 f_{max}}{N_{steps}}, \tag{4.4}$$

where Δf is the uncertainty in the spectrum and N_{steps} is the amount of time steps simulated.

Combining these two equations together, we obtain a criterion on N_{steps}:

$$N_{steps} \geq \frac{1}{\Delta t \Delta f} \tag{4.5}$$

4.2 Building an object on the Yee-grid

As mentioned before, we made a shift in the positions of vector field components in our grid (1.19), but we did not adress this in terms of permittivity and permeability tensor components. We could have say, that we keep all of our permittivity and permeability tensor components on the same places, without shifting either of them and then calculating the avarage value of naigboring components in order to calculate the value for shifted fields, but that would lead to another operation in our updating equations and since we want to run our simulation over extensive

amount of time and hence time steps. This would lead to unnecesery increase in machine-time computation.

For the reason above, we have to define each of the material tensor components on different spatial position, matching the position of field they stands at.

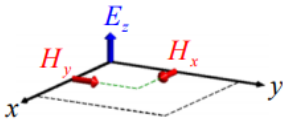


Figure 4.3: Electric and magnetic field vector components spatial position in TM mode. This picture is taken from [4].

In order to do so, in the provided program for building material input arrays, the real world coordinate is shifted, after in input is given, by half on a cell size in the direction, based on selected material parameter, that is being created.

4.3 Staircase smoothing

Another problem that rises from simulating an object on *Yee-grid* is that *Yee-grid* is orthogonal and cells are finite so that when we want to model something, that is not orthogonal, any curve has to be approximated by rectangles, causing the staircase effect:

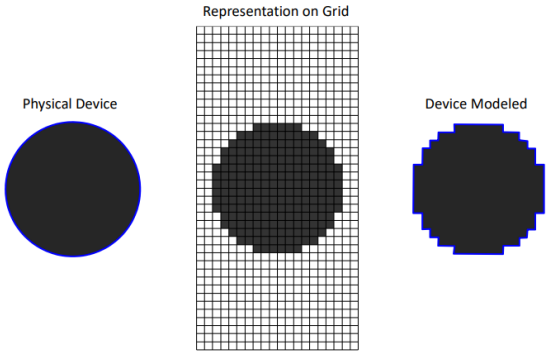


Figure 4.4: Staircase effect, caused by modeling a device on finite orthogonal grid. This picture is taken from [4].

We could reduce this effect by decreasing the size of a cell, but that would lead to large increase of cells that we have to use and since we made our time step fixed, based on the size of an cell (1.28), we would have to compute much more time steps in order to reach desired time.

There are several other options that we are going to discuss. First and the simplest option is to blur the staircased device by convolving it with some smoothing function. This would work, but only if we blur up to one pixel of distance. In other words, the support of the smoothing function would have to be 3×3 matrix. Another way could be modeling the device on grid with much higher resolution and then mapping that detailed grid into lower resolution grid by calculating cells in the lover resolution grid as an average of cells in the high resolution grid. It is

stated in [4], that in order to reach the fastest convergence, the object has to be built on the higher resolution grid, then smoothed by convolving with one pixel of lover resolution grid, built on higher resolution grid so that sum of each subpixel is equal to one, and then calculating the lover resolution pixel value as a value of a central subpixel on the higher resloution grid:

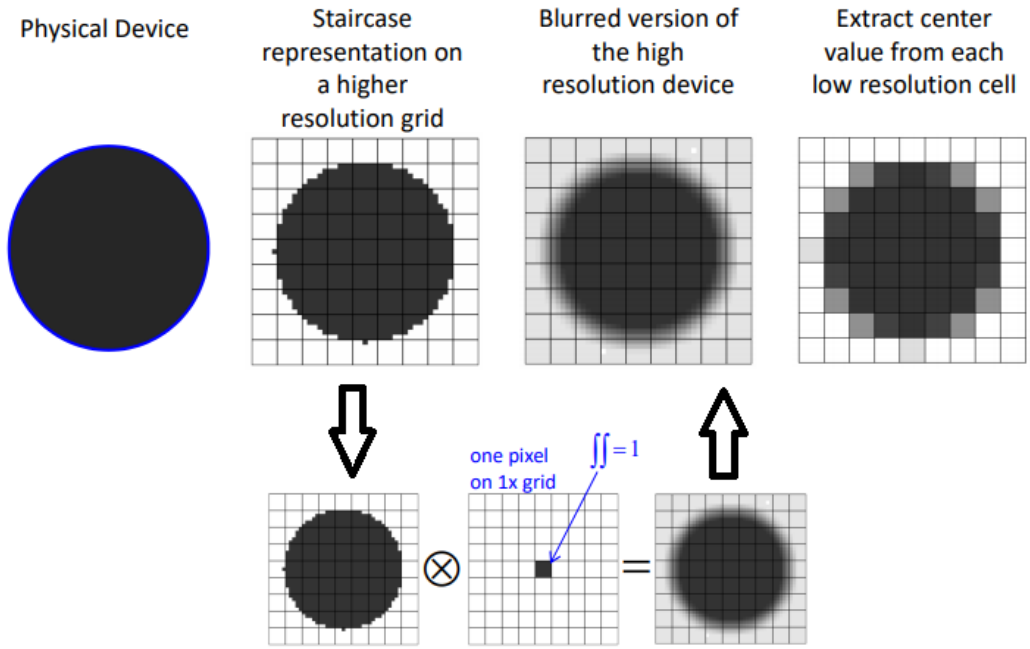
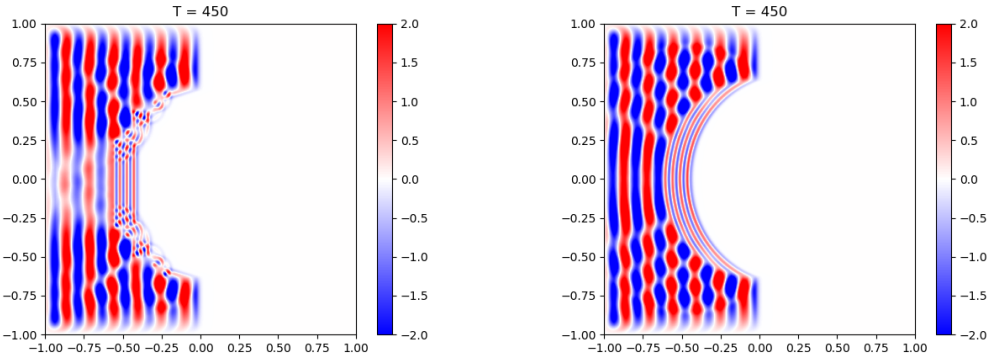


Figure 4.5: Smoothing procedure, using the detailed grid technique and convolution. This picture is taken from [4].

The result of this procedure is demonstrated on picture 4.6, where a dielectric circle with $\varepsilon_r = 10$, is iluminated by sinusoidal wavefront.

The following flowchart represents the procedure connection in Material creating file. It provides an option to ether create a new, or edit an existing material file. Right now, since the simulation only handles two-dimensional problem, there is an option to create only two-dimensional material files, but can be easily expanded for other cases. Once the dimensions and sampling parameters are selected, the high resolution grid is created. Into this grid, there are options for adding new shapes with various material parameters. Once all shapes are added, the high resolution grid is convolved with a lower resolution pixel of normalized value. The central point of each low resolution pixel in the high resolution grid is then selected to represent the value on the lover resolution grid. There is an option to visually check the material file created and to edit it further. Once the user is satisfied with the result, the file is saved in form of *NumPy array* and PNG image.

Picture 4.6 shows, how does the grid detail and smoothing procedure influence the shape of wavefront. The comparasion was made on delicate grid, because the shape error is always under the size of grid cell.



(a) Plane wave hitting the staircased dielectric circle.

(b) Plane wave hitting the smoothed dielectric circle.

Figure 4.6: Demonstration of the difference between rawly constructed round object on the Yee grid versus an object, constructed on more delicate one that was smoothed by the described procedure.

4.4 Source generation

Another important topic for FDTD simulation is the way, how is the source implemented. There are two types of sources - hard and soft. Hard source is a source that overwrites data on a specified point of the grid, while soft source only adds defined value to that point. Using hard sources is not recommended because they work as an edge of the grid and causes unwanted reflections.

Electric field is chosen to be the source field in this simulation and the way it is added is, that before the main loop is engaged, values of the source are computed on a slice of the grid at every given time step for the duration of simulation. This way, we do not add another evaluation into main loop and addition of the precomputed slice is the only new operation. We only need to know the source field on one slice because the solution is in fact Causchy's problem.

If we want to use only sources, which's wave vectors are parallel with one of the axes, we can improve the source generation by using the *Total/Scattered field formulation*, TFSF for short, described in [2], page 58, which splits the field to the incoming field, generated by source and scattered field, generated by scattering on the modeled device. The advantage of this approach is, that we can subtract the source field, before it hits the edge of the grid and so the load on PML is reduced. The source also behaves as set of one dimensional waves and so there is no divergence of the beam.

However, if we want to use wave with with oblique wavevector, TFTS method is no longer usable. For this reason, there is a new function in the function library provided, that computes the source field on three edges of the grid before the modeled device. Reason, why there are three slices is, because the wave added diverges at it's edges and so correcting values has to be computed and included. Demonstration of this function is shown on picture 4.8. On the picture, the plain wave is generated on the left edges of the left half-plain, before the dielectric slab. Irregularities of the wavefront on the edges are caused by absorption of the wave in the PML. Once the wave passes the dielectric slab, which works as a new source for the right half-plain, there are no correction walues of the field on the

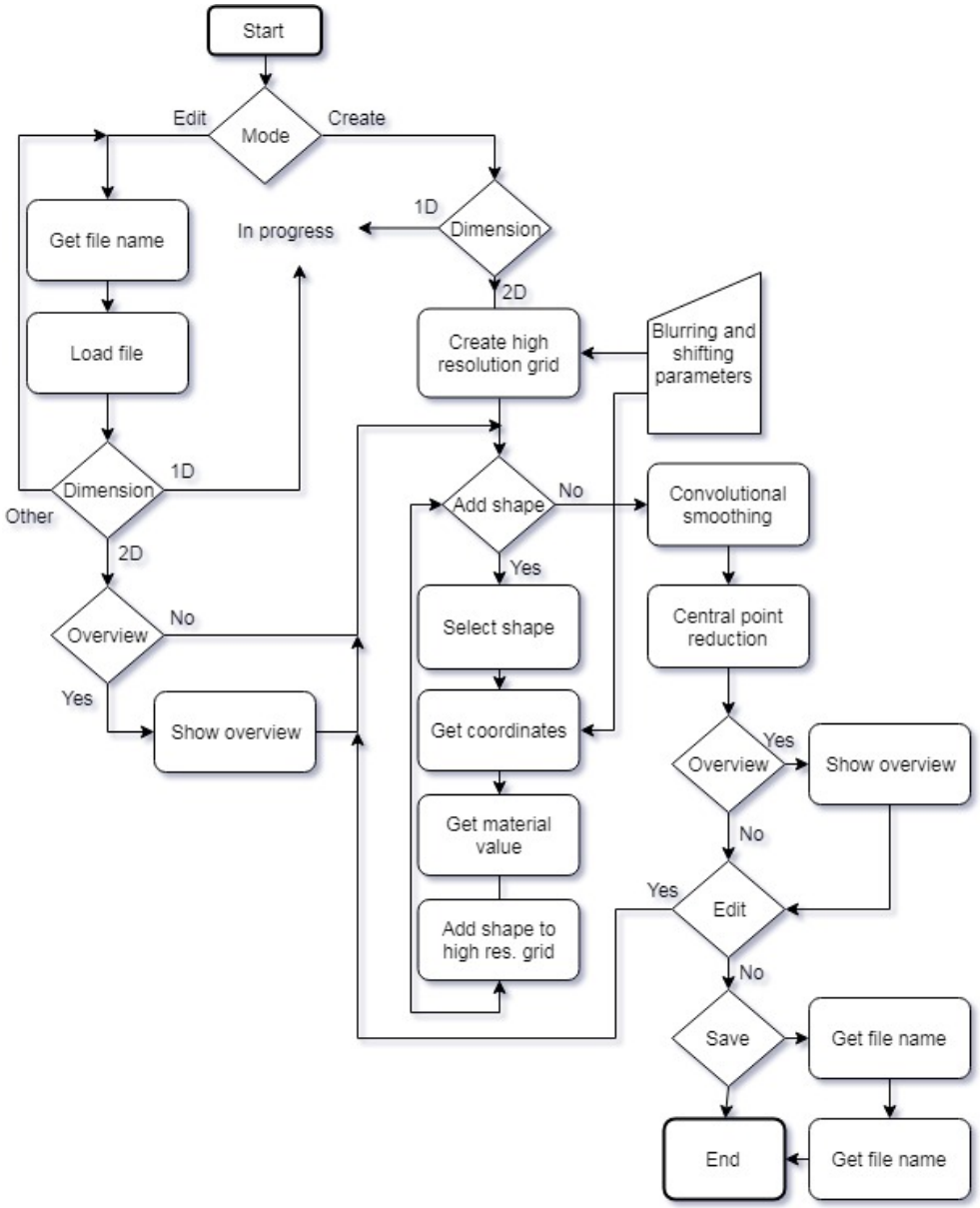
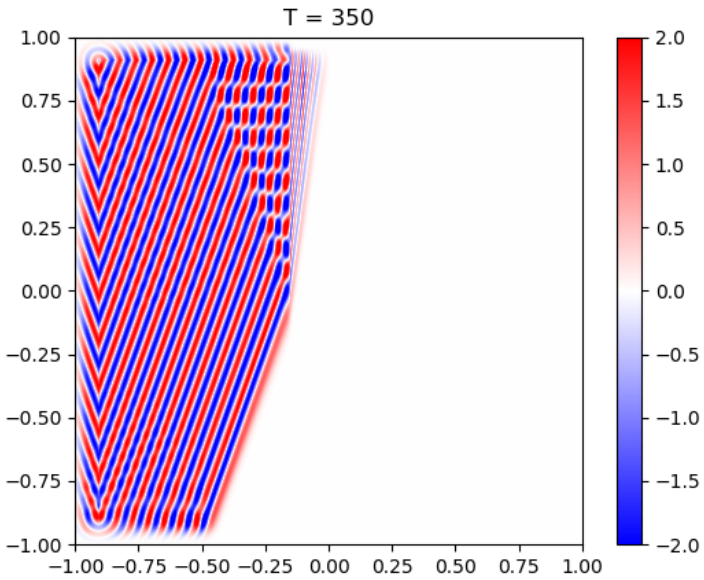
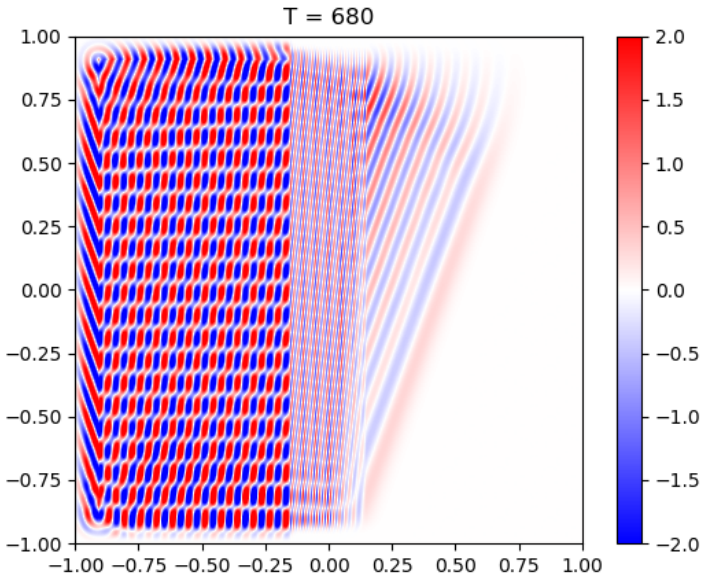


Figure 4.7: Flowchart of material Material making program

edge and so the wavefront is being deformed. This effect is clearly vissible on the pictue (b).



(a) Simulation after 350 time steps



(b) Simulation after 680 time steps.

Figure 4.8: Harmonical plain wave hitting dielectric slab of relative permittivity of 5

5. Post processing

While the algorithm we produced so far computes the electric and magnetic field vectors at every point of the grid, the raw information we obtain this way is not that much useful itself. To address that, the aim of this chapter is to provide some methods, how to extract the information we need from the result of our simulation.

5.1 Fourier Transform

The first information, that could be obtained, is the frequency respond of the simulated object by using the Fourier transform. If we were interested in broad band of frequency, we could simply illuminate our object by a monochromatic plane wave, compute the Fourier transform, make the same simulation without the object and then divide the results. This would have to be repeated for every single frequency, which would be impractical. Using the signal theory however, we could use a pulse, which contains every frequency, run our FDTD algorithm and store the time-domain result in every point of interest. Gaussian pulse is a suitable candidate. Then we could calculate the *Fourier transform* in our points of interest so that we know the amplitude of the E field, that would result from illuminating the object by each single wavelength. In linear materials, we can calculate the *Fourier transform* of our source to calculate the transmitance and reflectance for lossless objects.

In case, when we want to compute the *Fourier transform* only in few discrete positions, we can simply use standard Fourier transform, but in case, when we want to compute it on larger area, the memory requirement could become a problem. Fortunately, we can implement the computation into our main loop, where it takes only two additional scalar buffers for every point of interest.

The standard *Fourier Transform* can be written in form:

$$E(f_k,\, \boldsymbol{x}_{int}) = \int_0^{t_{max}} E(t,\, \boldsymbol{x}_{int}) e^{-2\pi i f_k t} dt \tag{5.1}$$

where the $E(f_k,\, \boldsymbol{x}_{int})$ is the amplitude of the harmonic wave of frequency f_k, calculated in the point of interest $\boldsymbol{x}_{int}$, t_{max} is the time length of the calculation. Notice, that the integration starts at 0 and ends at t_{max}. It is because FDTD assume, that there are no incident fields other then these that we described by our sources. We also assume, that the steady state occurs in a finite time t_{max}.

In a finite difference domain, this equation can be written as:

$$\begin{aligned} E(f_k,\, \boldsymbol{x}_{int}) &= \sum_{n=0}^{t_{max}} E(n\Delta t,\, \boldsymbol{x}_{int}) e^{-2\pi i f_k n\Delta t} \\ &= \sum_{n=0}^{t_{max}} E(n\Delta t,\, \boldsymbol{x}_{int}) \cos(2\pi_k n\Delta t) \\ &- i \sum_{n=0}^{t_{max}} E(n\Delta t,\, \boldsymbol{x}_{int}) \sin(2\pi_k n\Delta t) \end{aligned} \tag{5.2}$$

These equations can be solved in the post-processing procedure or directly in the main FDTD loop. To solve them inside the main loop, we have to split the sum into partial summations of each individual time step:

```
re[f,x]=re[f,x]+ez[x]*math.cos(2*math.pi*mf.freq(f)*dt*n)
im[f,x]=im[f,x]+ez[x]*math.sin(2*math.pi*mf.freq(f)*dt*n)
```

Parameter f determines the frequency, x chooses the position and n counts time steps.

Once these values are computed, the amplitude and phase can be calculated as:

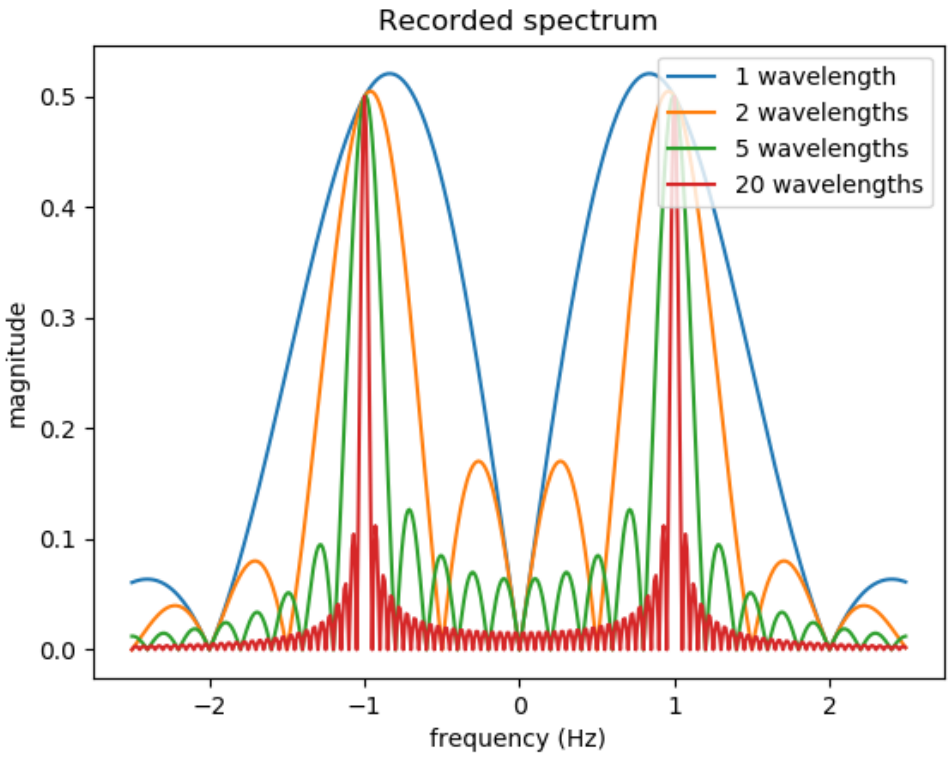


Figure 5.1: Demonstartion of spectral convergence with length of the simulation.

The picture 5.1 shows, how does the recorded spectrum converge, when the length of the simulation is increased. The source used is a sinusoidal waveform with frequency equal to 1. The reason, why we observe two peaks, while using only one wavelength as a source, is that the Fourier transformation of a real function is a symetric function and the electric field, we use for computing the fourier transform is a real function.

5.2 Near-field to Far-field Transformation

So far, we developed mechanism to calculate electromagnetic field, within finite space. The problem is, that this grid method allows us to compute electromagnetic fields only in small space in reasonable time. However we often need to know the radiation pattern further from the scattering object, where is our grid located.

In this subchapter, we are going to solve this problem by introducing the *Near-field to Far-field Transformation* which computes the far electromagnetic fields from near electromagnetic date, computed by our FDTD algorithm.

The first thing we need to define, is what does, and what does not, counts as a far field. A good approximation to use is the following:

$$\frac{2\pi R}{\lambda} \gg 1 \tag{5.3}$$

R stands for distance of the observed point of interest, where we want to compute the Far-field and λ is the wavelength used.

The key part of *Near-field to Far-field Transformation* is the *Surface Equivalence Theorem*

5.2.1 Surface Equivalence Theorem

Surface Equivalence Theorem is a theorem, by which real sources of electromagnetic waves, are replaced by equivalent sources. It is said that a field in a free space, created by a source in closed region, is equivalent as if it was created by tangential currents on surface, making a border around the real source.

This means that if we have our source and scattering device in a free space, we can simply surround the scattering device and the source by imaginary surface, box or rectangular in two dimensions more specifically, calculate tangential currents on this surface and then calculate the field in our point of interest without increasing a size of our grid. ([9])

This Theorem was introduced by Schelkunoff in 1936 and it is said to be more rigorous formulation od Huygens's principle which state that "each point on a primary wavefront can be considered to be a new source of a secondary spherical wave and that a secondary wavefront can be considered as the envelope of these secondary spherical waves." ([9])

5.2.2 Surface Currents

Once we select our surface box, defined by two points (S_{1x}, S_{1y}) and (S_{2x}, S_{2y}), we can start computing surface currents:

$$\begin{aligned} \boldsymbol{j}_S &= \boldsymbol{n} \times \boldsymbol{H} \\ \boldsymbol{M}_S &= -\boldsymbol{n} \times \boldsymbol{E}, \end{aligned} \tag{5.4}$$

where $\boldsymbol{n}$ is the normal vector to the surface.

It is worth to remind that in our calculations, $\boldsymbol{E}$ and $\boldsymbol{H}$ are not calculated in same spots, but rather shifted and so in order to simplify further calculations, we compute all currents in the center of Yee cell by averaging currents in vertices of the cell.

5.2.3 Far-field expression

Since we expect our point of interest to be in a free space, we need to compute only one filed, because we can simply calculate the other from the first one:

$$\boldsymbol{H}_{far}(\boldsymbol{r},\, t) = \frac{\boldsymbol{e}_r \times \boldsymbol{E}_{far}(\boldsymbol{r},\, t)}{Z_0}, \tag{5.5}$$

Where $\boldsymbol{e}_r$ is a radial unity vector, $\boldsymbol{r}$ is the spatial position vector and Z_0 is the vacuum impedance.

The formula for Electric Far-field is derived in ([10]):

$$\begin{aligned}\boldsymbol{E}_{far}(\boldsymbol{\rho},\, t) = &- \frac{1}{2\pi\sqrt{2|\boldsymbol{\rho}|}} \int_{L_a} dl' \\ &\times \int_{-\infty}^{u-\frac{d_{dist}}{c}} \Big(\frac{\partial \boldsymbol{M}_s(\boldsymbol{\rho'},\, t')}{\partial t} \times \boldsymbol{e}_\rho + Z_0 \frac{\partial \boldsymbol{J}_s(\boldsymbol{\rho'},\, t')}{\partial t} \Big) \frac{dt'}{\sqrt{c(u-t') - d_{dist}}}\end{aligned} \tag{5.6}$$

$\boldsymbol{\rho}$ is a projection of $\boldsymbol{r}$ to plane ($z = 0$) and $\boldsymbol{r}$ is the position of the calculated far-field. $\boldsymbol{\rho'}$ is a projection of $\boldsymbol{r}$ to the same plane ($z = 0$), which is a position of the current on the imaginary surface. L_a is a contour of the transversal section of the imaginary surface and l' is the integrated variable along L_a.

6. Surface plasmons

Surface plasmons are by definiton quanta of surface-charge-density oscilation, however, a same term is commonly used for collective oscilations in the electron density at the surface of a metal. There is a condition for interface mode of surface plasmon, derived in [11]. It states, that if we have an interface between two media, one of which can be described by real dielectric function ε_1 and the other one by complex dielectric function ε_2, following condition has to be fulfilled:

$$\begin{aligned} \varepsilon_1(\omega) \cdot \varepsilon_2(\omega) < 0, \\ \varepsilon_1(\omega) + \varepsilon_2(\omega) < 0 \end{aligned} \tag{6.1}$$

6.1 Dielectric model of noble metals

The mechanism behind Surface Plasmons is hidden inside dielectric model of noble metals. Noble metals, such as gold, or silver, has strong negative real part of dielectric constant and positive imaginary part for optical frequencyes.

6.1.1 Drude–Sommerfeld theory

In order to describe dielectric function of metals, let's consider only the effect of free electrons. Writing down the equation of motion for free electrons with applied field on the right hand side, we obtain:

$$m_e \frac{\partial^2 \boldsymbol{r}}{\partial t^2} + m_e \xi \frac{\partial \boldsymbol{r}}{\partial t} = \boldsymbol{E}_0 e^{-i\omega t}, \tag{6.2}$$

where m_e is the effective mass of the electron, ξ is the damping term, which is proportional to Fermi velocity over the electron mean free path between scattering events. $\boldsymbol{E}_0$ is the amplitude of incident field and ω is it's angular frequency. There is no restoring force term, because the electrons we consider, are free electrons. This equation is solved in [11] as:

$$\varepsilon_{Drude}(\omega) = 1 - \frac{\omega_p^2}{\omega^2 + \xi^2} + i \frac{\xi \omega_p^2}{\omega(\omega^2 + \xi^2)}, \tag{6.3}$$

where

$$\omega_p = \sqrt{\frac{ne^2}{m_e \varepsilon_0}} \tag{6.4}$$

is the plasma frequency.

This model can describe metals quite accurately in infrared part of the spectrum, however, it differs from reality significantly, as we approach optical part of the spectrum. In order to get more accurate results, it has to be supplemented with bound electrons.

6.1.2 Interband transitions

At optical frequencies, photons has energy to promote electrons from the lower energy bands into conduction band. Bound electrons can be described by equation of motion:

$$m_{be}\frac{\partial^2 \boldsymbol{r}}{\partial t^2} + m_{be}\gamma\frac{\partial \boldsymbol{r}}{\partial t} + \alpha \boldsymbol{r} = e\boldsymbol{E}_0 e^{-i\omega t} \tag{6.5}$$

Here m_{be} is the effective mass of the bound electron, which can be different from the effective mass of the free electron due to the effect of periodic potential. γ is the damping constant and α is the spring constant of the potential felt by bound electrons. Once again, the solution an be found in [11]:

$$\varepsilon_{Interband}(\omega) = 1 + \frac{\tilde{\omega}_p^2(\omega_0^2 - \omega^2)}{(\omega_0^2 - \omega^2)^2 + \gamma^2\omega^2} + i\frac{\gamma\tilde{\omega}_p^2\omega}{(\omega_0^2 - \omega^2)^2 + \gamma^2\omega^2} \tag{6.6}$$

Visualising this function 6.6 on picture 6.1, we can clearly see resonant behavior of imaginary part and dispersive behavior of real part. This brings us closer to the actual permittivity of gold for optical wavelengths, however in order to increase the precision, we would have to include interband absorbtion, which is further discussed and modeled in [12], where the model they use, can describe the dielectric function from 450 nm into infrared.

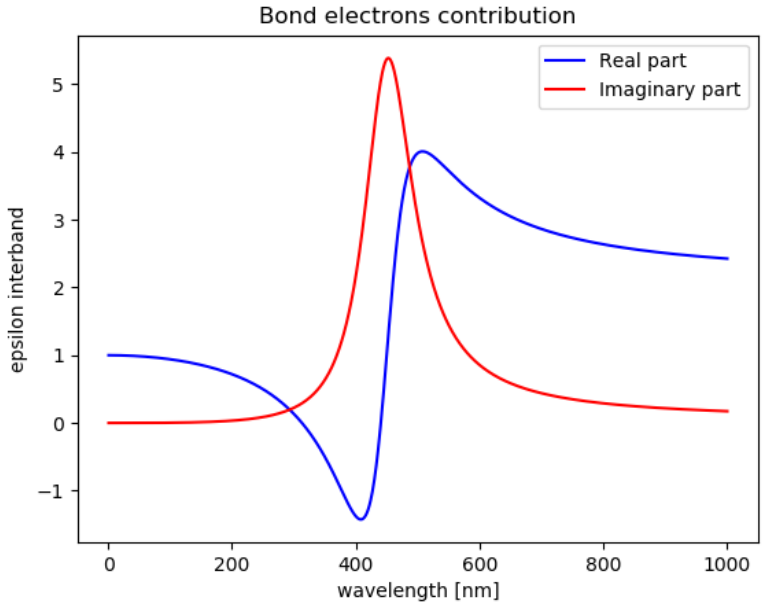


Figure 6.1: Permitivity contribution caused by bound electrons in gold.

At wavelenght 633 nm, we obtain $\varepsilon_2 = -11.6 + 1.2i$, which fulfills the condition 6.1, if we use as the other material some weak dielectricum, such as air.

6.2 Simulation of plasmon

Unfortunately, simulation of surface plasmon was not successfull, because an unknown source of numerical error occurred, which led to diverge of field amplitude

and supersaturation of the simulation. The simulation is displayed on picture 6.2.

Several attempts were made to fix this issue. The complex permittivity was split into real part $\varepsilon'(\omega)$ and imaginary part $\varepsilon''(\omega)$, which were interpreted as:

$$\varepsilon(\omega) = \varepsilon'(\omega) + i\varepsilon''(\omega) = \varepsilon'(\omega) + i\frac{\sigma(\omega)}{\omega}, \tag{6.7}$$

where σ is conductivity. Other approach was used by avoiding splitting permittivity into real and imaginary parts compleatl by moving the whole simulation to complex numbers, but that was not helpful either with similar result as displayed on picture 6.2.

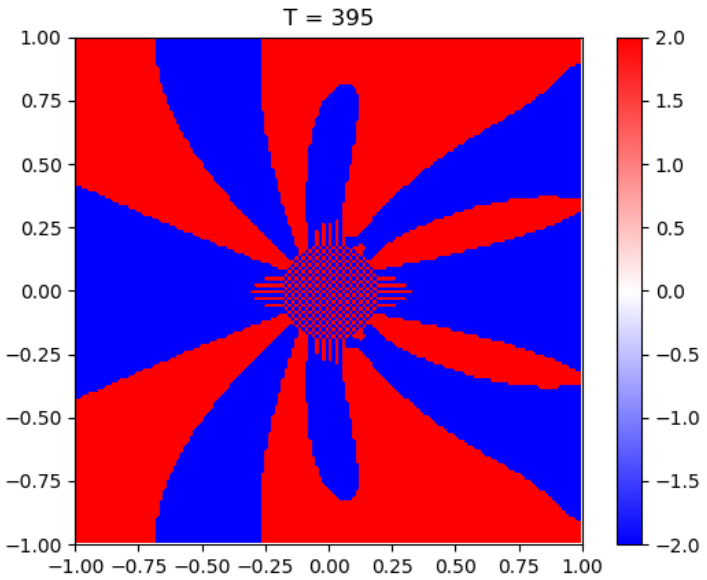


Figure 6.2: Supersaturated simulation of surface plasmon on thnin golden wire excited by sinusoidal plain wave. Units are normalized.

However, FDTD is effective an widely used in plasmonics. An example simulations can be found in [13] or [14]. There is mentioned in [13], that on of the problems could be, that avaragig the permittivity at the interface in order to model the shape of device more acuratly, could cause problems, because surface plasmons are very sensitive on values of permittivity used and smoothing the edge means to blur the interface and so the theory, described in [11] can no longer be applyed. To solve this, there is an alternative approach introduced, called *Effective Permittivity*.

Conclusion

At the beginning of this thesis, we started from the differential form of Maxwell's equations. We added Constitutive relations and expanded curl vector equations into vector component equations. The problem was then reduced into two dimensions which led to docoupling the system of equations to two separate subsystems. Applying the finite difference approximations to spatial and time derivations, together with half cell shifts, we obtained evolution equations. We then established relation between spatial step and time step, to insure computation stability and rearranged terms to final form of evolution equations.

Since this algorithm works only on finite grid and some waves eventually hits the edge of the grid, we introduced several versions of extra layers of artificial material around the grid to ensure that the wave die out, before hitting the edge. The first introduced method was PML, which is relatively simple to implement and light on computation demands, but can not deal properly with evanescent waves and works only if whole simulated object is in the Yee grid. To improve this, we introduced more sophisticated method, called CPML, which deals with these flaws of previous method, but is much more demanding on computation resources. Since there are no simulations, that would require use of CPML, we sticked with lighter version - PML for the rest of this thesis. For periodic structures, we briefly mentioned an existence of periodic boundary conditions, that are used in special cases.

In the following chapter, we examined evolution equations even further and replaced often repeating summations of neighboring elements with convolution operator so that each filed in a given time step can be processed as a single variable and so computation performance increased. In early versions of the code, scipy.signal.convolve2d was used to execute the convolution operation however, current versions use torch.nn.functional.conv2d which runs on GPU which is about ten to fifteen times faster. To move the computation to GPU, we had to move every variable, going into computation, to GPU and every multidimensional Numpy array had to be converted into torch tensor. The performance behavior of the computation for one specific GPU shown on graph 3.2 which tends to increase for larger grids.

Having the simulation optimized, we moved to the stability and precision discussion, where we derived, how many time steps we have to use and what grid density we have to use. Building of oblique and curved devices is being discussed with use of subcell averaging and bluring approximation. Combination of these two methods gives the best results. A simple script is provided, called Material_Create_2.2a.py to generate material data for simulation.

In the post processing section, we introduced Fourier transform, that can be implemented directly in the main loop to save memory requirements, or as a post processing, so that the core of the simulation is not slowed down. There is also a brief introduction to far field problematics and how to get the data from simulation results.

In the last chapter, we had a look at noble metal dielectric model and tried to simulate a surface plasmon, however this simulation was not successfull. The validity of the algorithm was tested by comparing result with examples in litera-

ture, especially in [2] and other then difference in values in PML area, caused by using different constants, the simulations were accurate.

Bibliography

[1] Veysel Demir Atef Elsherbeni. *The Finite-Difference Time-Domain Method*. First publish. Scitech Publishing, Inc., Releigh, 2008.

[2] Dennis M. Sullivan. *Electromagnetic simulation using the FDTD method*. First publish. IEEE Press, Piscataway, NJ 08855-1331 U.S.A., 2013.

[3] M. Kuzuoglu and R. Mittra. *Frequency dependance of the constitutive parameters of perfectly matched anisotropic absorbers*. vol. 6. IEEE Microwave and Guided Wave Letters, none, 1996.

[4] EE 5303 electromagnetic analysis using finite-difference time-domain. `http://emlab.utep.edu/ee5390fdtd.htm`. Accessed: 2019-03-26.

[5] Efficiently exploiting multiple cores with python. `https://python-notes.curiousefficiency.org/en/latest/python3/multicore_python.html`. Accessed: 2019-03-09.

[6] E. P. Cunningham. *Digital Filtration: An Introduction*. Princeton, 1992.

[7] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing*. Englewood Cliffs, 1975.

[8] Aliasing. `http://www.rctn.org/bruno/npb261/aliasing.pdf`. Accessed: 2019-05-5.

[9] Surface Equuivalence Theorem huygns's principle. `http://www.uniroma2.it/didattica/ap1/deposito/02_2-Balanis-Equivalence_Theorems.pdf`. Accessed: 2019-02-09.

[10] A Time-domain Near to far-field transformation for FDTD in 2D. `https://pdfs.semanticscholar.org/a613/5098e1770a3d1453d870bc5aff878d352bb3.pdf`. Accessed: 2019-02-11.

[11] Lukas Novotny and Bert Hecht. *Principles of nano-optics*. Cambridge university press, 2012.

[12] Peter B Johnson and R-W_ Christy. Optical constants of the noble metals. *Physical review B*, 6(12):4370, 1972.

[13] Naoki Okada and James B Cole. Effective permittivity for fdtd calculation of plasmonic materials. *Micromachines*, 3(1):168–179, 2012.

[14] Cristian Tira, Daniela Tira, Timea Simon, and Simion Astilean. Finite-difference time-domain (fdtd) design of gold nanoparticle chains with specific surface plasmon resonance. *Journal of Molecular Structure*, 1072:137–143, 2014.

List of Figures

1 An axample of the FDTD simulation. 3

1.1 Demonstration of reflection on boundary without using Absorbing boundary condition. The source is a gaussian pulse, centered in the middle of the grid. The grid represents a free space. 11

2.1 Demonstration of reflection on boundary with additional 10 layers of PML. The source is a sinusoidal pulse with profile $10\,sin(n/10)$, centered in the middle of the grid. The grid represents a free space. 16
2.2 Example of periodic boundary condition implementation. This picture is taken from [4] 20
2.3 Cylindrical wave with gausiian profile propgating trought periodic boundary condition 20

3.1 Flowchart of FDTD main loop with CPML implementation 21
3.2 Graph of the calculation speed, using different size of grid. Grid size already include PML layers. 26

4.1 Spectrum of the example source. This picture is taken from [4]. . 27
4.2 Generated field vs measured field. This picture is taken from [4]. . 28
4.3 Electric and magnetic field vector components spatial position in TM mode. This picture is taken from [4]. 29
4.4 Staircase effect, caused by modeling a device on finite orthogonal grid. This picture is taken from [4]. 29
4.5 Smoothing procedure, using the detailed grid technique and convolution. This picture is taken from [4]. 30
4.6 Demonstration of the difference between rawly constructed round object on the Yee grid versus an object, constructed on more delicate one that was smoothed by the described procedure. 31
4.7 Flowchart of material Material making program 32
4.8 Harmonical plain wave hitting dielectric slab of relative permittivity of 5 33

5.1 Demonstartion of spectral convergence with length of the simulation. 35

6.1 Permitivity contribution caused by bound electrons in gold. 39
6.2 Supersaturated simulation of surface plasmon on thnin golden wire excited by sinusoidal plain wave. Units are normalized. 40

List of Abbreviations

- FDTD - Finite difference time domain
- TM - Transversal electric
- TE - Transversal electric
- ABC - Absorbing boundary condition
- PML - Perfectly matching layer
- CPML - Complex Frequency-shifted PML
- CUDA - Compute unified device architecture
- GIL - Global interpreter lock
- GPU - Graphics processing unit
- CPU - Central processing unit
- NST - Nyquist sampling theorem
- PNG - Portable network graphics
- TFSF - Total field / Scattered field