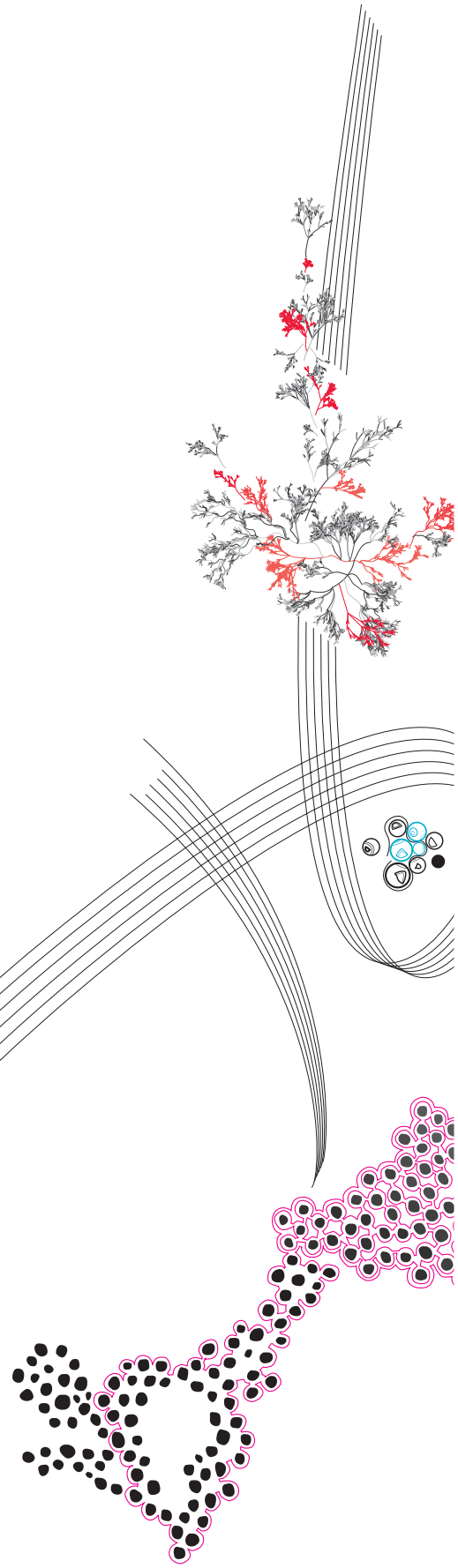

BSc Thesis Applied Mathematics and Applied Physics

Gaussian beam excitation of optical fibers using the Fictitious Domain Method

H.A. Hof

Supervisors: prof.dr. P.W.H. Pinkse, dr. M. Schlottbom

June, 2020

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

Department of Applied Physics
Faculty of Science and Technology

UNIVERSITY OF TWENTE.

Preface

This Bachelor thesis marks the end of my double bachelor studies in Applied Mathematics and Applied Physics. For the last ten weeks I put my blood, sweat and tears in this project, and I couldn't have done it alone.

First of all, I would like to express my deepest appreciation to my supervisors Pepijn Pinkse and Mathias Schlottbom for creating the opportunity for this research.

Secondly, I would like to thank the people from COPS I worked with, for the fruitful and very interesting meetings every Monday morning. A special thanks is in place for Matthijs Velsink who gave me a lot of advice and was able to read through my MATLAB code.

Last of all, I would like to extend my sincere thanks to Floor Venderbosch for proof-reading this paper and being very supportive of me during the last ten weeks.

Gaussian beam excitation of optical fibers using the Fictitious Domain Method

H.A. Hof*

June, 2020

Abstract

Excitation of a square optical fiber with rounded edges by a Gaussian beam is modelled numerically. The numerical simulation is based on the Fictitious Domain Method (FDM). The derivation of the two-dimensional Fictitious Domain Method is thoroughly studied and is then extended to three dimensional perfect prisms. Stability analysis is done on the two-dimensional Fictitious Domain Method. The Gaussian beams source is implemented using a modified version of a Total Field/Scattered Field formulation. A convergence study is done for a two dimensional trivial solution on a rotated square domain.

Keywords: Fictitious Domain Method, Finite Difference Time Difference, Gaussian beam, TF/SF, Optical fiber

1 Introduction

Optical fibers are thin waveguides, consisting of a core surrounded by a cladding layer with a higher refractive index. Light is transmitted inside the core by means of total internal reflection and propagates according to Maxwell's equations. When a beam of coherent light, e.g. a Guassian shaped pulse of light from a laser, is used to excitate an optical fiber, the coherent wavefronts propagating through the fiber interfere. The interference of these wavefronts is observed as a speckle pattern on the output image. For optical fibers with exact rectangular (or circular) cores, analytical solutions for their eigenmodes exist, and consequently the output speckle pattern can be obtained. [10] For more complex core shapes analytical solutions do not exist and numerical methods are used to simulate the wave propagation. These methods can be used to observe the effect of other core shapes on the output image. For example, the effect of the rounded corners of a square core (see Fig. 1) on the output speckle pattern can be simulated.

Simulations are based on the Finite Difference Time Difference (FDTD) method. This method, however, uses rectangular finite elements which need a very high resolution in order to approximate a curved domain. Resolving around non-trivial fiber shapes has been accomplished by using the Finite Element Method (FEM) [7, 8], but this method uses very small triangular elements, which lead to small time-steps and thus slower simulations. An interesting alternative is the Fictitious Domain Method (FDM) which has proved to be a faster and more precise alternative for these kind of differential equations. [1, 3, 4, 5]. The Fictitious Domain Method resolves complex domains by extending the computational domain to an outer domain bounded by a new, rectangular boundary. The boundary

*Email: h.a.hof@student.utwente.nl

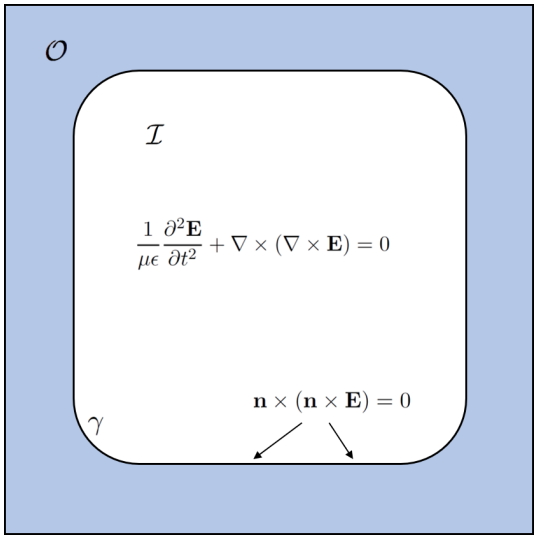

FIGURE 1: Geometry of a square optical fiber core with rounded corners. Light propagations is according to Maxwell's Equations for the electric field **E**. Blue region indicates cladding, white region indicates the fiber core.

condition between the core and the slabbing is taken into account by the introduction of a new variable, which is only defined at that boundary. The core idea is that the meshing of the computational domain can be done independently of the geometry of the fiber core, therefore allowing simple rectangular meshes.

In section 3 an implementation of the two-dimensional FDM will be developed and extended to three dimensional right prisms, as a simplification for the full three-dimensional FDM. Moreover, the Total Field/Scattered Field (TF/SF) formulation is introduced and modified for a Gaussian beam source.

In section 4 the stability of the scheme is investigated using the energy method. A continuous energy of Maxwell's equations is introduced. A discretized version of this quantity is shown to be conserved as well. From the discrete energy a stability condition relating the time-step size to the spatial step is derived.

In section 5 numerical results are presented. First, the stability condition that was derived, is used to calculate the time-step size by means of the power method. This stability condition is shown to converge towards the usual CFL condition. Thereafter, a trivial solution of Maxwell's equations is fabricated on a two dimensional square grid. The grid is then rotated and simulated by the FDM. An error is defined and the FDM is shown to converge. A test is performed to combine the TF/SF formulation and the FDM for scattering of a two-dimensional circle. Finally, Gaussian beam excitation of an optical fiber is modelled for two different fiber cores.

2 Theoretical background

2.1 Maxwell's equations

Maxwell's equations are a system of coupled equations that describe the behaviour of electric and magnetic fields. The system shows the relation between four field vectors: the electric field $\mathbf{E} = [E_x, E_y, E_z]$, and the magnetic field **H**, the electric displacement **D** and the magnetic induction **B**, likewise. Taking ρ as the charge density and **J** as the current

density in the medium, the fields are related as follows: [11]

$$\begin{aligned} \nabla \times \mathbf{E} + \frac{\partial \mathbf{B}}{\partial t} &= 0 & \nabla \cdot \mathbf{D} &= \rho. \\ \nabla \times \mathbf{H} - \frac{\partial \mathbf{D}}{\partial t} &= \mathbf{J} & \nabla \cdot \mathbf{B} &= 0. \end{aligned} \tag{1}$$

The set of equations is simplified by assuming that the medium is isotropic and linear, meaning the electric field has a linear relation with the electric displacement and the magnetic field has a linear relation with the magnetic induction. By standard conventions this is denoted as: $\mathbf{D} = \epsilon\mathbf{E}$ and $\mathbf{H} = \frac{1}{\mu}\mathbf{B}$. In an isotropic medium ϵ and μ are uniform in all directions and are called permittivity and permeability of the material respectively. We assume the core of the fiber has a constant permittivity and permeability. Further simplifications are achieved using the assumption that the medium is without charges and currents $\rho = 0$, $\mathbf{J} = 0$. After these simplifications the following equations arise:

$$\nabla \times \mathbf{E} + \mu\frac{\partial \mathbf{H}}{\partial t} = 0 \tag{2}$$

$$\nabla \cdot \mathbf{E} = 0 \tag{3}$$

$$\nabla \times \mathbf{H} - \epsilon\frac{\partial \mathbf{E}}{\partial t} = 0 \tag{4}$$

$$\nabla \cdot \mathbf{H} = 0. \tag{5}$$

In order to obtain a second order differential equation for the electric field alone, we take the time derivative of (4) and substitute using (2), which yields

$$\epsilon\frac{\partial^2 \mathbf{E}}{\partial t^2} = \frac{\partial}{\partial t}(\nabla \times \mathbf{H}) = \frac{1}{\mu}\nabla \times (\nabla \times \mathbf{E}). \tag{6}$$

This second order differential equation will be used in section 3.2, where the Fictitious Domain Method will be applied to it. The domain of these equations is made finite by introducing absorbing boundary conditions at an outer boundary.

2.2 Gaussian beams

We will now introduce Gaussian beams. In section 3.3.1 an input Gaussian beam source is obtained by modifying the TF/SF formulation and using the shape of the Gaussian beam at its focus. In order to find a description of a Gaussian beam we use the curl curl identity $\nabla \times (\nabla \times \mathbf{F}) = \nabla(\nabla \cdot \mathbf{F}) - \nabla^2\mathbf{F}$ and (3), to rewrite (6) as

$$\frac{1}{\mu}\nabla \times (\nabla \times \mathbf{E}) = \frac{1}{\mu}(0 - \nabla^2\mathbf{E}) \tag{7}$$

and thus

$$\frac{\partial^2 \mathbf{E}}{\partial t^2} + \frac{1}{\mu\epsilon}\nabla^2\mathbf{E} = 0. \tag{8}$$

Here ∇^2 denotes the Laplacian and (8) is just the wave equation. To find an equation for the propagation of a Gaussian beam, monochromatic, uniformly polarized waves will be considered under the scalar approximation. Taking ω as the angular frequency of the

wave, these waves are of the form

$$\mathbf{E}(x,y,z,t) = \tilde{\mathbf{E}}(x,y,z)\mathrm{e}^{\mathrm{i}\omega t}. \tag{9}$$

Substituting (9) in (8), and introducing the wavenumber k, as $k^2 = \frac{\omega^2}{\mu\epsilon}$, gives

$$(\nabla^2 + k^2)\tilde{\mathbf{E}}(x,y,z) = 0, \tag{10}$$

which is called the Helmholtz equation. In [12] it is shown that under the paraxial approximation, i.e. it is assumed that the wave traverses with a small angle to the z-axis, this equation has eigensolutions that keep their functional form as they propagate. The lowest order eigensolutions are called Gaussian beams. A Gaussian beam with beam waist radius w_0, can be described by

$$\mathbf{E}(x,y,z) = \mathbf{E}_0 \frac{w_0}{w} \mathrm{e}^{-\frac{x^2+y^2}{w^2}} \mathrm{e}^{\mathrm{i}(kz+\phi(z)-k\frac{x^2+y^2}{2R})}, \tag{11}$$

where, with the substitution of the Rayleigh range $z_R = \frac{kw_0^2}{2}$ in the newly introduced beam radius $w(z)$, radius of curvature $R(z)$ and phase correction $\phi(z)$, the following relations hold

$$w^2(z) = w_0^2(1 + (\frac{z}{z_R})^2), \qquad R(z) = z(1 + (\frac{z_R}{z})^2), \qquad \phi(z) = \arctan(\frac{z}{z_R}). \tag{12}$$

It should be noted that the shape of a Gaussian beam is determined by the beam waist radius and the wavenumber alone.

3 Numerical scheme

The electric field in an optical fiber behaves according to (6) in combination with a boundary constraint on the boundary of the fiber. The boundary constraint ensures the continuity of the tangential trace of the electric field at the boundary. Assuming an infinite refractive index outside the fiber core, and thus total internal reflection, this boundary constraint simply means that the parallel electric field at the boundary is zero, or $\mathbf{E}_{||} = 0$. We introduce the normal vector pointing outwards at the boundary, $\mathbf{n} = [n_x, n_y, n_z]$ such that the boundary constraint can be written as

$$\mathbf{n} \times (\mathbf{n} \times \mathbf{E}) = 0. \tag{13}$$

In this section, first the Yee scheme, a finite difference method used to simulate electromagnetic problems, will be introduced. The general framework of this method is then extended to the Fictitious Domain Method, which will be thoroughly examined. Lastly, the Total Field/Scattered Field formulation is introduced and modified to accomplish a Gaussian beam source

3.1 Yee scheme

In order to introduce the Yee scheme we discretize the first order Maxwell's equations (2) and (4). Note that equations (2) and 4) are equivalent to (6)), but lend themselves better for the introduction of the Yee scheme. We introduce the computational domain $\mathcal{C}$, which is bounded by an outer boundary $\partial\mathcal{C}$ where (13) holds. This domain is discretized using grid points $(x, y, z) = (n_x\Delta x, n_y\Delta y, n_z\Delta z)$. A staggered grid is introduced as in Fig. 2, where the discretized $\mathbf{E}$ field vectors appear on the edges of an element cube, the $\mathbf{H}$ field vectors appear in the center of the faces, and the nodes of the cube are the grid points $(n_x\Delta x, n_y\Delta y, n_z\Delta z)$.[13]

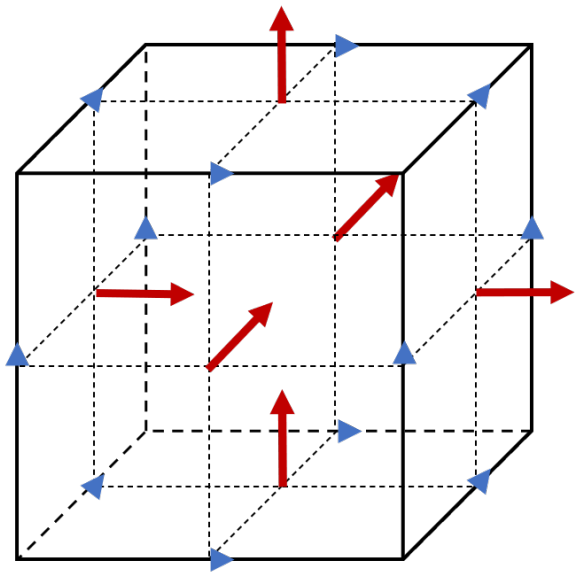

FIGURE 2: Yee cell, a finite element with a staggered grid. Blue triangles indicate direction and location of the vectors for the electric field, red arrow indicate direction and location of the vectors for the magnetic field.

Clearly, this discretization only works for brick-like domains $\mathcal{C}$. We introduce the time t, the time difference Δt and the time step l, such that $t = l\Delta t$. Moreover, we denote a discrete field F at time step l as $\mathbf{F}^l(n_x, n_y, n_z) = \mathbf{F}(n_x\Delta x, n_y\Delta y, n_z\Delta z)$. The staggered grid enables (2) to be discretized as

$$\frac{H_x^{l+\frac{1}{2}}(n_x, n_y+\frac{1}{2}, n_z+\frac{1}{2}) - H_x^{l-\frac{1}{2}}(n_x, n_y+\frac{1}{2}, n_z+\frac{1}{2})}{\Delta t} = \frac{1}{\mu}\left(\frac{E_y^l(n_x, n_y+\frac{1}{2}, n_z+1) - E_y^l(n_x, n_y+\frac{1}{2}, n_z)}{\Delta z} - \frac{E_z^l(n_x, n_y+1, n_z+\frac{1}{2}) - E_z^l(n_x, n_y, n_z+\frac{1}{2})}{\Delta y}\right),$$

$$\frac{H_y^{l+\frac{1}{2}}(n_x+\frac{1}{2}, n_y, n_z+\frac{1}{2}) - H_y^{l-\frac{1}{2}}(n_x+\frac{1}{2}, n_y, n_z+\frac{1}{2})}{\Delta t} = \frac{1}{\mu}\left(\frac{E_z^l(n_x+1, n_y, n_z+\frac{1}{2}) - E_z^l(n_x, n_y, n_z+\frac{1}{2})}{\Delta x} - \frac{E_x^l(n_x+\frac{1}{2}, n_y, n_z+1) - E_x^l(n_x+\frac{1}{2}, n_y, n_z)}{\Delta z}\right),$$

$$\frac{H_z^{l+\frac{1}{2}}(n_x+\frac{1}{2}, n_y+\frac{1}{2}, n_z) - H_z^{l-\frac{1}{2}}(n_x+\frac{1}{2}, n_y+\frac{1}{2}, n_z)}{\Delta t} = \frac{1}{\mu}\left(\frac{E_x^l(n_x+\frac{1}{2}, n_y+1, n_z) - E_x^l(n_x+\frac{1}{2}, n_y, n_z)}{\Delta y} - \frac{E_y^l(n_x+1, n_y+\frac{1}{2}, n_z) - E_y^l(n_x, n_y+\frac{1}{2}, n_z)}{\Delta x}\right).$$

Equation (4) can be similarly discretized at time step $l+\frac{1}{2}$. Clearly, these equations can be made explicit time-stepping, multiplying by Δt and rearranging the terms. This explicit scheme can be written as

$$\begin{aligned} H^{l+\frac{1}{2}} &= H^{l-\frac{1}{2}} - \frac{\Delta t}{\mu} R_h(E^l), \\ E^{l+1} &= E^l + \frac{\Delta t}{\epsilon}\ R_h^T(H^{l+\frac{1}{2}}), \end{aligned} \tag{14}$$

where the x, y and z coordinates are omitted by using column vectors E^l and H^l, the matrix R_h represents the discrete curl operator and R_h^T denotes its transpose. The Fictitious Domain Method introduced in the next section will turn out to be an extension of this general framework. For a MATLAB implementation of the two and three dimensional curl operators see Appendix A.1.1 and A.2.1, respectively.

3.2 Fictitious Domain Method

For the derivation of the Fictitious domain method, it is necessary to start at the second order form of the Maxwell's equations (6). An example of a problem that involves solving this equation subject to (13) on a non-rectangular domain is depicted in Fig. 1. The inner and outer regions are denoted $\mathcal{I}$ and $\mathcal{O}$, and meet on the boundary γ. Obviously, the Yee scheme cannot be applied easily to $\mathcal{I}$, because of the rounded edges of the domain.
The idea of the Fictitious Domain Method is to extend the computational domain to the outer region, and introduce a new computational domain $\mathcal{C}$, where $\mathcal{C} = \mathcal{I} \cup \mathcal{O}$. This way the outer boundary $\partial\mathcal{O}$, and thus $\partial\mathcal{C}$, can be freely chosen. The permeability and permittivity of the inner domain are also extended into $\mathcal{O}$ with the same constant value. The solution e of the new problem is a continuous and differentiable function on $\mathcal{C}$. To ensure consistency with the original problem, this function is enforced to be zero parallel to the boundary γ. This e can be found as the first argument of the solution (e, λ) of a variational problem, using test functions $\tilde{e} \in X$, a curl conforming function space on $\mathcal{C}$, and $\tilde{\lambda} \in L$, a function space on the boundary γ. For the full functional analytic setting we refer to [3]. The variational scheme is

$$\begin{aligned} \frac{\mathrm{d}^2}{\mathrm{d}t^2}(e, \tilde{e}) + \frac{1}{\mu\epsilon} a(e, \tilde{e}) &= b(\tilde{e}, \lambda) && \forall \tilde{e} \in X \\ b(e, \tilde{\lambda}) &= 0, && \forall \tilde{\lambda} \in L \end{aligned} \tag{15}$$

with the inner product and bilinear forms

$$\begin{aligned} (e, \tilde{e}) &= \int_{\mathcal{C}} e\tilde{e}\ \mathrm{d}x \\ a(e, \tilde{e}) &= \int_{\mathcal{C}} (\nabla \times e)(\nabla \times \tilde{e})\ \mathrm{d}x \\ b(e, \lambda) &= \int_{\gamma} \mathbf{n} \times (\mathbf{n} \times e) \cdot \lambda\ \mathrm{d}\gamma, \end{aligned} \tag{16}$$

3.2.1 Semi discretization in space

In order to find discrete approximations to (15), we introduce a piecewise linear polygonal approximation γ_h to γ. The space L is approximated by the discrete space L_h on γ_h.

Moreover, the discrete subspace $X_h \in X$, and the discrete functions e_h, $\tilde{e}_h$ and λ_h are considered. In principle, the meshes L_h and X_h can be chosen independently and do not need to overlap. The discrete variational version of (15) can now be formulated.

$$\begin{aligned} &\text{Find } (e_h, \lambda_h) \in X_h \times L_h \text{ such that for all } (\tilde{e}_h, \tilde{\lambda}_h) \in X_h \times L_h : \\ &\frac{\mathrm{d}^2}{\mathrm{d}t^2}(e_h, \tilde{e}_h) + \frac{1}{\mu\epsilon} a(e, \tilde{e}_h) = b(\tilde{e}_h, \lambda_h) \\ &\qquad\qquad b(e_h, \tilde{\lambda}_h) = 0. \end{aligned} \tag{17}$$

Because clearly all identities in (17) are linear or bilinear functions in e, $\tilde{e}$ and λ, they can be replaced by matrix operations on their respective basis representations in X_h and L_h. More precisely, let $\{\mathbf{v}_i, 1 \leq i \leq p = \dim(X_h)\}$ be a basis for X_h and $\{\mathbf{w}_i, 1 \leq i \leq q = \dim(L_h)\}$ be a basis for L_h. Now, for every e_h (respectively λ_h) there is a coordinate vector $\{E_{h,i}$, $i = 1, 2..p\}$ (respectively $\{\Lambda_{h,j}$, $j = 1, 2..q\}$) such that

$$\begin{aligned} e_h &= \sum_{i=1}^{p} E_{h,i} \mathbf{v}_i \\ \tilde{e}_h &= \sum_{i=1}^{p} \widetilde{E}_{h,i} \mathbf{v}_i \\ \lambda_h &= \sum_{i=1}^{q} \Lambda_{h,i} \mathbf{w}_i. \end{aligned} \tag{18}$$

Then, matrices M_h, A_h and B_h acting on the decompositions E_h, $\widetilde{E}_h$, λ_h can be constructed such that $(e_h, \tilde{e}_h) = \widetilde{E}_h^T M_h E_h$, $a(e_h, \tilde{e}_h) = \widetilde{E}_h^T A_h E_h$ and $b(e_h, \lambda_h) = \Lambda_h^T B_h E_h$. These matrices

$$\begin{aligned} M_h(i,j) &= \int_{\mathcal{C}} \mathbf{v}_j \cdot \mathbf{v}_i \, \mathrm{d}x \\ A_h(i,j) &= \int_{\mathcal{C}} (\nabla \times \mathbf{v}_j) \cdot (\nabla \times \mathbf{v}_i) \, \mathrm{d}x \\ B_h(i,j) &= \int_{\gamma} \mathbf{n} \times (\mathbf{n} \times \mathbf{v}_j) \cdot \mathbf{w}_i \, \mathrm{d}\gamma \end{aligned} \tag{19}$$

are determined by the choice of the basis functions $\mathbf{v}_i$ and $\mathbf{w}_i$. Indeed, because of the bilinearity in (16), these matrices together with the column vector decompositions can now form the basis representation version of (17)

$$\begin{aligned} \frac{\mathrm{d}^2}{\mathrm{d}t^2}(\widetilde{E}_h^T M_h E_h) + \frac{1}{\mu\epsilon} \widetilde{E}_h^T A_h E_h &= \Lambda_h^T B_h \widetilde{E}_h, && \forall \widetilde{E}_h \in \mathbb{R}^p, \\ \widetilde{\Lambda}_h^T B_h E_h &= 0, && \forall \widetilde{\Lambda}_h \in \mathbb{R}^q. \end{aligned} \tag{20}$$

Since choosing the basis functions $\{\mathbf{v}_i, i = 1, 2, ..., p\}$ and $\{\mathbf{w}_i, i = 1, 2, ...q\}$ obviously induces a bijection between X_h and $\mathbb{R}^p$, and L_h and $\mathbb{R}^q$, (20) is equivalent to (17). The equations (20) hold for all possible $\widetilde{E}_h$ and $\tilde{\lambda}_h$. Therefore, and by using the fact that M_h

and A_h are symmetric matrices by definition, it easily follows that

$$M_h \frac{\mathrm{d}^2 E_h}{\mathrm{d}t^2} + \frac{1}{\mu\epsilon} A_h E_t = B_h^T \Lambda_h$$
$$B_h E_h = 0. \tag{21}$$

The matrix M_h is called the mass matrix and A_h is called the stiffness matrix. This terminology is extended by referring to B_h as the boundary matrix.

3.2.2 Remarks on the mass and stiffness matrix

By making some assumptions on the basis vectors $\mathbf{v}_i$ and the meshing, some properties of the matrices M_h and A_h can be determined. From now on, it is assumed the grid is isotropic, meaning $\Delta x = \Delta y = \Delta z = h$ and the basis vectors are lowest order Nédelec edge elements. We introduce the same meshing of the rectangular grid as before, using gridpoints $(x, y, z) = (n_x h, n_y h, n_z h)$, which induce cubic grid cells. A grid cell is denoted K_r, and an edge of a grid cell is denoted $f_j \in F_h$, the set of all edges. Either an edge lies in the interior of the computational domain, denoted $f_j \in F_{\mathcal{C}}$, or at the boundary of the computational domain, $f_j \in F_{d\mathcal{C}}$ and thus $F_h = F_{\mathcal{C}} \cup F_{d\mathcal{C}}$. Moreover, we denote the tangent to edge f_j by $\boldsymbol{\tau}_j$. The lowest order Nédelec edge elements are associated with an edge f_j such that $\mathbf{v}_i \cdot \boldsymbol{\tau}_j = 1$ if $i = j$ and $\mathbf{v}_i \cdot \boldsymbol{\tau}_j = 0$ if $i \neq j$. Inside the grid cells, the basis functions $\mathbf{v}_i$ are linear, for example as shown in Fig. 3 for the two-dimensional Nédelec edge elements. These functions are translated and scaled per grid cell to obtain the global basis functions.

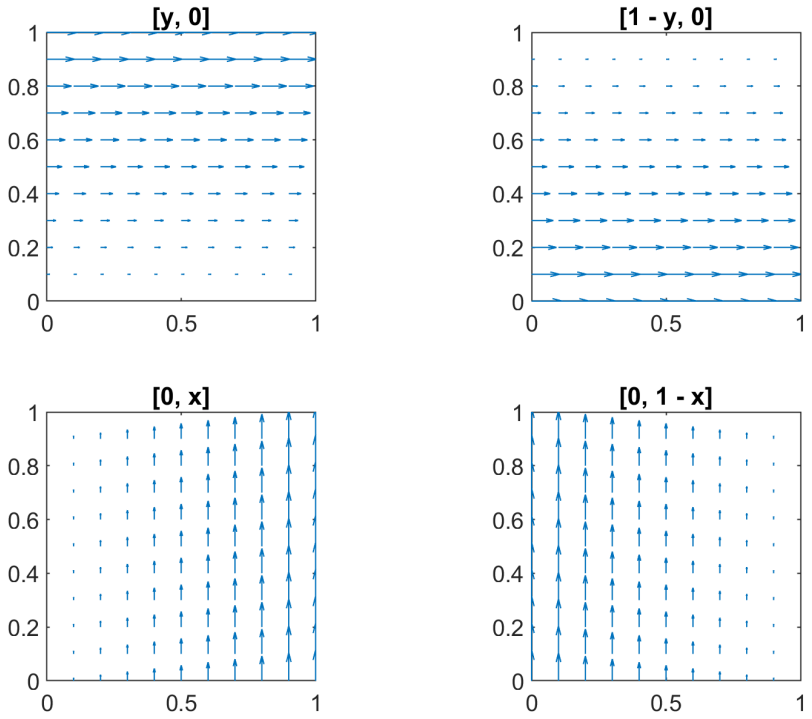

FIGURE 3: Lowest order Nédelec edge elements on a two-dimensional unit grid cell.

It is important to remark that basis functions $\mathbf{v_i}$ are nonzero in multiple grid cells. In a two dimensional grid interior edges have two adjacent grid cells, and in a three dimensional grid four adjacent grid cells. Moreover, in a three dimensional grid, edges $f_j \in F_{d\mathcal{C}}$ can also have two adjacent grid cells. In particular, an edge f_j is an intersection between grid cells K_r, $r = 1, 2..$, i.e. $f_j = \bigcap_r K_r$. The support of $\mathbf{v}_j$ (the domain where the basis function is non-zero) is the union of those adjacent grid cells, and we write $\text{supp}(\mathbf{v}_j) = \bigcup_r K_r$. Now, from (19) it is easily shown that all columns (and rows) of M_h associated with a basis vector $\mathbf{v}_j$ associated to an edge $f_j \in F_{\mathcal{C}}$, add up to h^2. Using mass-lumping

$$M_h = h^2 I, \tag{22}$$

where I denotes the identity matrix.

A similar, somewhat more elaborate remark will now be made about the stiffness matrix. Again, from (19) it is clear that $A_h(i,j)$ is non-zero if $\text{supp}(\mathbf{v}_i) \cap \text{supp}(\mathbf{v}_j) \neq \emptyset$. In particular, $A_h(i,j) = 2$ if $i = j$, $A_h(i,j) = 1$ if $\mathbf{v}_i$ and $\mathbf{v}_j$ both contribute to the discrete curl in the z-direction with the same sign in the intersection of their support, and $A_h(i,j) = -1$ if $\mathbf{v}_i$ and $\mathbf{v}_j$ contribute to the discrete curl in the z-direction with a different sign in the intersection of their support. By direct calculation and from the definition of R_h,

$$A_h = h^2 R_h^T R_h. \tag{23}$$

Combining (21)-(23) gives

$$\begin{aligned} \frac{\mathrm{d}^2 E_h}{\mathrm{d}t^2} + \frac{1}{\mu\epsilon} R_h^T R_h E_h &= \frac{1}{h^2} B_h^T \Lambda_h \\ B_h E_h &= 0. \end{aligned} \tag{24}$$

3.2.3 Time discretization

In order to do a time discretization, the interval $[0, t_{end}]$ is introduced and divided into pieces using a time step Δt. Denoting by E_h^l the decomposition E_h at time-step l, the second order central finite difference is applied to (24) to obtain

$$\begin{aligned} \frac{E_h^{l+1} - 2E_h^l + E_h^{l-1}}{(\Delta t)^2} + \frac{1}{\mu\epsilon} R_h^T R_h E_h^l &= \frac{1}{h^2} B_h^T \Lambda_h^l \\ B_h E_h^l &= 0. \end{aligned} \tag{25}$$

To obtain an explicit first order time stepping scheme as in (14), the magnetic field has to be incorporated in the scheme. With this in mind, the same first order central finite difference is used to discretize the magnetic field in time

$$\frac{H_h^{l+\frac{1}{2}} - H_h^{l-\frac{1}{2}}}{\Delta t} = -\frac{1}{\mu} R_h(E_h^l) \tag{26}$$

and Λ_h^l is treated as the derivative of a new variable λ_h

$$\Lambda_h = \frac{\lambda_h^{l+\frac{1}{2}} - \lambda_h^{l-\frac{1}{2}}}{\Delta t}. \tag{27}$$

Substituting (26) and (27) into the first equation of (25) yields

$$\frac{1}{\Delta t}\left(\frac{E_h^{l+1}-E_h^l}{\Delta t}-\frac{E_h^l-E_h^{l-1}}{\Delta t}\right)-\frac{1}{\epsilon}R_h^T\left(\frac{H^{l+\frac{1}{2}}-H^{l-\frac{1}{2}}}{\Delta t}\right)=\frac{1}{h^2}B_h^T\left(\frac{\lambda_h^{l+\frac{1}{2}}-\lambda_h^{l-\frac{1}{2}}}{\Delta t}\right) \quad (28)$$

which, using a time translation symmetry argument, is equivalent to

$$\frac{E_h^{l+1}-E_h^l}{\Delta t}-\frac{1}{\epsilon}R_h^T H^{l+\frac{1}{2}}=\frac{1}{h^2}B_h^T\lambda_h^{l+\frac{1}{2}}. \quad (29)$$

Finally, (26) and (29) can be solved together with the constraint $B_h E_h^l = 0$ to obtain the promised scheme in the form of (14):

$$\begin{aligned} H_h^{l+\frac{1}{2}} &= H_h^{l-\frac{1}{2}}+\frac{\Delta t}{\mu}R_h(E_h^l), \\ E_h^{l+1} &= (1-B_h^T(B_hB_h^T)^{-1}B_h)(E_h^l+\frac{\Delta t}{\epsilon}\ R_h^T(H_h^{l+\frac{1}{2}})). \end{aligned} \quad (30)$$

Comparing this with the main result in [3] shows a difference by a factor Δt in the term that incorporates the boundary, whereas (30) is the correct result. It might be noted that B_h is not a square matrix because $p > q$ and consequently B_h does not have an inverse. If there is no boundary present $B_h = 0$ (the zero matrix) and (30) is just the Yee scheme (14).

3.2.4 Two-dimensional boundary matrix

The term $B_h^T(B_hB_h^T)^{-1}B_h$ in (30) is the correction term that takes the boundary condition into account. In this section the entries of the B_h matrix will be determined for a particular choice of basis elements $\mathbf{v}_i$ and $\mathbf{w}_j$. As has been stated in section 3.2.1 the meshes X_h and L_h can be chosen independently. For simplicity, the Maxwell's equations are solved on a two-dimensional grid only involving E_x, E_y and H_z, see Fig. 4. Later, the result will be extended to the full three-dimensional equations.

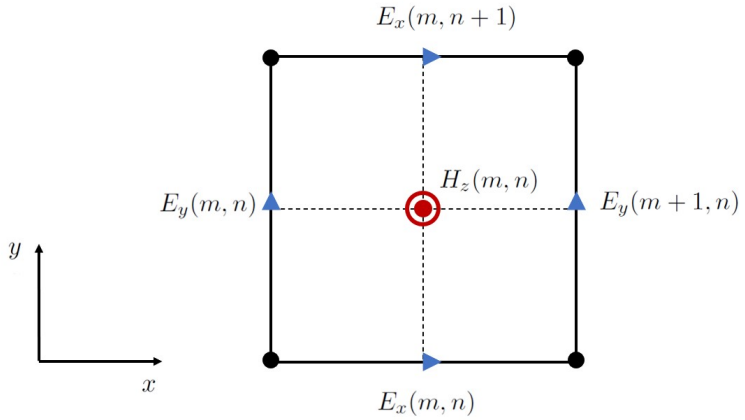

FIGURE 4: Two dimensional projection of the Yee cell

A method to find the matrix B_h for the two-dimensional FDM will now be provided, by finding the value of an arbitrary non-zero entry in the boundary matrix. The basis elements

for X_h were already determined to be the lowest order Nédelec edge elements (Fig. 3). The basis elements for L_h are chosen to be the P^1 (linear) elements. We introduce the points γ_i, $i = 1, 2, ...q$ on the polygonal approximation γ_h of the boundary γ, such that γ_h consists of line segments $[\gamma_i, \gamma_{i+1}]$. A line segment between γ_i and γ_{i+1} is paramatrized by

$$\sigma_i(\theta) = (\gamma_{i+1} - \gamma_i)\theta + \gamma_i, \qquad \theta \in [0, 1], \tag{31}$$

and has a tangent $\boldsymbol{\tau}_i$

$$\boldsymbol{\tau}_i = \frac{\gamma_{i+1} - \gamma_i}{\|\gamma_{i+1} - \gamma_i\|}. \tag{32}$$

Here $\|\gamma_{i+1} - \gamma_i\|$ denotes the Euclidean norm. Then, from these parametrizations the basis elements $\mathbf{w}_i$ are defined as

$$\begin{cases} \mathbf{w}_i(\sigma) = (1-\theta)\boldsymbol{\tau}_i & \text{if } \sigma = \sigma_i(\theta) \\ \mathbf{w}_i(\sigma) = \theta\boldsymbol{\tau}_{i-1} & \text{if } \sigma = \sigma_{i-1}(\theta) \\ \mathbf{w}_i(\sigma) = 0 & \text{otherwise.} \end{cases} \tag{33}$$

Now having determined the choice of $\mathbf{v}_j$ and $\mathbf{w}_i$, the only thing left to do is to evaluate the integral in (19) giving the entries of the matrix B_h. An example of two basis functions corresponding to a non-zero entry in B_h is shown in Fig. 5.

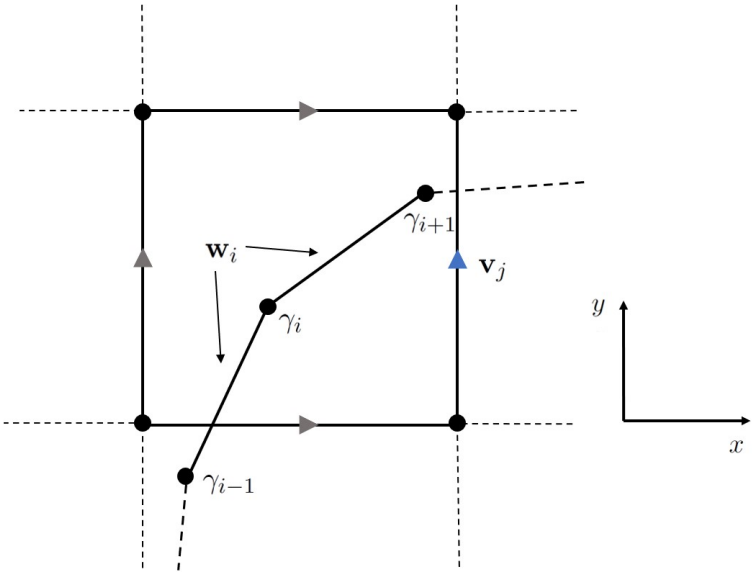

FIGURE 5: Example of two basis functions $\mathbf{v}_j$ and $\mathbf{w}_i$ contributing to a non-zero entry $B_h(i, j)$ in the boundary matrix

Every boundary segment, e.g $[\gamma_i, \gamma_{i+1}]$, can intersect with the boundary of a particular grid cell one time, intersect with the boundary of a cell twice, or lie completely in the grid cell. These intersections determine the intersections of the supports of the basis functions $\mathbf{v}_j$ and $\mathbf{w}_i$, and consequently the bounds of the integral for an entry of the boundary matrix. The first and the latter option are true here as seen in Fig. 5, however the method is equivalent for all cases. First, the intersections between the boundary and the mesh grid of $\mathcal{C}$ will be found. Second, the line will be parametrized to change the integration variable. Finally a Gauss Legendre quadrature is used to evaluate the integral.

Grid and boundary intersections
The intersections between the edges f_j and the line segments σ_i have to be found in a two dimensional domain. Due to notational convenience two grid points γ_h are here indicated by $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$. The same points can also uniquely be represented as

$$\begin{aligned} p_1 &= \left(\left\lfloor \frac{x_1}{h} \right\rfloor h + \tilde{x}_1, \left\lfloor \frac{y_1}{h} \right\rfloor h + \tilde{y}_1 \right) \\ p_2 &= \left(\left\lfloor \frac{x_2}{h} \right\rfloor h + \tilde{x}_2, \left\lfloor \frac{y_2}{h} \right\rfloor h + \tilde{y}_2 \right) \end{aligned} \tag{34}$$

where $\lfloor x \rfloor$ denotes the floor function. The line segment between these two points has intersections with edges f_j if the values of the floor functions for the x and y coordinates are not the same. In particular, the number of intersections with a vertical edge is $|\lfloor \frac{x_1}{h} \rfloor - \lfloor \frac{x_2}{h} \rfloor|$ and the number of intersections with a horizontal edge is $|\lfloor \frac{y_1}{h} \rfloor - \lfloor \frac{y_2}{h} \rfloor|$. The x coordinates of the intersections with a vertical edge are now found to be

$$x^{|} = \min\left(\left\lfloor \frac{x_1}{h} \right\rfloor, \left\lfloor \frac{x_2}{h} \right\rfloor \right) + kh \qquad\qquad 1 \le k \le |\left\lfloor \frac{x_1}{h} \right\rfloor - \left\lfloor \frac{x_2}{h} \right\rfloor| \tag{35}$$

and for the y coordinates of the intersections with a horizontal edge

$$y^{-} = \min\left(\left\lfloor \frac{y_1}{h} \right\rfloor, \left\lfloor \frac{y_2}{h} \right\rfloor \right) + kh \qquad\qquad 1 \le k \le |\left\lfloor \frac{y_1}{h} \right\rfloor - \left\lfloor \frac{y_2}{h} \right\rfloor|. \tag{36}$$

The corresponding x^{-} and $y^{|}$ values can easily be found from the line parametrization σ_i, substituting a known value from (35) or (36) and then solving for the unknown variable. Some caution should be taken to resolve the fact that not all points found are unique if a line intersects with a grid cell exactly at a grid point $(n_x h, n_y h, n_z h)$.

Line parametrization
A basis function $\mathbf{w}_i$ on a line segment parametrized by σ_i is simply equal to $\theta\boldsymbol{\tau}$ or $(1-\theta)\boldsymbol{\tau}$. The goal is to transform the integral over the boundary into an integral over θ using these parametrizations. The value of the basis functions $\mathbf{v}_j$ are evaluated as $\mathbf{v}_j(\sigma_i(\theta))$. The (x, y) coordinates of the line segments σ_i are then mapped to the unit square to find the value of $\mathbf{v}_j$.

Note that for two dimensional grids, where the tangent of a boundary segment is $\boldsymbol{\tau} = (\tau_1, \tau_2)$, the normal vector can be written as $\mathbf{n} = (\tau_2, -\tau_1)$ and consequently, using (33)

$$\mathbf{n} \times (\mathbf{n} \times \mathbf{v}_j) \cdot \mathbf{w_i} = \mathbf{v}_j \cdot \mathbf{w}_i. \tag{37}$$

Further note that, for a boundary segment σ_i from $p_1 = (x_1, y_1)$ to $p_2 = (x_2, y_2)$, the differential of the integration variable can be substituted with

$$\mathrm{d}\gamma = \sqrt{\mathrm{d}x^2 + \mathrm{d}y^2} = \sqrt{((x_2 - x_1)\mathrm{d}\theta)^2 + ((y_2 - y_1)\mathrm{d}\theta)^2} = \|p_2 - p_1\| \, \mathrm{d}\theta. \tag{38}$$

In conclusion, the integral for B_h can now be computed using a change of variables as

$$\begin{aligned} B_h(i,j) &= \int_\gamma \mathbf{n} \times (\mathbf{n} \times \mathbf{v}_j) \cdot \mathbf{w}_i \, \mathrm{d}\gamma = \int_{\gamma_{i-1}}^{\gamma_i} \mathbf{v}_j \cdot \mathbf{w}_i \, \mathrm{d}\gamma + \int_{\gamma_i}^{\gamma_{i+1}} \mathbf{v}_j \cdot \mathbf{w}_i \, \mathrm{d}\gamma \\ &= \|\gamma_{i-1} - \gamma_i\| \int_0^1 \mathbf{v}_j \left(\sigma_{i-1}(\theta)\right) \cdot \boldsymbol{\tau}_{i-1} \, \theta \, \mathrm{d}\theta + \|\gamma_i - \gamma_{i+1}\| \int_0^1 \mathbf{v}_j \left(\sigma_i(\theta)\right) \cdot \boldsymbol{\tau}_i (1-\theta) \, \mathrm{d}\theta \end{aligned}$$

Gauss Legendre quadrature

The intersections of the line segments with the grid edges, can be combined with the parametrizations σ_i to find α_1, β_1, α_2 and β_2, such that for example $\sigma_{i-1}(\theta) \subset \text{supp}(\mathbf{v}_j)$ for $\theta < \alpha_1$ and $\theta > \beta_1$, and $\sigma_{i-1}(\theta) \cap \text{supp}(\mathbf{v}_j) = \emptyset$ for $\theta > \alpha_1$ and $\theta < \beta_1$. The integral bounds can be changed to

$$B_h(i,j) = \|\gamma_{i-1} - \gamma_i\| \int_{\alpha_1}^{\beta_1} \mathbf{v}_j\left(\sigma_{i-1}(\theta)\right) \cdot \theta\boldsymbol{\tau} \, \mathrm{d}\theta + \|\gamma_i - \gamma_{i+1}\| \int_{\alpha_2}^{\beta_2} \mathbf{v}_j\left(\sigma_i(\theta)\right) \cdot (1-\theta)\boldsymbol{\tau} \, \mathrm{d}\theta \tag{39}$$

and the functions $\mathbf{v}_j$ in the integral are now continuous linear functions without a kink, i.e. the integral bounds are chosen such that $\mathbf{v}_j$ is only evaluated inside grid cells it is supported in. Note that possibly the integration is still over two cells $K_r \subset \text{supp}(\mathbf{v}_j)$. In this case, the integral should be split once more with the intersections found above using α_3 and so on. Thereafter, the functions $\mathbf{v}_j$ are linear functions in θ and so are $\theta\boldsymbol{\tau}$ and $(1-\theta)\boldsymbol{\tau}$, and consequently the functions inside the integral are quadratic functions in θ. Hence, these functions can be exactly evaluated using a second order quadrature, e.g. Gauss Legendre quadrature

$$\int_a^b f(t) \, \mathrm{d}t = \frac{b-a}{2} \sum_{n=1}^{2} f\left(\frac{b-a}{2}\xi_n + \frac{a+b}{2}\right), \qquad \xi_n = \pm\frac{1}{\sqrt{3}}. \tag{40}$$

By taking $f(t) = \mathbf{v}_j\left(\sigma_{i-1}(t)\right) \cdot t\boldsymbol{\tau}$ or $f(t) = \mathbf{v}_j\left(\sigma_i(t)\right) \cdot (1-t)\boldsymbol{\tau}$, the value of the entry of $B_h(i,j)$ is found. Repeating this method will yield the boundary matrix B_h. For a MATLAB implementation of this result we refer to the Appendix A.1.3.

3.2.5 Extension to three-dimensional right prisms

The method that was derived, does not work for three-dimensional problems. However, it will be argued that the boundary matrix that was found in section 3.2.4 can be used for right prisms. Right prisms are objects with a constant two-dimensional shape shifted in a third dimension, here the z-axis. An example can be seen in Fig. 6.

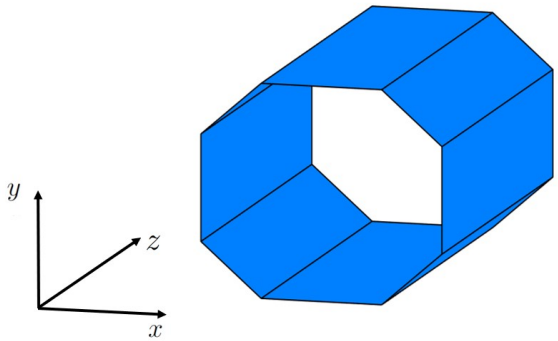

FIGURE 6: Example of a right prism

A discretization of the surface of such a right prism is $S_h = \{(x, y, z)|(x, y) \in \gamma_h, z = kh, k = 0, 1, ..\}$, where γ_h is a polygonal approximation to the shape in the xy-plane as before. The normals on the faces of S_h are parallel to the xy-plane and hence, the tangential fields at the boundary can be decomposed uniquely as the sum of a component in the xy-plane, and a component in the z-direction. Now, the constraint at the boundary can be decoupled for the E_x and E_y-components on the one hand, and the E_z-component on the other hand. The integrals for B_h as in (19) over the boundary will be surface integrals. The basis functions for $\mathbf{v}_j$ are now the three-dimensional lowest order Nédelec elements. These three dimensional basis functions $\mathbf{v}_j$ are still associated with an edge f_j as before. We denote by $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ the unit vectors in the directions of E_x, E_y and E_z. In the unit cube, there are four edges with an associated basis function in the direction of E_x

$$\begin{aligned} \mathbf{v}_1 &= (1-y)(1-z)\ \hat{\mathbf{x}}, \\ \mathbf{v}_2 &= (1-y)z\ \hat{\mathbf{x}}, \\ \mathbf{v}_3 &= y(1-z)\ \hat{\mathbf{x}}, \\ \mathbf{v}_4 &= yz\ \hat{\mathbf{x}}, \end{aligned} \tag{41}$$

and similarly, four edges associated with basis functions in the direction of both E_y and E_z. These basis functions are then scaled and translated to the edge f_j they are associated with. Every basis function $\mathbf{v}_j$ has a direction $\hat{\mathbf{v}}_j \in \{\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{x}}\}$ and can be decomposed as $\mathbf{v}_j = v_j(x, y)v_j(z)\hat{\mathbf{v}}_j$. Basis functions $\mathbf{w}_i$ on the boundary are either chosen tangent to the curve γ_h in the xy plane or parallel to the z axis. We write $\mathbf{w}_i = w_i(x, y)w_i(z)\hat{\mathbf{w}}$, $\hat{\mathbf{w}} \in \{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2..., \boldsymbol{\tau}_q, \hat{\mathbf{z}}\}$. It is important to note that for every basis function $\mathbf{v}_j$ (respectively $\mathbf{w}_i$) with the same z location of the cells it has support in, $v_j(z)$ (respectively $w_i(z)$) is the same function translated to the support of the basis function. In particular, due to the rectangular shape of the boundary elements, the surface integrals over the boundary can be split into an integral over x, y, and an integral over z which has a constant value. Therefore, for basis functions $\mathbf{v}_j$, $\mathbf{w}_i$ in a layer, introducing two constants C_1, C_2

$$\begin{cases} B_h(i,j) = \displaystyle\int_\gamma \mathbf{v}_j \cdot \mathbf{w}_i \ \mathrm{d}S = C_1 \int_\gamma v_j(x,y)\hat{\mathbf{v}}_j \cdot w_i(x,y)\hat{\mathbf{w}}_i \ \mathrm{d}\gamma, & \hat{\mathbf{v}}_j \in \{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}, & \hat{\mathbf{w}}_i = \boldsymbol{\tau}_i, \\ B_h(i,j) = 0, & \hat{\mathbf{v}}_j = \hat{\mathbf{z}}, & \hat{\mathbf{w}}_i = \boldsymbol{\tau}_i, \\ B_h(i,j) = 0, & \hat{\mathbf{v}}_j \in \{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}, & \hat{\mathbf{w}}_i = \hat{\mathbf{z}}, \\ B_h(i,j) = \displaystyle\int_\gamma \mathbf{v}_j \cdot \mathbf{w}_i \ \mathrm{d}S = C_2 \int_\gamma v_j(x,y)\hat{\mathbf{v}}_j \cdot w_i(x,y)\hat{\mathbf{w}}_i \ \mathrm{d}\gamma, & \hat{\mathbf{v}}_j = \hat{\mathbf{z}}, & \hat{\mathbf{w}}_i = \hat{\mathbf{z}}. \end{cases} \tag{42}$$

The entries of this matrix B_h acting on the basis functions with $\hat{\mathbf{v}}_j \in \{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}$ and $\hat{\mathbf{w}}_i = \boldsymbol{\tau}_i$ is just a scaled version of the matrix B_h found in section 3.2.4. Because a scaling of the matrix B_h has no impact on the scheme in (30), the matrix found in section 3.2.4 can be applied to such a layer to enforce the boundary condition for the fields E_x and E_y. Moreover, because the final result is independent of the z variable, the same matrix B_h can be applied to every layer of the three-dimensional grid. For the other basis functions $\mathbf{w}_i'$, $\mathbf{v}_j'$, denoted with a prime to avoid confusion, $\hat{\mathbf{w}}_i' = \hat{\mathbf{v}}_j' = \hat{\mathbf{z}}$. A similar derivation as in section 3.2.4 is used to find a matrix B_h' to enforce the boundary constraint for these basis

functions (and thus E_z) per layer. The corresponding boundary matrix elements are

$$B'_h(i,j) = \|\gamma_{i-1} - \gamma_i\| \int_{\alpha_1}^{\beta_1} \mathbf{v}'_j\left(\sigma_{i-1}(\theta)\right) \cdot \theta \hat{\mathbf{z}} \, \mathrm{d}\theta + \|\gamma_i - \gamma_{i+1}\| \int_{\alpha_2}^{\beta_2} \mathbf{v}'_j\left(\sigma_i(\theta)\right) \cdot (1-\theta) \hat{\mathbf{z}} \, \mathrm{d}\theta. \quad (43)$$

Obviously, these entries are also independent of the z position of the basis functions and can be applied per layer in the z direction. This way, three-dimensional simulations can be done on perfect prism shaped objects. An implementation in MATLAB can be found in Appendix A.2.2.

3.3 Total Field/Scattered Field formulation

The Total Field/Scattered Field (TF/SF) formulation is a method to insert plane waves into a numerical simulation using the Yee scheme. The method is introduced for the two-dimensional case, from which the result for the three-dimensional case is easily obtained. In principle TF/SF can be used together with the FDM as long as the boundary conditions are not violated by the inserted waves. In Fig. 7 part of a two-dimensional grid is shown, consisting of two dimensional finite elements as in Fig. 4.

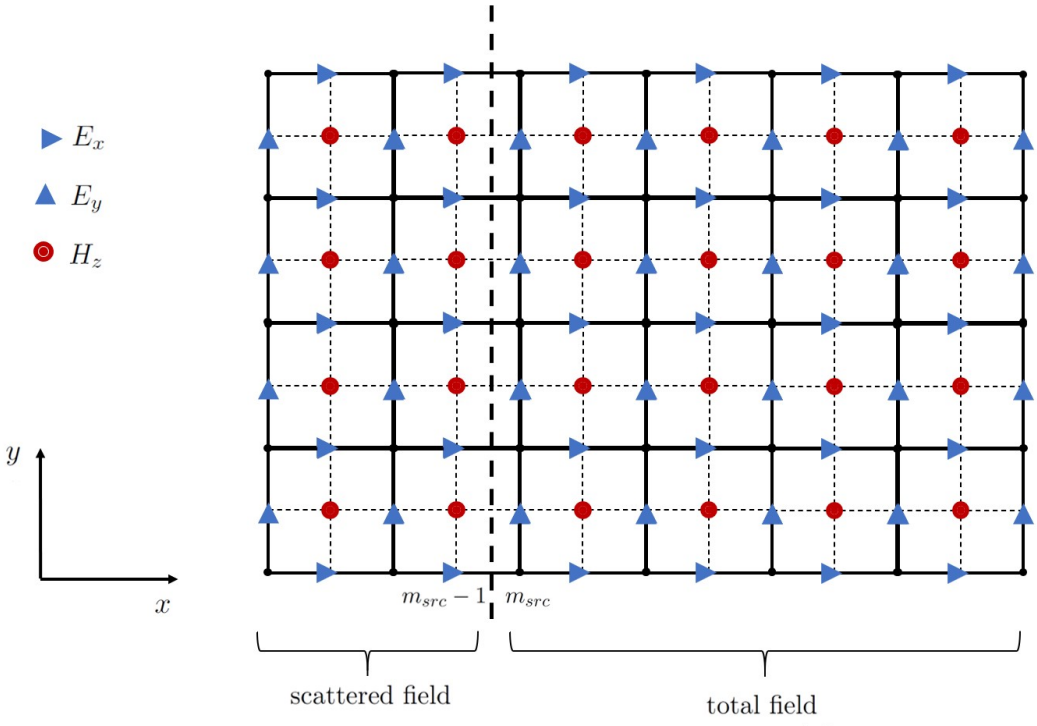

FIGURE 7: Two dimensional FDTD grid for TF/SF formulation

A plane wave travelling in the positive x direction will be inserted into the simulation. The grid is divided into two parts, a total field where the plane wave is present and a scattered field where the plane wave is not present. First, the update equations for the two-dimensional grid are introduced. We denote by $E_x(n_x, n_y)$, $E_y(n_x, n_y)$, $H_z(n_x, n_y)$ the fields at a location determined by the indexing as in Fig. 4, where the bottom left grid point of the cell has location $(n_x h, n_y h)$. Using this indexing, the update equations are

(see also equation (14))

$$\begin{aligned}
E_x^{l+1}(n_x,n_y) &= E_x^l(n_x,n_y) &&+ \frac{\Delta t}{h\epsilon}\left[H_z^{l+\frac{1}{2}}(n_x,n_y) - H_z^{l+\frac{1}{2}}(n_x,n_y-1)\right]\\
E_y^{l+1}(n_x,n_y) &= E_y^l(n_x,n_y) &&- \frac{\Delta t}{h\epsilon}\left[H_z^{l+\frac{1}{2}}(n_x,n_y) - H_z^{l+\frac{1}{2}}(n_x-1,n)\right]\\
H_z^{l+\frac{1}{2}}(n_x,n_y) &= H_z^{l-\frac{1}{2}}(n_x,n_y) &&- \frac{\Delta t}{h\mu}\left(\left[E_y^l(n_x+1,n_y) - E_y^l(n_x,n_y)\right] - \left[E_x^l(n_x,n_y+1) - E_x^l(n_x,n_y)\right]\right)
\end{aligned} \tag{44}$$

These update equations will be altered to insert a plane wave, where fields with indices $\{n_{x,src} \leq n_x \leq N_x, 1 \leq n_y \leq N_y\}$ are in the total field region and fields with the remaining indices are in the scattered field region. From (44) it is obvious that only the update equations for $E_y(n_{x,src}, n_y)$ and $H_z(n_{x,src}-1, n_y)$ need to be compensated. In particular, in the update equation for $E_y(n_{x,src}, n_y)$ the term $H_z(n_{x,src}-1, n_y)$ is a scattered field quantity that has to update a total field quantity, and in the update equation for $H_z(n_{x,src}-1, n_y)$ the term $E_y(n_{x,src}, n_y)$ is a total field quantity that has to update a scattered field quantity. Therefore, the magnetic field of the plane wave will be added to $H_z(n_{x,src}-1, n_y)$ and the electric field of the plane wave will be subtracted from $E_y(n_{x,src}, n_y)$ when those particular discrete curls are calculated. By moving these compensations out of the usual curl calculations, the update equations for E_y and H_z become

$$\begin{aligned}
E_y^{l+1}(n_{x,src},n_y) &= E_y^l(n_x,n_y) - \frac{\Delta t}{h\epsilon}\left[H_z^{l+\frac{1}{2}}(n_{x,src},n_y) - H_z^{l+\frac{1}{2}}(n_{x,src}-1,n_y)\right] + \frac{\Delta t}{h\epsilon}H_z^{src}((l+\tfrac{1}{2})\Delta t)\\
H_z^{l+\frac{1}{2}}(n_{x,src}-1,n_y) &= H_z^{l-\frac{1}{2}}(n_{x,src}-1,n_y) - \frac{\Delta t}{h\mu}\left[E_y^l(n_{x,src},n_y) - E_y^l(n_{x,src}-1,n_y)\right]\\
&\quad + \frac{\Delta t}{h\mu}\left[E_x^l(n_{x,src}-1,n_y+1) - E_x^l(n_{x,src}-1,n_y)\right] + \frac{\Delta t}{h\mu}E_y^{src}(l\Delta t)
\end{aligned} \tag{45}$$

We introduce a plane wave source with wave number k, angular frequency ω and denote by c the speed of light in the medium, where the relation $k = \frac{\omega}{c}$ holds. The plane wave is described by

$$\begin{aligned}
E_y^{src}(x,t) &= E_0\ \mathrm{e}^{\mathrm{i}(kz-\omega t)},\\
H_z^{src}(x,t) &= H_0\ \mathrm{e}^{\mathrm{i}(kz-\omega t)}.
\end{aligned} \tag{46}$$

The constants E_0 and H_0 are related (see [9]) as

$$H_0 = \sqrt{\frac{\epsilon}{\mu}}E_0. \tag{47}$$

The values of these plane waves can be calculated exactly at every time and location in the grid. The location of the source is fixed and determined by the location of $n_{x,src}$. The functions in (46) are then evaluated at times indicated in (45), which concludes the TF/SF formulation in two dimensions.

For a plane wave travelling in three dimensional space, a similar derivation can be done. For a wave travelling in the positive z direction with the total field at indices $k > k_{src}$, only the update equations for $H_x(n_x, n_y, n_{z,src}-1)$, $H_y(n_x, n_y, n_{z,src}-1)$, $E_x(n_x, n_y, n_{z,src})$ and $E_y(n_x, n_y, n_{z,src})$ have to be compensated.

3.3.1 Gaussian beams

In order to mimic a Gaussian beam source, a Gaussian window can be applied to a plane wave source, which is added to simulation by means of the TF/SF method. From (11) we derive the shape of the Gaussian window from the shape of the Gaussian beam at $z = 0$,

$$\mathbf{E}(x, y, 0) = \mathbf{E}_0 \, \mathrm{e}^{\frac{x^2+y^2}{w_0^2}}. \tag{48}$$

Multiplying (48) with a plane wave with wave number k and angular frequency ω gives the approximation to the beam

$$\mathbf{E}(x, y, z, t) = \mathbf{E}_0 \, \mathrm{e}^{\frac{x^2+y^2}{w_0^2}} \, \mathrm{e}^{kz-\omega t}. \tag{49}$$

According to [6], (49) can accurately be used to construct Gaussian beam distribution on a TF/SF interface as long as the Gaussian window (48) decays to a low level at the edges of the grid in the xy plane, and the wavelength $\lambda = \frac{2\pi}{k}$ is smaller than the beam width w_0.

4 Stability

In this section the stability of the FDM will be investigated. An energy of Maxwell's Equations is introduced and a discrete version of this energy is shown to be a conserved quantity as well. From this discrete energy, a stability condition relating the spatial step size h and the time-step Δt will be derived.

4.1 Energy conservation

If $\mathbf{E}$ and $\mathbf{H}$ satisfy Maxwell's equations in a lossless medium and the boundary conditions $\mathbf{E} \times \mathbf{n} = 0$ and $\mathbf{H} \times \mathbf{n} = 0$ are satisfied, an energy conserved in the system is

$$\mathcal{E} = \int_{\mathcal{C}} \left(\epsilon \left| \frac{\partial \mathbf{E}}{\partial t} \right|^2 + \mu \left| \frac{\partial \mathbf{H}}{\partial t} \right|^2 \right) \, \mathrm{d}x. \tag{50}$$

This can easily be proved using Maxwell's equations (2)-(5), see for example [2]. To obtain a discrete variant of this energy, approximations are made to the partial time derivatives, using the inner product and bilinear forms in (16). At time $t = (l + \frac{1}{2})\Delta t$

$$\begin{aligned} \int_{\mathcal{C}} \left| \frac{\partial \mathbf{E}(t)}{\partial t} \right|^2 \, \mathrm{d}x &\approx \left(\frac{e_h^{l+1} - e_h^l}{\Delta t}, \frac{e_h^{l+1} - e_h^l}{\Delta t} \right), \\ \int_{\mathcal{C}} \left| \frac{\partial \mathbf{H}(t)}{\partial t} \right|^2 \, \mathrm{d}x &\approx \left(-\frac{1}{\mu} R_h e_h^{l+1}, -\frac{1}{\mu} R_h e_h^l \right) = \frac{1}{\mu^2} a(e_h^{l+1}, e_h^l). \end{aligned} \tag{51}$$

Remember that $E_h(t)$ in the final FDM scheme (30), is the coefficient vector of $e_h(t)$ in the chosen basis, and $e_h(t)$ is the solution to the variational problem in (15). In the equation above e_h^l thus represents the solution of the variational problem at time step l. Substituting the approximations of the derivatives into (50), we define the discrete energy

at time $t = (l + \frac{1}{2})\Delta t$ as

$$\mathcal{E}_h^{l+\frac{1}{2}} = \epsilon \left(\frac{e_h^{l+1} - e_h^l}{\Delta t}, \frac{e_h^{l+1} - e_h^l}{\Delta t} \right) + \frac{1}{\mu} a(e_h^{l+1}, e_h^l). \tag{52}$$

To show this quantity is conserved, the difference in energy at two consecutive time steps

$$\mathcal{E}_h^{l+\frac{1}{2}} - \mathcal{E}_h^{l-\frac{1}{2}} = \epsilon \int_{\mathcal{C}} \frac{(e_h^{l+1})^2 - 2e_h^{l+1}e_h^l + 2e_h^l e_h^{l-1} - (e_h^{l-1})^2}{(\Delta t)^2} \, \mathrm{d}x + \frac{1}{\mu} \left(a(e_h^{l+1}, e^l) - a(e_h^l, e_h^{l-1}) \right), \tag{53}$$

will be shown to be zero.

It should be observed that (25) still represents the variational scheme of (15), although now with space and time discretization. The corresponding variational scheme with space and time discretization is

$$\begin{aligned} \left(\frac{e_h^{l+1} - 2e_h^l + e_h^{l-1}}{(\Delta t)^2}, \tilde{e}_h \right) + \frac{1}{\mu\epsilon} a(e_h^l, \tilde{e}_h) = b(\tilde{e}_h, \lambda_h^l) \qquad & \forall \tilde{e}_h \in X, \\ b(e_h^l, \tilde{\lambda}_h) = 0 \qquad & \forall \tilde{\lambda} \in L. \end{aligned} \tag{54}$$

Choosing the test function $\tilde{e}_h = e_h^{l+1} - e_h^{l-1}$ and using the linearity of the bilinear forms $a(e, \tilde{e})$ and $b(\tilde{e}, \lambda)$, the first equation becomes

$$\begin{aligned} \int_{\mathcal{C}} \frac{(e_h^{l+1})^2 - 2e_h^{l+1}e_h^l + 2e_h^l e_h^{l-1} - (e_h^{l-1})^2}{(\Delta t)^2} + \frac{1}{\mu\epsilon} \left(a(e_h^{l+1}, e^l) - a(e_h^l, e_h^{l-1}) \right) \\ = b(e_h^{l+1}, \lambda_h^l) - b(e_h^{l-1}, \lambda_h^l). \end{aligned} \tag{55}$$

Now choosing $\tilde{\lambda}_h = \lambda_h^{l-1}$ in the second equation of (54), it follows that $b(e_h^l, \lambda_h^{l-1}) = 0$, and using a time-symmetry argument, also $b(e_h^{l+1}, \lambda_h^l) = 0$. A similar argument is used to prove $b(e_h^{l-1}, \lambda_h^l) = 0$. Substituting (55) in (53) now gives the desired result

$$\mathcal{E}_h^{l+\frac{1}{2}} - \mathcal{E}_h^{l-\frac{1}{2}} = \epsilon \left[b(e_h^{l+1}, \lambda_h^l) - b(e_h^{l-1}, \lambda_h^l) \right] = 0, \tag{56}$$

proving the discrete energy is indeed conserved.

4.2 Stability condition

Now that the discrete energy has been shown to be a conserved quantity, the scheme will be shown to be stable in a newly introduced norm. Notice the following trick using the linearity and symmetry of $a(e, e)$:

$$\begin{aligned} & a\left(\frac{e_h^{l+1} + e_h^l}{2}, \frac{e_h^{l+1} + e_h^l}{2} \right) - a\left(\frac{e_h^{l+1} - e_h^l}{2}, \frac{e_h^{l+1} - e_h^l}{2} \right) = \\ & a\left(\frac{e_h^{l+1}}{2}, \frac{e_h^{l+1} + e_h^l}{2} \right) + a\left(\frac{e_h^l}{2}, \frac{e_h^{l+1} + e_h^l}{2} \right) - a\left(\frac{e_h^{l+1}}{2}, \frac{e_h^{l+1} - e_h^l}{2} \right) + a\left(\frac{e_h^l}{2}, \frac{e_h^{l+1} - e_h^l}{2} \right) = \\ & a\left(\frac{e_h^{l+1}}{2}, e_h^l \right) + a\left(\frac{e_h^l}{2}, e_h^{l+1} \right) = a\left(e_h^{l+1}, e_h^l \right) \end{aligned}$$

As a result, (52) can be rewritten as

$$\mathcal{E}_h^{l+\frac{1}{2}} = \epsilon\left(\frac{e_h^{l+1}-e_h^l}{\Delta t}, \frac{e_h^{l+1}-e_h^l}{\Delta t}\right) + \frac{1}{\mu}a\left(\frac{e_h^{l+1}+e_h^l}{2}, \frac{e_h^{l+1}+e_h^l}{2}\right) - \frac{1}{\mu}a\left(\frac{e_h^{l+1}-e_h^l}{2}, \frac{e_h^{l+1}-e_h^l}{2}\right)$$

We define the bilinear form Q on tuples $(u, v) \in X_h \times X_h$ as

$$Q\left((u_1, v_1), (u_2, v_2)\right) = \frac{\epsilon}{(\Delta t)^2}(u_1 - v_1, u_2 - v_2) + \frac{1}{\mu}a(u_1 + v_1, u_2 + v_2) - \frac{1}{\mu}a(u_1 - v_1, u_2 - v_2). \tag{57}$$

Clearly, Q is symmetric and linear in it's first argument. Then, Q is an inner product if it satisfies the positive definiteness property, i.e. $Q\left((u, v), (u, v)\right) > 0$, for $(u, v) \neq (0, 0)$. Using the fact that $a(v, v) > 0$, this is equivalent to

$$\frac{\epsilon}{(\Delta t)^2}(u, u) - \frac{1}{4\mu}a(u, u) > 0 \qquad \forall u \in X_h \backslash 0. \tag{58}$$

If this constraint is met, Q is an inner product and consequently $|||(u, v)||| = \sqrt{Q\left((u, v), (u, v)\right)}$ is a norm. Using (56) we write

$$\left|\left|\left|(e_h^{l+1}, e_h^l)\right|\right|\right|^2 = \mathcal{E}_h^{l+\frac{1}{2}} = \mathcal{E}_h^{l-\frac{1}{2}} = \left|\left|\left|(e_h^l, e_h^{l-1})\right|\right|\right|^2 \tag{59}$$

and the system is stable in this norm. In fact, (58) is just a CFL condition on the Courant number, denoted C, linking the maximal time step Δt to the spatial step h. Identifying $c = 1/\sqrt{\mu\epsilon}$ as the speed of light in matter,

$$C := \frac{c\Delta t}{h} < 2\left[\frac{a(e_h, e_h)}{h^2(e_h, e_h)}\right]^{-\frac{1}{2}} \qquad \forall e_h \in X_h \backslash 0. \tag{60}$$

In order to determine the maximal time step, the minimum value of the left hand side of the equation is found by maximizing the function in brackets over all possible functions e_h. Remember that E_h is the representation of e_h with respect to the basis functions of the space X_h. Taking the matrix definitions of (19) into account as well as the identities in (22) and (23)

$$\sup_{e_h \neq 0} \frac{h^2 a(e_h, e_h)}{(e_h, e_h)} = \sup_{E_h \neq 0} \frac{h^2 E_h^T A_h E_h}{E_h^T M_h E_h} = \sup_{E_h \neq 0} \frac{E_h^T A_h E_h}{E_h^T E_h}. \tag{61}$$

In this final equation, the Rayleigh quotient can be recognized. The maximum value of the Rayleigh quotient $R(A, x) = \frac{x^T A x}{x^T x}$ is equal to the largest eigenvalue of A. Moreover, in that case the supremizer is an eigenvector corresponding to that eigenvalue. Any suitable algorithm, for example power iteration, can now be used to obtain the largest eigenvalue of A_h and consequently the CFL condition. The power method is a recurrence relation described by

$$x_{n+1} = \frac{A x_n}{\|A x_n\|}, \tag{62}$$

where using sufficient iterations and starting at a random initial vector, x converges to the eigenvector corresponding to (one of) the largest eigenvalue of A. Note that $A_h E_h = h^2 R_h^T R_h E_h$ and hence the power method consists just of calculating discrete curls and normalizing the vector in this case.

5 Results

5.1 CFL conditions

In section 4.2 the CFL-condition relating the time step size to the spatial step size has been found. We denote the CFL condition by C_{max} such that

$$C_{max} = 2 \left[\sup \frac{E_h^T A_h E_h}{E_h^T E_h} \right]^{-\frac{1}{2}}. \tag{63}$$

The power method has been applied on a two-dimensional grid with domain $[0,1] \times [0,1]$ to calculate the CFL condition for increasing numbers of grid-points N_x, N_y. The number of iterations are 10, 100 and 1000. The results can be seen in Fig. 8.

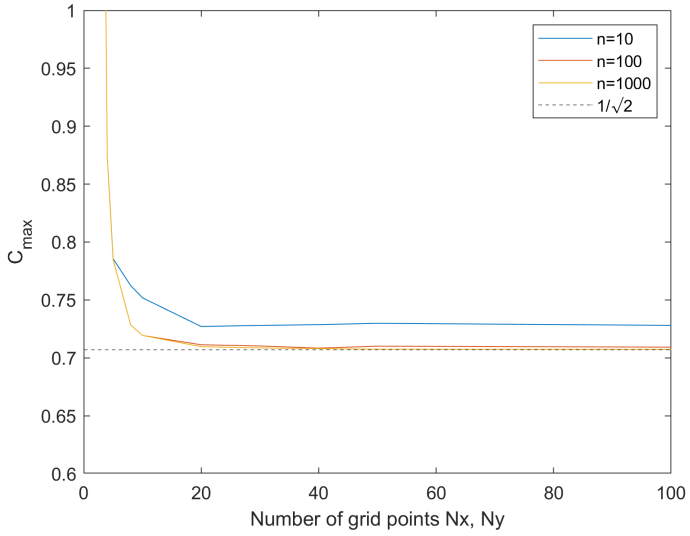

FIGURE 8: CFL condition for a two dimensional grid with mesh refinement, n is the number of iterations

C_{max} can be seen to converge to $1/\sqrt{2}$ for large mesh refinements. Likewise, for one and three dimensional problems $C_{max} = 1$ and $C_{max} = 1/\sqrt{3}$ respectively.

5.2 Convergence on a tilted square

In order to test the convergence and errors of the FDM, a solution of the two dimensional Maxwell's equations is created on a square. The initial conditions are then injected on a tilted square and the fields are simulated by the FDM. To find a simple solution of

Maxwell's equations it is assumed that $\epsilon = \mu = 1$ and the equations are just

$$\begin{aligned} \nabla \times \mathbf{E} + \frac{\partial \mathbf{H}}{\partial t} &= 0 \\ \nabla \times \mathbf{H} - \frac{\partial \mathbf{E}}{\partial t} &= 0 \end{aligned} \tag{64}$$

These equations will be solved on an internal domain $\mathcal{I} = [0, 5] \times [0, 5]$. To make sure the electric field obeys the boundary condition $\mathbf{n} \times (\mathbf{n} \times \mathbf{E}) = 0$ on ∂I, the following ansatz is used

$$\begin{aligned} E_x &= -\cos\left(\frac{\pi}{5}t\right) \sin\left(\frac{\pi}{5}y\right) \\ E_y &= -\cos\left(\frac{\pi}{5}t\right) \sin\left(\frac{\pi}{5}x\right). \end{aligned} \tag{65}$$

Calculating the curl of these equations only gives a result in the z direction. Integrating this curl gives an expression for the magnetic field

$$H_z = \sin\left(\frac{\pi}{5}t\right) \left[\cos\left(\frac{\pi}{5}x\right) - \cos\left(\frac{\pi}{5}y\right)\right] \tag{66}$$

Obviously, the physically correct boundary condition $H_{||} = 0$ on ∂I is discarded here. However, for the convergence study of the electric field with respect to the FDM boundary condition on the electric field this does not matter. Calculating the curl of the H_z field and integrating over time shows that these fields are indeed a solution to the equations in (64). Simply using the FDM on a square boundary, would disregard the interaction between the E_x and E_y fields at the boundary. This interaction is of particular interest, because it differentiates the FDM from the Yee scheme. For this reason the square grid will be tilted as can be seen in Fig. 9.

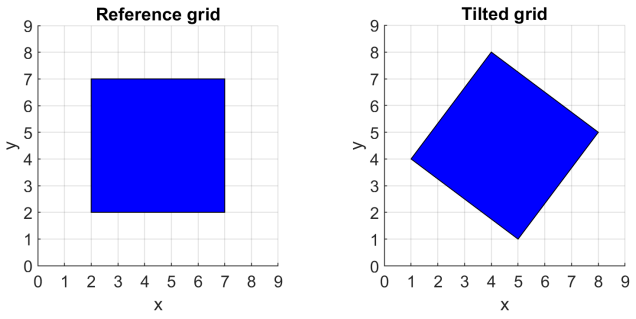

FIGURE 9: Geometry of the problem with the reference grid and the tilted grid of size 5×5

The tilted square is just a rotation around the middle point of the square, using the rotation matrix

$$R = \begin{bmatrix} \frac{4}{5} & \frac{3}{5} \\ \frac{-3}{5} & \frac{4}{5} \end{bmatrix} \tag{67}$$

The initial conditions at time $t = 0$ and $t = -1/2\Delta t$ can be calculated exactly, and rotated onto the tilted grid. A simulation is done for half a period of the cosine, i.e. the end time is $t_{end} \approx 5$. The time step is chosen with the limit of the CFL condition $\Delta t = h/\sqrt{2}$ and therefore the end time is not exactly 5. The convergence study will be concerning the E_x field in the tilted grid, which is a combination of the E_x and E_y fields of the reference grid, therefore validating this choice. The boundary mesh L_h is evenly spaced on the boundary with distance $2h$. Results for $h = 0.2, 0.1, 0.05$ at the end time can be seen in Fig. 10.

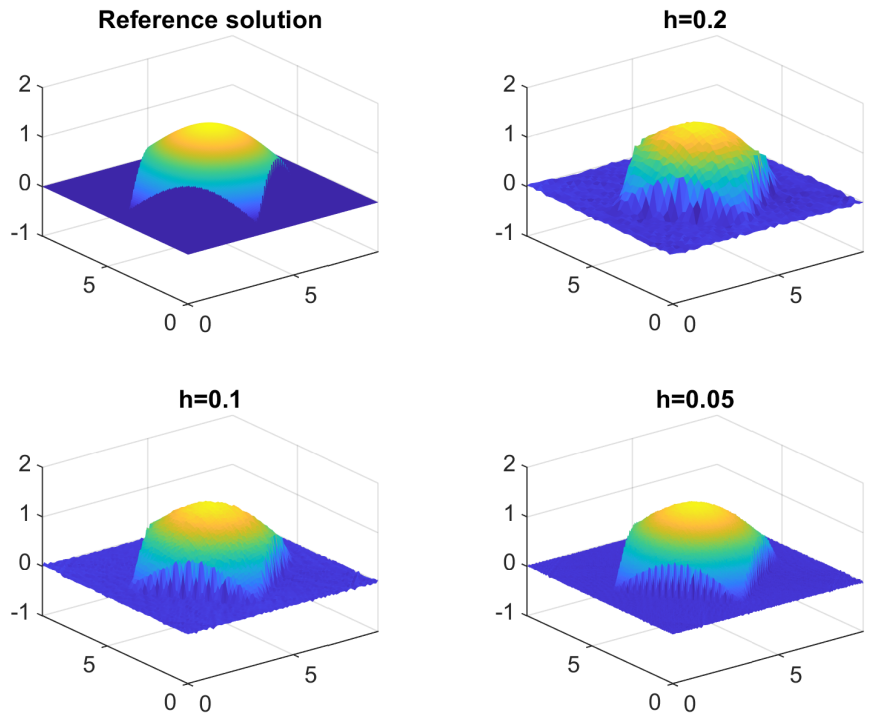

FIGURE 10: FDM approximation of the E_x component fabricated solution on the tilted square at $t = 5$

Let $E^l_{x,h}(n_x, n_y)$ denote the values of the FDM approximation of the solution for the electric field E_x at time $l\Delta t$ and $(x, y) = (n_x h, n_y h)$, and $E^{ref}_x(n_x h, n_y h, t)$ denote the reference value at the same points at time t. We define the error as

$$\eta = h\sqrt{\sum_{n_x,n_y} \left(E^l_{x,h}(n_x, n_y) - E^{ref}_x(n_x h, n_y h, l\Delta t)\right)^2}. \tag{68}$$

This is just an L^2 norm and consequently, we define the relative error as

$$\eta_r = \frac{\eta}{h\sqrt{\sum_{n_x,n_y}\left(E_x^{ref}(n_x h, n_y h, l\Delta t)\right)^2}}. \tag{69}$$

For an error analysis, the approximations at $t_{end} \approx 5$ are compared to the true solution. These errors are shown in Table 1.

TABLE 1: Errors and relative errors of the E_x approximation on the tilted square at time $t \approx 5$

spatinal step size	error	relative error
0.2	0.56931	0.12084
0.1	0.3918	0.083087
0.05	0.2752	0.058371
0.025	0.19537	0.041437
0.0125	0.13778	0.029224

Clearly, the approximations show to converge towards the real solution, although with a low rate. A plot of the absolute difference $|E_x(n_x, n_y, l\Delta t) - E^l_{x,h}(n_x, n_y)|$, is shown in Fig. 11.

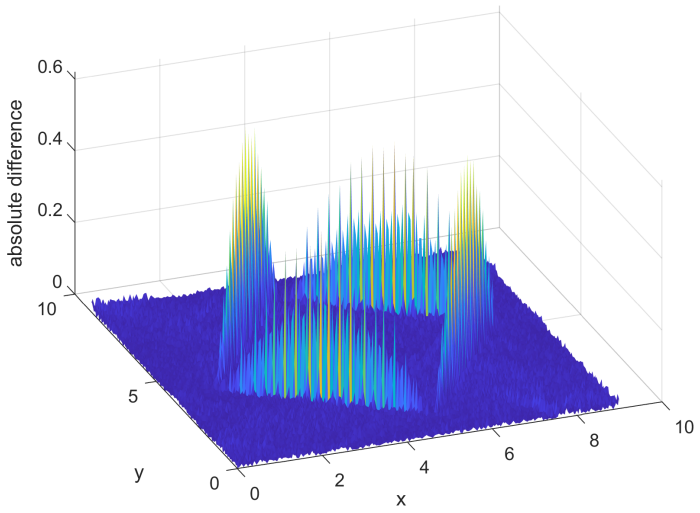

FIGURE 11: Absolute difference $|E_{approx}(n_x, n_y) - E(n_x, n_y)|$

From the figure it can be observed that the values near the boundary have a large impact on the error of the approximation. This induces the idea to look at the error of the approximation inside the square. In Table 2 the errors and relative errors are shown for an inner region $[0.5, 4.5] \times [0.5, 4.5]$ in the middle of the square. Indeed, the errors are much lower and the FDM shows to be a much better approximation to fields within the square than it is to fields at the boundary.

TABLE 2: Errors and relative errors of FDM in an inner region of the tilted square at time $t = 5$

spatial step size	error	relative error
0.2	0.10444	0.024193
0.1	0.073245	0.016948
0.05	0.048508	0.011225
0.025	0.034615	0.0080099
0.0125	0.023981	0.0055497

5.3 A first test: combining FDM and TF/SF

As an example combining the FDM and a TF/SF source, scattering on a two dimensional domain with the TE mode fields, E_x, E_y and H_z, is considered. The domain used is $[0, 7] \times [0, 7]$ microns. A circle with infinite refractive index is centered at $(x, y) = (4.5, 3.5)$, and has a radius of 1µm . A plane wave travelling in the positive x direction is inserted into the total field at $x = 2$µm, with a wavelength $\lambda = 800 \times 10^{-9}$m. The wave polarization is in the direction of E_y and H_z. The spatial step $h = 0.02$µm and the time step is determined by the CFL condition. The boundary condition for the circle is the same boundary condition as the boundary condition between the core and the outside region of an optical fiber, i.e. the tangential electric field is zero. Therefore, the boundary of the circle is discretized with spatial step $2h$ and the boundary condition is modelled using the FDM. Before the initial time $t = 0$, the source is zero. The resulting magnetic field H_z at $t = 1.5331 \times 10^{-14}s$ is shown in Fig. 12. Interference of the waves reflected from the circle can clearly be observed. The reflected waves can also be seen in the Scattered Field region.

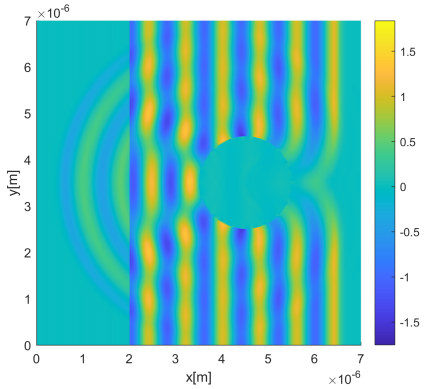

FIGURE 12: TF/SF of magnetic field H_z at $t = 1.5331 \times 10^{-14}$s, after plane wave scattering with wavelength $\lambda = 800$nm on a circle with infinite refractive index. Total field is to the right of $x = 2$µm

5.4 Gaussian beam source excitation of an optical fiber

Excitation of an optical fiber using a Gaussian beam source is simulated. The core of the fiber has size $[0, 7] \times [0, 7]$ µm with permittivity and permeability of free space , i.e. $\epsilon = \epsilon_0$ and $\mu = \mu_0$. For a second simulation the same core size is used with rounded edges. The corners have all different radii between 4µm and 7µm. The input Gaussian beam is polarized in the directions of E_x and H_y, and travels through the fiber in the

positive z direction, normal to the core. The source has a wavelength of $\lambda = 800$nm, a beam width of $w_0 = 0.5$µm and is implemented into the simulation as described in section 3.3.1. Moreover, this Gaussian beam source is transformed into a Gaussian shaped pulse of 75fs (75×10^{-15}s) long, by multiplying it with a Gaussian envelope, the result of which can be seen in Fig. 13. The value of the electric field in the source is normalized and has a maximum equal to 1 in the middle of its focus at time $t = 140.1$fs. Both simulations use a mesh grid with spatial step $h = 0.1$µm, and a time step chosen accordingly from the CFL condition. The distance between points on the boundary is approximately equal to $2h$.

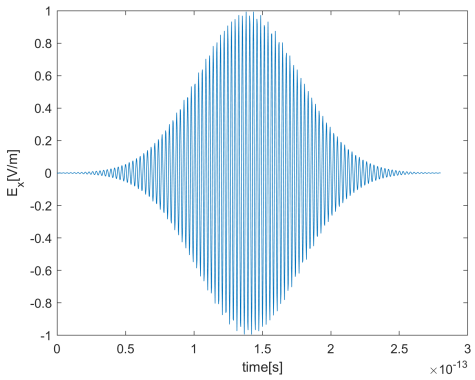

FIGURE 13: Electric field of a Gaussian shaped pulse source in the middle of the focus of a Gaussian beam against time.

The fiber shape perpendicular to its core is assumed to be constant, i.e. it is a right prism. Hence, the method from section 3.2.5 can be used for the simulations. In Fig. 14 the squared sums of the transverse components, $E_x^2 + E_y^2$, are shown at a location 40µm from the source at $t = 280$fs.

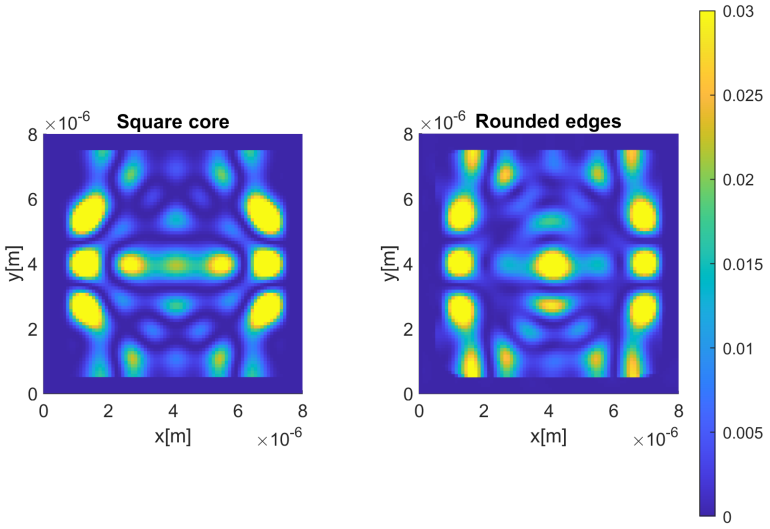

FIGURE 14: Gaussian beam source output pattern $E_x^2 + E_y^2$ for two optical fibers with different core shapes at $t = 280$fs

The patterns observed throughout the fiber for the two core shapes are similar, apart from

the fact that the simulation for a fiber with rounded edges has interference in the middle of the core, where the square core has two speckles in the middle. Further research is needed to characterize the relation between the core shape and the output patterns. It should be noted that the core of the fiber used in this simulation is near the size of a single-mode optical fiber, while optical fibers with a core size of around 50mu, multi-mode fibers, are also of interest.

6 Conclusion

In this work a derivation for the Fictitious Domain Method to solve Maxwell's Equations on a non-rectangular domain is shown. The FDM solves these problems by introducing a new variable at the boundary and a matrix to interact between a boundary mesh and a grid mesh. The two dimensional boundary matrix has been thoroughly investigated, and extended to three dimensional perfect prisms. The method has shown to be stable under the usual CFL conditions. A trivial solution for a rotated two-dimensional square was used to test the FDM, which was shown to converge to it's true solution. The FDM is much better for approximating the fields in the middle of a fiber than at the boundaries. However, it should be noted that waves reflected at the boundary are still very well approximated in the middle of a fiber. An implementation of a Gaussian beam source as an extension on the TF/SF formulation was introduced. Future work should include more three dimensional simulations of optical fibers with a Gaussian beam source, in particular to study the effects of the shape on the speckle pattern and to obtain speckle patterns for multi-mode optical fibers. Moreover, future research should be done on incorporating a more realistic boundary constraint for cladding with a finite refractive index. Two simulations can be run, one for the field inside and one for the field outside, with $B_h E_{in} = B_h E_{out}$ as a boundary constraint for the new problem.

References

[1] V.A. Bokil and R Glowinski. A distributed lagrange multiplier based fictitious domain method for maxwell's equations. *International Journal of Computational and Numerical Analysis and Applications (IJCNAA)*, 6, 2006.

[2] W. Chen, X. Li, and D. Liang. Energy-conserved splitting FDTD schemes for maxwell's equations. *Numer. Math.*, 108:445–485, 2008.

[3] Francis Collino, Patrick Joly, and Florence Millot. Fictitious domain method for unsteady problems. *J. Comput. Phys.*, 138(2):907–938, December 1997.

[4] R. Glowinski, T.W. Pan, T.I. Hesla, and D.D. Joseph. A distributed lagrange multiplier/fictitious domain method for particulate flows. *International Journal of Multiphase Flow*, 25(5):755 – 794, 1999.

[5] R. Glowinski, T.W. Pan, and J. Periaux. A fictitious domain method for external incompressible viscous flow modeled by navier-stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 112(1):133 – 148, 1994.

[6] Lai-Ching Ma and Raj Mittra. Implementation of gaussian beam sources in FDTD for scattering problems. In *2007 IEEE Antennas and Propagation Society International Symposium*, pages 1665–1668, 2007.

[7] T. A. Lenahan. Calculation of modes in an optical fiber using the finite element method and eispack. *The Bell System Technical Journal*, 62(9):2663–2694, 1983.

[8] N. Mabaya and P. E. Lagasse. Finite element analysis of optical waveguides. In *1980 IEEE MTT-S International Microwave symposium Digest*, pages 329–331, 1980.

[9] L. Novotny and B. Hecht. *Principles of Nano-Optics.* Cambridge University Press, New York, 2006.

[10] Katsunari Okamoto. Chapter 3 - optical fibers. In *Fundamentals of Optical Waveguides (Second Edition)*, pages 57 – 158. Academic Press, Burlington, second edition edition, 2006.

[11] J.A. Stratton. *Electromagnetic Theory.* McGraw-Hill Book Company, Inc., Reading, Massachusetts, 1941.

[12] O. Svelto. *Principles of Lasers.* Springer Science+Bsiness Media, LLC, 233 Spring Street, New York, NY 10013, USA, 2010.

[13] K.S. Yee. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *IEEE Trans. on Antennas and propagation*, pages 302–307, May 1966.

A Code

A.1 Two dimensional FDM

Code for the two dimensional FDM is provided. First, the discrete curls and the basis functions are given. Then interactions between electric field matrices and electric field columns are stated, which are used to connect the Yee scheme and the FDM. Thereafter, the functions to find the boundary matrix are given, and finally a main function for the FDM is provided. The main function provided here, has $\epsilon = \mu = 1$ as used in the convergence studies. The FDM used for the TF/SF formulation is just a simple modification on this main function and is therefore omitted.

A.1.1 Curls and basisfunctions

```
function CEz = Ecurl2D(Ex,Ey,dx,dy)
    [Nx,Ny] = size(Ex);
    CEz = deal(zeros(Nx,Ny));

    CEz(1:Nx-1,1:Ny-1) = 1/dx*(Ey(2:Nx,1:Ny-1)-Ey(1:Nx-1,1:Ny-1))...
        -1/dy*(Ex(1:Nx-1,2:Ny)-Ex(1:Nx-1,1:Ny-1));
    CEz(Nx,1:Ny-1) = 1/dx*(-Ey(Nx,1:Ny-1))-1/dy*(Ex(Nx,2:Ny)-Ex(Nx,1:Ny-1));
    CEz(1:Nx-1,Ny) = 1/dx*(Ey(2:Nx,Ny)-Ey(1:Nx-1,Ny))-1/dy*(-Ex(1:Nx-1,Ny));
    CEz(Nx,Ny) = 1/dx*(-Ey(Nx,Ny))-1/dy*(-Ex(Nx,Ny));
end
```

```
function [CHx,CHy] = Hcurl2D(Hz,dx,dy)
    [Nx,Ny] = size(Hz);
    [CHx,CHy] = deal(zeros(Nx,Ny));

    CHx(1:Nx,2:Ny) =  1/dy * (Hz(1:Nx,2:Ny)-Hz(1:Nx,1:Ny-1));
    CHy(2:Nx,1:Ny) = -1/dx * (Hz(2:Nx,1:Ny)-Hz(1:Nx-1,1:Ny));

    CHx(1:Nx,1) = 1/dy * (Hz(1:Nx,1));
    CHy(1,1:Ny) = -1/dx*(Hz(1,1:Ny));
end
```

```
%Here I get the value of the basisfunctions v, so that they can be changed
%to some others functions, with the then proper Gauss-Legendre
%approximation of the polynomial
function value = basisFunctions(point,loc)
if loc=='b'
    value = [1 - point(2),0];
elseif loc=='t'
    value = [point(2),0];
elseif loc=='r'
    value = [0,point(1)];
elseif loc=='l'
    value = [0,1-point(1)];
else
    disp(This should not happen);
```

```
end
```

A.1.2 Column and Matrix mappings

```
%This makes Ex and Ey into a single column vector that can be used for the
%matrix B for the FDM. The reverse function can be used to
%retrieve the Ex/Ey. This makes the standard Fdtd part more clear
%and also makes plotting a lot easier.
%The vector looks like E = [Ex(:,1) Ex(:,2) ..Ex(:,nx) Ey(:,1)..Ey(:,ny)]
function E = mapToEcolumn(Ex,Ey)
    E = [Ex(:); Ey(:)];
end
```

```
%This retrieves Ex and Ey from the single column vector that can be used for
     the
%matrix B for the FDM. The reverse function can be used to
%get the E vector. This makes the standard Fdtd part more clear
%and also makes plotting a lot easier.

function [Ex, Ey] = mapToEmatrix(E,nx,ny)
    Ex = reshape(E(1:nx*ny), [nx ny]);
    Ey = reshape(E(nx*ny+1:2*nx*ny),[nx ny]);
end
```

```
%Returns the index of a E vector in the column with as input the location
%of the Ex or Ey vector in the matrix
%orientation 0 = Ex, 1 = Ey
function index = getColumnIndex(x,y,Nx,Ny,orientation)
    index = orientation*Nx*Ny + (y-1)*Nx + x;
end
```

A.1.3 Intersections, Boundary Matrix

```
% Divides the given straight line in parts adding intersections with edges
% of the grid depending on dx,dy.
% Also adds the beginpoint, and endpoint to the return vector, and makes
% sure it's direction is not changed

function points = intersections(beginpoint,endpoint,dx,dy)
x1 = beginpoint(1);
x2 = endpoint(1);
y1 = beginpoint(2);
y2 = endpoint(2);
m = [floor(x1/dx), floor(x2/dx)]; %locate in which square we are
n = [floor(y1/dy), floor(y2/dy)];
deltax = x2-x1;
deltay = y2-y1;
if(m(1) > m(2))
```

```
    m = flip(m); %make sure we start at smallest square
end
if(n(1) > n(2))
    n = flip(n);
end

points = zeros(m(2)-m(1) + n(2) - n(1)+2,2);
points(1,:) = [x1 y1];
row = 2;

%if the x-position of the squares is different there are some intersections
if(m(1) ~= m(2))
    for i = m(1)+1:m(2)
        %now fill in solved formula for points intersecting vertical grid
        %lines
        points(row,:) = [i*dx, y1+(i*dx-x1) * deltay/deltax];
        row=row+1;
    end
end
%same for if the y-position of the squares is different
if(n(1) ~= n(2))
    for i = n(1)+1:n(2)
        points(row,:) = [x1+(i*dy-y1)* deltax/deltay,i*dy];
        row = row+1;
    end
end
points(row,:) = [x2,y2];

%filter out the points that are intersecting twice
points = uniquetol(points,'ByRows',true);

%return the points sorted, and also in the direction from the first point
%to the second point.
if(x1<x2)
    points = sortrows(points,1)';
elseif(x1>x2)
    points = sortrows(points,1,'descend')';
elseif(y1<y2) %don't forget those vertical lines
    points = sortrows(points,2)';
else
    points = sortrows(points,2,'descend')';
end

end
```

```
% This is the main function for creating matrix B
%
% Nx,Ny,dx,dy are defining the square grid
% boundarySegments are the indices of the boundaryPoints that are
```

```
% connected, for example for a connected boundary with three points it could be:
% boundarySegments = [1 2; 2 3; 3 1]
% This makes sure other non-connected boundaries can be used

function B = boundaryMatrix(Nx,Ny,dx,dy,boundaryPoints,boundarySegments)

sparseValues = [];
sparseSet = [];
sparseM = length(boundaryPoints);
sparseN = Nx*Ny*2;
for segment = boundarySegments %loop through the boundary segments
    %here the segment is splitted into parts that are contained in a single
    %grid square
    startNode = segment(1);
    endNode = segment(2);
    points = intersections(boundaryPoints(:,startNode)...
        ,boundaryPoints(:,endNode),dx,dy);

    %length of this line
    l = norm(boundaryPoints(:,startNode)-boundaryPoints(:,endNode));

    %there is a line less than the number of nodes
    for index = 1:length(points)-1

        lambdaA = points(:,index);
        lambdaB = points(:,index+1);
        tA = norm(lambdaA-boundaryPoints(:,startNode)) / l;
        tB = norm(lambdaB-boundaryPoints(:,startNode)) / l;

        %find the square that this is by looking at the midpoint
        %of the two lambdas
        m = floor((lambdaA(1)+lambdaB(1))/(2*dx));
        n = floor((lambdaA(2)+lambdaB(2))/(2*dy));

        %find the tangent vector
        tau = [lambdaB(1)-lambdaA(1),lambdaB(2)-lambdaA(2)];
        tau = tau/norm(tau);

        %find the gauss legendre points and the transformed points on the
        %grid [0 1 0 1],
        xi = [-1/sqrt(3) 1/sqrt(3)];
        GLPoints = (lambdaB-lambdaA)/2*xi + (lambdaA+lambdaB)/2;
        TGLPoints = [GLPoints(1,:)/dx - m; GLPoints(2,:)/dy-n]; %transformed points

        %values of the ascending basisfunction w from lambdaA to lambdaB
        wValues = [norm(GLPoints(:,1)-points(:,1)), norm(GLPoints(:,2)-
            points(:,1))]/l;
```

```
51
52          % Here the integral are evaluated the first one is always the
53          % descending basisfunction w which is 1 on the startNode, the second
54          % one is the ascending basisfunction w, e.g. which is 1 on the
                endNode
55          % This part could be shortened with some more fancy loops, but this
56          % way it is more understandable
57
58          %bottom basisvector
59          i = startNode;
60          j = getColumnIndex(m+1,n+1,Nx,Ny,0);
61          value = l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'b'),tau)*(1-
                wValues(1))...
62              + dot(basisFunctions(TGLPoints(:,2),'b'),tau)*(1-wValues(2)));
63          sparseSetCheck(i,j,value)
64
65          i = endNode;
66          j = getColumnIndex(m+1,n+1,Nx,Ny,0);
67          value = l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'b'),tau)*(
                wValues(1))...
68              + dot(basisFunctions(TGLPoints(:,2),'b'),tau)*(wValues(2)));
69          sparseSetCheck(i,j,value)
70
71          %upper basisvector
72          i = startNode;
73          j = getColumnIndex(m+1,n+2,Nx,Ny,0);
74          value =  l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'t' ),tau)
                *(1-wValues(1))...
75              + dot(basisFunctions(TGLPoints(:,2),'t' ),tau)*(1-wValues(2)));
76          sparseSetCheck(i,j,value)
77
78          i = endNode;
79          j = getColumnIndex(m+1,n+2,Nx,Ny,0);
80          value =  l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'t' ),tau)*(
                wValues(1))...
81              + dot(basisFunctions(TGLPoints(:,2),'t' ),tau)*(wValues(2)));
82          sparseSetCheck(i,j,value)
83
84          %left basisvector
85          i = startNode;
86          j = getColumnIndex(m+1,n+1,Nx,Ny,1);
87          value =  l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'l'),tau)
                *(1-wValues(1))...
88              + dot(basisFunctions(TGLPoints(:,2),'l'),tau)*(1-wValues(2)));
89          sparseSetCheck(i,j,value)
90
91          i = endNode;
92          j = getColumnIndex(m+1,n+1,Nx,Ny,1);
93          value =  l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'l'),tau)*(
```

```
            wValues(1))...
94              + dot(basisFunctions(TGLPoints(:,2),'l'),tau)*(wValues(2)));
95          sparseSetCheck(i,j,value)
96
97          %right basisvector
98          i = startNode;
99          j = getColumnIndex(m+2,n+1,Nx,Ny,1);
100         value =  l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'r'),tau)
                *(1-wValues(1)))...
101             + dot(basisFunctions(TGLPoints(:,2),'r'),tau)*(1-wValues(2)));
102         sparseSetCheck(i,j,value)
103
104         i = endNode;
105         j = getColumnIndex(m+2,n+1,Nx,Ny,1);
106         value =  l*(tB-tA)/2*(dot(basisFunctions(TGLPoints(:,1),'r'),tau)*(
                wValues(1)))...
107             + dot(basisFunctions(TGLPoints(:,2),'r'),tau)*(wValues(2)));
108         sparseSetCheck(i,j,value)
109     end
110 end
111
112 B = sparse(sparseSet(:,1),sparseSet(:,2),sparseValues,sparseM,sparseN);
113
114     function sparseSetCheck(i,j,value)
115         if(~isempty(sparseSet))
116             [bool, ~] = ismember(sparseSet,[i,j],'rows');
117         else
118             bool = 0;
119         end
120         if(sum(bool))
121             sparseValues(bool) = sparseValues(bool) + value;
122         else
123             sparseSet = [sparseSet; [i,j]];
124             sparseValues = [sparseValues, value];
125         end
126     end
127 end
```

A.1.4 2D Fictitious Domain Method

```
1 % Main function for the 2D FDM
2 %
3 % h is the spatial grid step
4 % t_end the end time
5 % boundaryPoints and boundarySegments determine the boundary by linking the
6 % boundary points with indices in boundarySegments
7 % Ex0,Ey0,Hz0 are initial conditions
8 % dim is the domain of the simulation
9 %
```

```
% output is here Ex (resultEx) at corresponding times (resultTime)
function [resultEx,resultTime] = FDM(h,t_end,boundaryPoints,boundarySegments
    ,Ex0,Ey0,Hz0,dim)

    %length of the fields
    dx = h;
    dy = h;
    Nx = dim(1)/dx+1;
    Ny = dim(2)/dy+1;

    %initialising fields
    Ex = Ex0;
    Ey = Ey0;
    Hz = Hz0;

    %calculating the timestep
    alphaCFL = sqrt(1/2);
    dt = dx*alphaCFL;

    %here the boundary matrix is constructed
    B = boundaryMatrix(Nx,Ny,dx,dy,boundaryPoints,boundarySegments);
    Q_h = B*B';
    [L,U,P,Q] = lu(sparse(Q_h));

    %timesteps and initialising result variable
    nsteps = ceil(t_end/dt);
    resultEx = zeros([nsteps+1,size(Ex)]);
    resultTime = zeros(nsteps+1,1);
    resultEx(1,:,:) = Ex;
    resultTime(1) = 0;
    tic
    for tn = 1:nsteps

        %Calculate curl of the E field into the z-direction and then the H
        %field
        CEz = Ecurl2D(Ex,Ey,h,h);
        Hz = Hz - dt*CEz;

        %calculate curl of the H field into x/y direcion and then the D
            field
        [CHx, CHy] = Hcurl2D(Hz,h,h);

        %calculate E field
        Ex = Ex + dt*CHx;
        Ey = Ey + dt*CHy;

        %Here I map the Ex,Ey field into one large E column vector that I
            now
```

```
        %can use to interact with the Boundary matrix, which is constructed
            to
        %use on a vector E, after this it is mapped back
        E = mapToEcolumn(Ex,Ey);
        lambda = Q*(U\(L\(P*B*E)));
        diff = B'*lambda;
        E = E-diff;
        [Ex,Ey] = mapToEmatrix(E,Nx,Ny);

        %saving the result, here Ex
        resultEx(tn+1,:,:) = Ex;
        resultTime(tn+1) = (tn)*dt;

    end
end
```

A.2 Three dimensional FDM

The additional functions for the three dimensional FDM are provided here.

A.2.1 Curls and additional basis function for Ez

```
%Calculates the discrete 3D curl
%Ex,Ey,Ez are the fields in the 3 directions, with dimensions Nx,Ny,Nz
%dx,dy,dz are the sizes of the grid
%This function leaves the outer row and column of the curl matrices to zero
%THIS IS ONLY FOR THE CURL OF THE E FIELD
%ONLY FOR THE GRID WITH E AT THE LOWER BOUNDARY EDGE

function [Cx, Cy, Cz] = Ecurl3D(Ex,Ey,Ez,dx,dy,dz)
[Nx,Ny,Nz] = size(Ex);
[Cx,Cy,Cz] = deal(zeros(Nx,Ny,Nz));

Cx(2:Nx-1,2:Ny-1,2:Nz-1) = 1/dy*(Ez(2:Nx-1,3:Ny,2:Nz-1)-Ez(2:Nx-1,2:Ny-1,2:
    Nz-1))-1/dz*(Ey(2:Nx-1,2:Ny-1,3:Nz)-Ey(2:Nx-1,2:Ny-1,2:Nz-1));
Cy(2:Nx-1,2:Ny-1,2:Nz-1) = 1/dz*(Ex(2:Nx-1,2:Ny-1,3:Nz)-Ex(2:Nx-1,2:Ny-1,2:
    Nz-1))-1/dx*(Ez(3:Nx,2:Ny-1,2:Nz-1)-Ez(2:Nx-1,2:Ny-1,2:Nz-1));
Cz(2:Nx-1,2:Ny-1,2:Nz-1) = 1/dx*(Ey(3:Nx,2:Ny-1,2:Nz-1)-Ey(2:Nx-1,2:Ny-1,2:
    Nz-1))-1/dy*(Ex(2:Nx-1,3:Ny,2:Nz-1)-Ex(2:Nx-1,2:Ny-1,2:Nz-1));

end
```

```
%Calculates the discrete 3D curl
%Hx,Hy,Hz are the fields in the 3 directions, with dimensions Nx,Ny,Nz
%dx,dy,dz are the sizes of the grid
%This function leaves the outer row and column of the curl matrices to zero
%THIS IS ONLY FOR THE CURL OF THE H FIELD
%ONLY FOR THE GRID WITH E AT THE LOWER BOUNDARY EDGE

```

```
function [Cx, Cy, Cz] = Hcurl3D(Hx,Hy,Hz,dx,dy,dz)
[Nx,Ny,Nz] = size(Hx);
[Cx,Cy,Cz] = deal(zeros(Nx,Ny,Nz));

Cx(2:Nx-1,2:Ny-1,2:Nz-1) = 1/dy*(Hz(2:Nx-1,2:Ny-1,2:Nz-1)-Hz(2:Nx-1,1:Ny
    -2,2:Nz-1))-1/dz*(Hy(2:Nx-1,2:Ny-1,2:Nz-1)-Hy(2:Nx-1,2:Ny-1,1:Nz-2));
Cy(2:Nx-1,2:Ny-1,2:Nz-1) = 1/dz*(Hx(2:Nx-1,2:Ny-1,2:Nz-1)-Hx(2:Nx-1,2:Ny
    -1,1:Nz-2))-1/dx*(Hz(2:Nx-1,2:Ny-1,2:Nz-1)-Hz(1:Nx-2,2:Ny-1,2:Nz-1));
Cz(2:Nx-1,2:Ny-1,2:Nz-1) = 1/dx*(Hy(2:Nx-1,2:Ny-1,2:Nz-1)-Hy(1:Nx-2,2:Ny
    -1,2:Nz-1))-1/dy*(Hx(2:Nx-1,2:Ny-1,2:Nz-1)-Hx(2:Nx-1,1:Ny-2,2:Nz-1));

end
```

```
%Here I get the value of the basisfunctions v, so that they can be changed
%to some others functions, with the then proper Gauss-Legendre
%approximation of the polynomial
function value = basisFunctionsZ(point,xloc,yloc)
value = ((1-point(1))^(1-xloc))*((point(1)^xloc))*...
    ((1-point(2))^(1-yloc))*((point(2)^yloc));
end
```

A.2.2 Boundary Matrix for Ez

```
% This is the main function for creating matrix B_h' (for z vectors)
% (it is very similar to boundaryMatrix.m
%
% Nx,Ny,dx,dy are the dimensions of the grid
% boundaryPoints are the nodes of gamma_h
% boundarySegments are the indices of the boundaryPoints that are
% connected, for example for a connected boundary with three points it could
    be:
% boundarySegments = [1 2; 2 3; 3 1]
% This makes sure other non-connected boundaries can be used

function B = boundaryMatrix_acc(Nx,Ny,dx,dy,boundaryPoints,boundarySegments)

sparseValues = [];
sparseSet = [];
sparseM = length(boundaryPoints);
sparseN = Nx*Ny;
for segment = boundarySegments %loop through the boundary segments
    %here the segment is splitted into parts that are contained in a single
    %grid square
    startNode = segment(1);
    endNode = segment(2);
    points = intersections(boundaryPoints(:,startNode)...
        ,boundaryPoints(:,endNode),dx,dy);

    %length of this line
```

```
26      l = norm(boundaryPoints(:,startNode)-boundaryPoints(:,endNode));
27
28      %there is a line less than the number of nodes
29      for index = 1:length(points)-1
30
31          lambdaA = points(:,index);
32          lambdaB = points(:,index+1);
33          tA = norm(lambdaA-boundaryPoints(:,startNode)) / l;
34          tB = norm(lambdaB-boundaryPoints(:,startNode)) / l;
35
36          %find the square that this is by looking at the midpoint
37          %of the two lambdas
38          m = floor((lambdaA(1)+lambdaB(1))/(2*dx));
39          n = floor((lambdaA(2)+lambdaB(2))/(2*dy));
40
41          %find the gauss legendre points and the transformed points on the
42          %grid [0 1 0 1],
43          xi = [-1/sqrt(3) 1/sqrt(3)];
44          GLPoints = (lambdaB-lambdaA)/2*xi + (lambdaA+lambdaB)/2;
45          TGLPoints = [GLPoints(1,:)/dx - m; GLPoints(2,:)/dy-n]; %transformed
                points
46
47          %values of the ascending basisfunction w from lambdaA to lambdaB
48          wValues = [norm(GLPoints(:,1)-points(:,1)), norm(GLPoints(:,2)-
                points(:,1))]/l;
49
50          % Here the integral are evaluated the first one is always the
51          % descending basisfunction w which is 1 on the startNode, the second
52          % one is the ascending basisfunction w, e.g. which is 1 on the
                endNode
53          % This part could be shortened with some more fancy loops, but this
54          % way it is more understandable
55
56          %bottomleft basisvector
57          i = startNode;
58          j = n*Nx+m+1;
59          value = l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),0,0)*(1-wValues
                (1))...
60              + basisFunctionsZ(TGLPoints(:,2),0,0)*(1-wValues(2)));
61          sparseSetCheck(i,j,value)
62
63          i = endNode;
64          value = l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),0,0)*(wValues(1)
                )...
65              + basisFunctionsZ(TGLPoints(:,2),0,0)*(wValues(2)));
66          sparseSetCheck(i,j,value)
67
68          %bottom right basisvector
69          i = startNode;
```

```
        j = n*Nx+m+2;
        value =  l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),1,0)*(1-wValues
            (1))...
            + basisFunctionsZ(TGLPoints(:,2),1,0)*(1-wValues(2)));
        sparseSetCheck(i,j,value)

        i = endNode;
        value = l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),1,0)*(wValues(1)
            )...
            + basisFunctionsZ(TGLPoints(:,2),1,0)*(wValues(2)));
        sparseSetCheck(i,j,value)

        %top left basisvector
        i = startNode;
        j = (n+1)*Nx+m+1;
        value =  l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),0,1)*(1-wValues
            (1))...
            + basisFunctionsZ(TGLPoints(:,2),0,1)*(1-wValues(2)));
        sparseSetCheck(i,j,value)

        i = endNode;
        value =   l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),0,1)*(wValues
            (1))...
            + basisFunctionsZ(TGLPoints(:,2),0,1)*(wValues(2)));
        sparseSetCheck(i,j,value)

        %top right basisvector
        i = startNode;
        j = (n+1)*Nx+m+2;
        value =  l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),1,1)*(1-wValues
            (1))...
            + basisFunctionsZ(TGLPoints(:,2),1,1)*(1-wValues(2)));
        sparseSetCheck(i,j,value)

        i = endNode;
        value =  l*(tB-tA)/2*(basisFunctionsZ(TGLPoints(:,1),1,1)*(wValues
            (1))...
            + basisFunctionsZ(TGLPoints(:,2),1,1)*(wValues(2)));
        sparseSetCheck(i,j,value)
    end
end

B = sparse(sparseSet(:,1),sparseSet(:,2),sparseValues,sparseM,sparseN);

    function sparseSetCheck(i,j,value)
        %this funtions checks if the sparse location in B_h will already be
        %set
        if(~isempty(sparseSet))
            [bool, ~] = ismember(sparseSet,[i,j],'rows');
```

```
        else
            bool = 0;
        end
        if(sum(bool))
            sparseValues(bool) = sparseValues(bool) + value;
        else
            sparseSet = [sparseSet; [i,j]];
            sparseValues = [sparseValues, value];
        end
    end
end
```

A.2.3 3D Fictitious Domain Method

```
% Main function for the 2D FDM
%
% h is the spatial grid step
% t_end the end time
% boundaryPoints and boundarySegments determine the boundary by linking the
% boundary points with indices in boundarySegments
% Ex0,Ey0,Ez0,Hx0,Hy0,Hz0 are initial conditions
% dim is the domain of the simulation
%
% outputs are multiple fields at the final timestep

function [result,dt] = FDM3D(h,t_end,boundaryPoints,boundarySegments,dim,Ex0
    ,Ey0,Ez0,Hx0,Hy0,Hz0)

    %length of the fields
    [dx,dy,dz] = deal(h);
    Nx = round(dim(1)/dx+1);
    Ny = round(dim(2)/dy+1);
    Nz = round(dim(3)/dz+1);

    %initialising fields
    Ex = Ex0;
    Ey = Ey0;
    Ez = Ez0;
    Hx = Hx0;
    Hy = Hy0;
    Hz = Hz0;

    %some constants
    c0 = 299792458;
    alphaCFL = 1/2;
    dt = alphaCFL*h/c0/4;
    mu0 = 4*pi*1e-7;
    e0 = 8.8541878128*1e-12;
    nsteps = ceil(t_end/dt);
```

```
35
36      %here the boundary matrix B_XY and B_Z are constructed
37      B_XY = boundaryMatrixXY(Nx,Ny,dx,dy,boundaryPoints,boundarySegments);
38      Q_h = B_XY*B_XY';
39      [L_XY,U_XY,P_XY,Q_XY] = lu(sparse(Q_h));
40      B_Z = boundaryMatrixZ(Nx,Ny,dx,dy,boundaryPoints,boundarySegments);
41      Q_h = B_Z*B_Z';
42      [L_Z,U_Z,P_Z,Q_Z] = lu(sparse(Q_h));
43
44       for tn = 0:nsteps
45          disp(tn)
46          %Calculate curl of the E field into the z-direction and then the H
47          %field
48          [CEx,CEy,CEz] = Ecurl3D(Ex,Ey,Ez,dx,dy,dz);
49
50          Hx = Hx - (dt/mu0)*CEx;
51          Hy = Hy - (dt/mu0)*CEy;
52          Hz = Hz - (dt/mu0)*CEz;
53
54          %calculate curl of the H field into x/y direcion and then the E
                field
55          [CHx,CHy,CHz] = Hcurl3D(Hx,Hy,Hz,dx,dy,dz);
56
57          %calculate E field
58          Ex = Ex + (dt/e0)*CHx;
59          Ey = Ey + (dt/e0)*CHy;
60          Ez = Ez + (dt/e0)*CHz;
61          tic
62          for nz = 1:Nz %boundary condition per layer
63              %Here I map the Ex,Ey field into one large E column vector that
                    I now
64              %can use to interact with the boundary matrix, which is
                    constructed to
65              %use on a vector E, after this it is mapped back
66              Ecolumn_XY = mapToEcolumn(Ex(:,:,nz),Ey(:,:,nz));
67              lambda_XY = Q_XY*(U_XY\(L_XY\(P_XY*B_XY*Ecolumn_XY)));
68              Ecolumn_XY = Ecolumn_XY-B_XY' * lambda_XY;
69              [Ex(:,:,nz),Ey(:,:,nz)] = mapToEmatrix(Ecolumn_XY,Nx,Ny);
70
71              %Here Ez is transformed in a column, to
72              %interact with it's own boundary
73              %matrix
74              Ecolumn_Z = Ez(:,:,nz);
75              Ecolumn_Z = Ecolumn_Z(:);
76              lambda_Z = Q_Z*(U_Z\(L_Z\(P_Z*B_Z*Ecolumn_Z)));
77              Ecolumn_Z = Ecolumn_Z - B_Z' * lambda_Z;
78              Ez(:,:,nz) = reshape(Ecolumn_Z,[Nx Ny]);
79          end
80      end
```

```
81
82      %saving the result
83      result = {Ex,Ey,Ez};
84  end
```