MASTER THESIS

FDTD Simulation of Lightning Electromagnetic Fields An approach with the software package MEEP

submitted at the
Faculty of Electrical Engineering and Information Technology
TU Wien
in partial fulfillment of the requirements for the degree of
Diplom-Ingenieur, Dipl.-Ing.
(equals Master of Science, MSc)

Under the supervision of
Univ.-Prof. Dr.-Ing. Wolfgang Gawlik
E370 Institute of Energy Systems and Electrical Drives

Dr. Wolfgang Schulz
OVE Service GmbH, ALDIS

Dr. Gerhard Diendorfer
OVE Service GmbH, ALDIS

---

by

Hannes Kohlmann, BSc
Student ID.: 00925089

January 7, 2020

FDTD Simulation of Lightning Electromagnetic Fields

An approach with the software package MEEP

Hannes Kohlmann 0925089

January 7, 2020

Abstract

A substancial part of lightning research is the analysis of electromagnetic fields caused by lightning strikes. The propagation characteristics of lightning electromagnetic waves depend in most cases on complex irregular ground structures with various possible sets of material parameters, such that analytical calculation methods cannot be applied. Among many numerical methods, the Finite Diffenerce Time Domain (FDTD) method is due to its simplicity a commonly used technique to determine the effects of the propagation environment on lightning electromagnetic fields. In that thesis, the open source FDTD software package MEEP (MIT electromagnetic equation propagation), originally developed for photonics, is tested with respect to its applicability in lighting electromagnetic field computation. After a brief summary of the field of lightning research, the FDTD algorithm, its stability behavior and numerical dispersion are explained, since one has to be aware of those inevitable phenomena in order to produce correct results. Then the software package MEEP is presented. It is shown, how a typical simulation environment is initialized and how lightning current models are introduced into the simulation. After verifying results for simple, perfectly conducting and lossy flat ground in 2D cylindrical symmetry simulations, the effect of irregular terrain profile is analyzed. A real measured lightning current waveform, obtained at the Gaisberg tower in Austria, is used in the simulation and the vertical electric field in a distance of 108.8 km (Neudorf) is computed. Related to the interpretation of simulation results the importance of the FDTD stair-stepping effect is stressed. The result of the E-field is further compared to the related electric field, which was recorded with a GPS-synchronized electric field antenna located in Neudorf. Although non-varying ground conductivity was assumed, the simulated field for the used return stroke model shows good agreement with the recorded field for the chosen parameters.

Kurzfassung

Ein wichtiger Bestandteil der Blitzforschung ist die Berechnung eletromagnetischer Felder, welche durch Blitzentladungen angeregt werden. Die Ausbreitung der damit verbundenen eletromagnetischen Wellen hängen für gewöhnlich von komplexen räumlichen Strukturen und deren Materialparametern ab, welche keiner analytischen Berechnung zugänglich sind. Stattdessen werden numerische Methoden angewendet, wobei eine unter zahlreichen Techniken die Finite Difference Time Domain (FDTD) Methode (dt. Finite-Differenzen Methode im Zeitbereich) ist. Diese Methode wird aufgrund ihrer Einfachheit in der Blitzforschung häufig verwendet, um den Einfluss der Umgebung auf die eletromagnetische Wellenausbreitung bei einer Blitzentladung zu bestimmen. In dieser Arbeit wird das Open Source FDTD Softwarepaket MEEP (MIT electromagnetic equation propagation), das ursprünglich für den Wissenschaftsbereich der Photonik entwickelt wurde, auf seine Anwendbarkeit zur Berechnung von elektromagnetischen Feldern in der Blitzforschung getestet. Nach einer kurzen Zusammenfassung rund um das Wissenschaftsgebiet der Blitzforschung, wird zunächst der FDTD Algorithmus erklärt. Die damit im Zusammenhang stehenden Phänomene der numerischen Instabilität und numerischen Dispersion werden erläutert, da der Anwender mit ihnen vertraut sein muss, um gröbere numerische Artefakte in den Ergebnissen zu vermeiden. Danach folgt eine Einführung in das Softwarepaket MEEP, und es wird gezeigt, wie Simulationsumgebungen initialisiert und Blitzströme zur Anregung der Felder eingefügt werden. Nach der Verifizierung der Resultate für einfache Strukturen wie flachen Untergrund mit unendlicher und endlicher Leitfähigkeit in einer 2D zylindersymmetrischen FDTD Simulation, wird des weiteren der Effekt eines irregulären Geländemodells auf die Blitzfelder analysiert und der Einfluss des FDTD Stufen- (stair-stepping) Effektes auf die Resultate verdeutlicht. Ein real gemessener Blitzstrom, welcher am Gaisberg Turm (Salzburg, Österreich) gemessen wurde, wird in einer Simulation verwendet, und das vertikale elektrische Feld in einer Distanz von 108.8 km (Neudorf, Oberösterreich) ausgewertet. Dieses Resultat wird mit dem zur Blitzentladung zugehörigen elektrischen Feld verglichen, welches in Neudorf mit einer GPS-synchronisierten E-Feld Antenne aufgezeichnet wurde. Trotz der einfachen Annahme von unveränderlicher Bodenleitfähigkeit, stimmen der Feldverlauf der Simulation für die verwendeten Parameter des Blitzstrommodells gut mit dem real gemessenen elektrischen Feld überein.

Contents

Chapter 1

Lightning research - Concepts and methods

Since the beginning of scientific observation of thunderstorms and the lightning discharge, which can be dated back to the time of meticulously conducted experiments by Benjamin Franklin 250 years ago (see [1] as a well written historic abstract of his profound work), lightning observation and study has advanced to a broad field of research thanks to the technological progress. It interconnects various disciplines, reaching from atmospheric physics and thermodynamics over electrical engineering to data sciences and many more. One central goal of that interdisciplinary research field is the exploration of physical processes related to diverse phenomena that lead to the initiation of a lightning discharge and the time before, during and after a discharge. Analysis and simulation of electromagnetic waves in power transmission lines and the destructive effect of currents and fields to objects and electronic devices of different measurement scales is of large interest. As another example, lighting location systems use information contained in the waveform of far fields. Therefore the simulation of wave propagation over terrain with diverse properties plays nowadays an important role for the improvement of lightning detection systems, their detection efficiency and location accuracy. The field simulation by means of various numerical methods permits finding solutions for fields under certain environmental circumstances and parameters, such as complex structured objects or special terrain properties. Their analytic solutions can be obtained with high effort only, if they can be found at all. Many concepts have emerged and evolved over the last century, of which some have proved to be very valuable and others maybe less, but due the advances in computing power some have gained new importance in lightning research. After a brief abstract of the mechanisms and phenomena related to a lightning discharge, in the following subsections I will introduce the most important concepts, which are used for simulating and analyzing electromagnetic fields caused by lightning discharges.

1.1 Cumulonimbus clouds, charge separation, lightning discharge

Before getting into the field of simulating lightning electromagnetic fields radiated by lightning strokes, first the origin and mechanisms of lightning shall be very briefly explained. Complete compilations on and various phenomena related to lightning, have been published by Uman ([2]) or Rakov ([3],[4]). [4] serves as an up-to-date reference for undergraduate and graduate students in lightning physics. Beyond that, the very detailed compendium, 'The Lightning Flash' ([5]), edited by Cooray, contains a lot of physical background related to thunderstorms and lightning in the first chapters and later on compiles numerous important scientific studies conducted in the past decades. Additionally, 'Lightning Electromagnetics' ([6]), edited by Cooray, is an even more detailed book when it comes to the engineering aspects of lightning theory and simulation. The latter two books are recommendable general references for lightning researchers, and in this context they will also be used for comparison of some results obtained by simulations in that thesis.

1.1.1 Charge separation in cumulonimbus clouds

Ordinary clouds consist entirely of liquid droplets. They are not strongly electrified and therefore do not initiate lightning. A necessary condition to develop electrically charged thunderclouds is related to a strong thermal convectieeeon, rapidly extending vertically and reaching far into the freezing level (below 0°C). The final stage of such a development is called cumulonimbus cloud with a typical shape of a mushroom. The generation of a cumulonimbus cloud is for instance caused by hot weather conditions. Rising air masses convey water vapor into higher regions, where it is cooled down. This updraft process lifts air masses. Above the 0°C isotherm water droplets are further cooled down, not freezing until they hit a solid particle (e.g. readily formed ice crystals or graupel). This initiates the crystallization process and graupel and larger ice particles (hail) are formed. Charge separation takes place, when the necessary condition of mixed-phases in the cloud is met: The simultaneous presence of vapor, liquid and solid state of water in the region below 0°C. The charge separation itself is the prerequisite for lightning to occur and happens at the moment when graupel, ice crystals and super-cooled droplets collide.

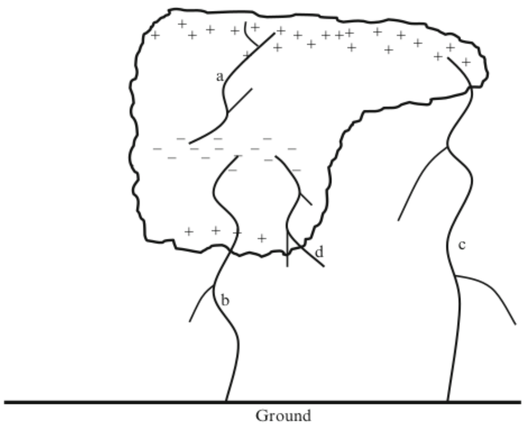


Figure 1.1: Tripolar charge structure of a thundercloud illustrated together with an a) intracloud (IC), b) negative cloud-to-ground (CG) discharge, c) positive CG discharge and d) air discharge with no ground contact. Adapted from [7].

A thunder cloud has a typical tripolar charge structure as shown in Fig. 1.1. The charging mechanism which has found broadest consensus is explained by the Baker et al. hypothesis ([8]). Due to the different vapor pressure of super-cooled water droplets and ice, the graupel will grow at the expense of the water droplets. There is a liquid interface layer between the solid ice state and the surrounding vapor, which is negatively charged (because of neutralizing ion movements caused by water molecule orientation, which follows an energetic optimum). Baker et al. have shown that a collision of such graupel with smaller graupel particles or ice crystals will transfer charge to them which is proportional to their growth rate. Further, it depends on the temperature and the cloud water content, whether a positive or negative net charge graupel particle remains after collision. This was shown in riming experiments by Reynolds et al. [9] and Takahashi [10]. Fig. 1.2 shows the regions where positive charge remains (warmer than about -10℃) and negative charge, when temperatures sink below about -10℃ to -15℃ and a given water content (around 1 g/m^3).

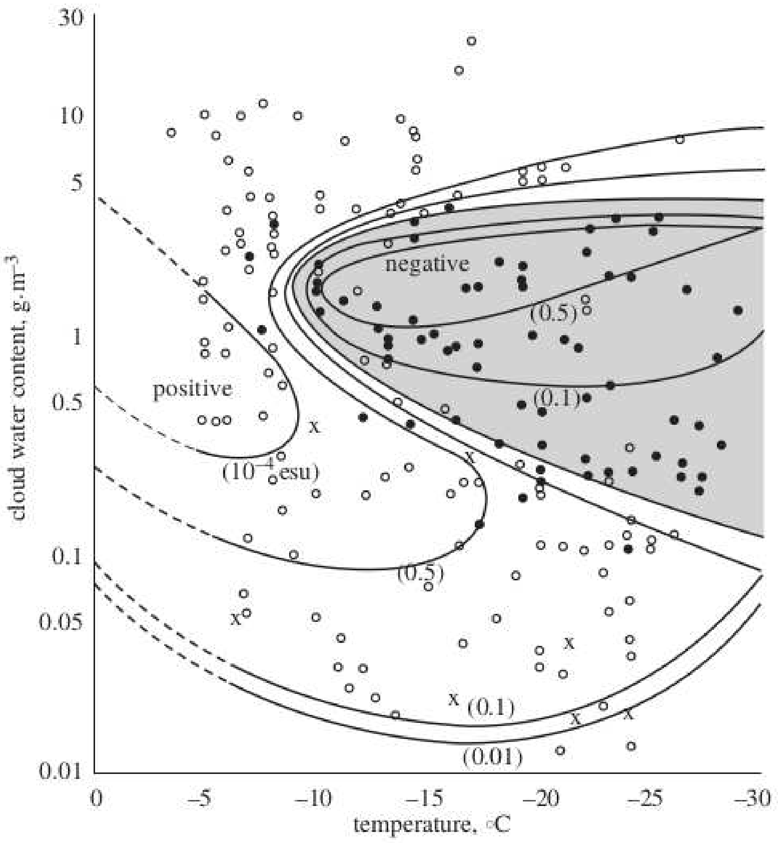


Figure 1.2: Remaining charge of graupel after collision with dependency on temperature and cloud water content. Negative charge remains at temperatures below -10℃ to -15℃, when the water content is around 1 g/m^3 (adapted from [10]).

The height dependency of the charging behavior can now be explained as follows: Graupel that falls from high (cold) regions will collide with ice crystals in the region. When this happens in altitudes corresponding to temperatures below -10℃ where negative graupel remains, the collision will have charged the ice crystals positively. Updrafts will transport that positive charge along with the ice crystals into higher regions, leaving a positively charged upper layer, while the lower regions will be negatively charged by the falling graupel. The vertical extent of the negative charge layer is about 1 km, but can be distributed broadly in horizontal direction. As the graupel falls deeper than the -10℃ zone, collisions will now cause graupel to become positively charged. This leads to a small positive charge pocket below the negative charge region. The overall process results in a tripolar charge structure of the thundercloud, now being able to initiate lightning events, once the electric field strengths on the edge of the regions are high enough. This mechanism is explained in the next subsection.

1.1.2 Initial breakdown, stepped leader, connecting leader

In the last section the charge separation in a thundercloud was explained. Once a critical electrical field strength is reached due to the accumulated charge, in the course of the "preliminary breakdown" free accelerated electrons (electron runaway) will cause electron avalanches at certain points in space. They in turn extend to small channels which are called streamers. When the electron density in the streamers increases, the gas within those channels get ionized in a thermodynamic process (thermalization), resulting in a hot conducting channel, where at some point thermodynamic equilibrium is reached. The result of this conversion is called leader. Leaders are columns of charge which proceed stepwise towards ground in segments of approximately 10-100 m in intervals of 10-100 μs. In order to understand the return stroke modeling approaches in the following subsection, it is important to note that in the process of stepped leaders, charge is deposited densely in the core of the leader (order of a few centimeters in diameter). Due to the high potential of the core, (corona) charges will leak into the region around it, building up a corona shaft, which can be a few meters in diameter or even more, depending on the amount of charge in the core.

The downward moving stepped leader will lower the potential of the cloud towards ground. This in turn will raise the potential at ground level resulting in electric field enhancement. On sharp structures such as leaves, trees, buildings, masts and towers this enhancement is higher. This effect becomes stronger, the lower the stepped leader gets, and charge of opposite sign will move from ground towards the head of the stepped leader. This is the so called connecting leader. This charge grows into the direction, where the streamers of the stepped leaders would have advanced. Finally, the actual return stroke begins at the moment when the connecting leader and the stepped leader contact each other at a height of several tens of meters. Several connecting leaders may emerge from close-by objects, but only the object with a 'successful' connection leader will be struck by the lightning, while the other connecting leaders will degenerate. This is shown Fig. 1.3, where the stepped leader propagating downwards initiates connecting leaders from two towers.

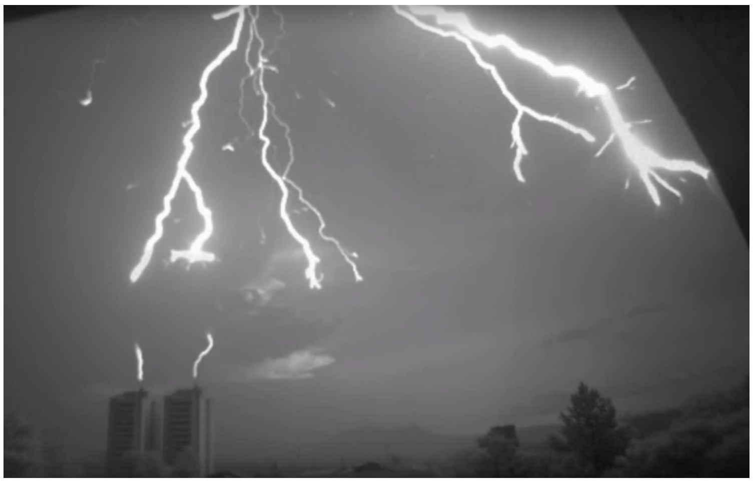


Figure 1.3: Connecting leader extending from two nearby towers towards the downwards moving stepped leader. Screenshot taken from a high-speed video recording made by Marcelo Saba, Instituto Nacional de Pesquisas Espaciais (INPE), 2017 [11].

1.2 Return stroke (RS)

As already described above, at the moment when the stepped leader touches the connecting leader emerging from the ground, an intense potential wave and current pulse, which is related to it, will travel up the channel moving deposited charge to ground. This process is called the return stroke. It gives rise to an electromagnetic field which will propagate hundreds of kilometers over ground at speed of light. The mathematical structure which describes wave propagation are the Maxwell equations, explained later in section 1.3. The shape of the field evoked by a current surge strongly depends on the type of discharge (first, subsequent return stroke), which ground conditions (permittivity, conductivity, ...) prevail and the distance to the lightning channel. In the following two subsections, the mathematical description of the return stroke and the return stroke current are given. It forms the foundation of analytical or numerical field calculation under various conditions.

1.2.1 RS model

The purpose of the model is to calculate or specify the spatial and temporal variation of current sources of the lightning channel. Based on the understanding of the physical structure of the channel, the models shall describe as accurately as possible the way the charge is neutralized from the bottom to the top of the channel, how the related wave fronts travel along the channel and at which speeds. Parameters shall be found for simplified models, that describe the channel currents and their height dependencies (for example based on a base current and a RS velocity). In a way, all the modeles have intersections and similarities or are extensions of simpler ones. Their common goal is to describe RS currents over time in different altitudes that lead to electric and magnetic fields close to real measured remote fields. As reviewed in [12], the return stroke models can be categorized in four classes: 1. electrothermodynamic models, 2. electromagnetic models, 3. distributed-circuit models and 4. the engineering models. The electrothermodynamic model is not used for electromagnetic calculations, thus irrelevant for the scope of that thesis. The resulting spatial and temporal distribution of the current sources are used for analytical calculations of fields or feeding solvers such as MoM or FDTD algorithms.

The most important models within the scope of this work are the so-called "engineering" models. Although being derived based on the physical understanding of the lightning channel, engineering models 'downplay' the physical background by summarizing the current distribution in only two parameters: The base current and the wave front propagation velocity v_{RS}. These parameters are chosen such that the current sources produce electromagnetic fields that are close to practically measured remote fields. The channel current $I(z,t)$ at a height z at time t is related to the base current $I(0,t)$ at ground level $(z = 0)$ or the top of a tall grounded strike object, and incorporates the return stroke propagation wave front. Thus the models summarize the current distribution in only two parameters.

The general description reads:

$$I(z,t) = A(z)F(z,t - z/v) \tag{1.1}$$

$F(z,t - z/v)$ is the channel current. If the channel current does not change its shape along height z, the explicit dependence on z drops out and it can be related to the channel base current with a time retardation:

$$F(z,t - z/v) = I(0,t - z/v) \;\text{ with }\; t > z/v \tag{1.2}$$

The height dependence is modeled by $A(z)$, which describes a change of the amplitude with height z by multiplying this function with $F(z,t - z/v)$. Among the 'transmission line (TL) type' current propagation models (which are part of the 'engineering models', see [6], Chapter

7), there are two models called MTLL (modified TL model with linear amplitude decay) and MTLE (modified TL model with exponential decay). In MTLL, $A(z) = 1 - \frac{z}{H}$, which means that the amplitude at the channel top ($z = H$) is 0 after having decayed linearly. In MTLE, $A(z) = \exp(-z/\lambda)$. At height $z = 5\lambda$, the amplitude has decayed practically to 0. The model used in that thesis is the MTLE and Eq. 1.1 can be rewritten to

$$I(z,t) = e^{-\frac{z}{\lambda}} I(0, t - z/v) \tag{1.3}$$

with a channel base current $I(0,t)$ that is introduced in the next section.

1.2.2 RS current

In every point along the channel, a point source with a corresponding current function will be activated. This describes how the charge changes at this point and depends on the time and height of the point over the strike point. As explained in the previous section, these relations are either calculated by the electromagnetic or distributed current return stroke models or by the engineering models.

The current used in Eq. 1.2 is the channel base current $I(0,t)$ that can be modeled by means of appropriate functions, where, as shown in this section, specific parameters determine the rise and decay time of the current. A special current is constructed by a sum of exponential functions with four curve shaping parameters. These Heidler functions, as they are called, are presented in [13] and can often be found in many analyses in literature. In compact form it can be written as:

$$I(0,t) = \sum_{j=1}^{N} \frac{I_j}{\eta_j} \frac{(\frac{t}{\tau_{j1}})^{n_j}}{(\frac{t}{\tau_{j1}})^{n_j} + 1} \, e^{-\frac{t}{\tau_{j2}}} \tag{1.4}$$

with

$$\eta_i = \exp\left(-\frac{\tau_{j1}}{\tau_{j2}} \left[n_j \frac{\tau_{j2}}{\tau_{j1}}\right]^{1/n_j}\right), j \in \{1, ..., N\}$$

I_j are current amplitudes, that can be interpreted as weights of the exponential terms. The decay of each component is given by τ_{j2}. The resulting peak I_{max} and the rise time ($\Delta t = t|_{I,max \cdot 90\%} - t|_{I,max \cdot 10\%}$) of the final current waveform is a result of the sum of the terms with all corresponding parameters. N depends on the return stroke type that is modeled. It is often chosen as 1 for first return strokes (FS) and as 2 for subsequent return strokes (SS), since they exhibit shorter rise times due to the pre-ionized channel formed by the FS. More terms ($N > 2$) offer more shaping possibilities but are rather uncommon in literature. Frequently occurring values of the parameters can be found for instance in Rachidi et al. [14].

1.3 Wave propagation - electric and magnetic fields

The fundamental understanding of how electromagnetic field components interact (how they are coupled) was completed by J.C. Maxwell by extending Ampere's law to the AmpreMaxwell equation. This completion made it possible to describe the propagation of electromagnetic waves, that do not need any carrying medium. They form the very foundation of electromagnetic field theory, well treated in works such as [15] or [16].

The Maxwell equations in their local form read

$$\frac{d\mathbf{B}}{dt} = -\nabla \times \mathbf{E} - \mathbf{M} \tag{1.5}$$

$$\frac{d\mathbf{D}}{dt} = \nabla \times \mathbf{H} - \mathbf{J} \tag{1.6}$$

$$\mathbf{B} = \mu \mathbf{H} \tag{1.7}$$

$$\mathbf{D} = \varepsilon \mathbf{E} \tag{1.8}$$

where in the MaxwellAmpre law (Eq. 1.6) $\mathbf{J} = \mathbf{J}_{src} + \sigma \mathbf{E}$. $\mathbf{J}_{src}$ denotes the current source. The slight modification in the Maxwell-Faraday equation (Eq. 1.5) is used in [17] with $\mathbf{M} = \mathbf{M}_{src} + \sigma_H \mathbf{H}$ where $\mathbf{M}_{src}$ is a magnetic source current component, which represents fictitious moving magnetic monopoles. It makes the Maxwell equations fully symmetric. Yet it plays a role in the FDTD algorithms, which will be derived later, since it acts as an energy source that can directly stimulate the H-field, with respect to $\mathbf{M}$ equally-oriented, magnetic field $\mathbf{H}$.

When applying the source currents J to equation (1.6), a wave propagation will be initiated due to the coupled partial differential equations of the E-fields and H-fields. The shape of the time domain waveform registered at some space point will not only depend on the time-dependent current waveform and its spatial propagation and development (decay) along the channel itself, but also on the surrounding geometries, the ground parameters such as conductivity, permittivity and magnetic permeability, which influence the wave propagation.

Now the question is, how solutions for the horizontal and vertical E- and H-field components caused by such source currents can be obtained for given conditions. Simple geometries and parameters can be solved in an analytical way and more complex environments need advanced numerical methods. The following two sections introduce the most important concepts, which will be used for comparison and verification throughout this master thesis.

1.4 Relevant analytical solutions

Formally, the governing equations that have to be solved are the (free space) Maxwell equations 1.5 and 1.6. By introducing a vector potential $\mathbf{A}$ and a scalar potential Φ, they can be written as (see [5] pages 351-403 or [4] Appendix 3.1):

$$\mathbf{E}(\mathbf{r}_s, t) = \nabla \Phi - \frac{\partial \mathbf{A}}{\partial t} \tag{1.9}$$

$$\mathbf{B}(\mathbf{r}_s, t) = \nabla \times \mathbf{A} \tag{1.10}$$

The scalar potential can be derived by time-retarded component of source charges (where the location is indicated by the prime) in a volume V':

$$\Phi(\mathbf{r}_s, t) = \frac{1}{4\pi\varepsilon_0} \int_{V'} \frac{\rho(\mathbf{r}'_s, t - \frac{R}{c})}{R} dV' \tag{1.11}$$

and the vector potential can be obtained from time-retarded source current components (where the location indicated by the prime):

$$\mathbf{A}(\mathbf{r}_s, t) = \frac{\mu_0}{4\pi\varepsilon_0} \int_{V'} \frac{\rho(\mathbf{J}'_s, t - \frac{R}{c})}{R} dV' \tag{1.12}$$

Both Φ and $\mathbf{A}$ are related to each other by the Lorenz gauge condition (see [18]):

$$\nabla \cdot \mathbf{A} + \frac{1}{c^2}\frac{\partial \Phi}{\partial t} = 0 \tag{1.13}$$

Whether these equations can be solved analytically or not depends strongly on the complexity of the problem. If the ground conditions are assumed to be simple and the RS model is of a special kind one may find analytical solutions of the temporal behavior of fields at a point $\mathbf{r_s}$.

1.4.1 Applicable solutions

In this section, some solutions and approximations of the above stated formulations are presented. A few are mentioned which are directly applicable either as rules of thumb for estimating fields or easily implementable as algorithms in order to verify or compare simulation results.

A specifically fundamental setup, which has high importance in lightning research, is the case of a vertical lightning stroke to a perfectly conducting plane. A closed solution of the vertical E-field at a large distance (far field, hence only the radiation field component of E_z) is the so-called 'Uman formula' (Eq. 1.14. It can be found in the appendix A of [2] or respectively in [12]:

$$E_z^{rad}(r,t) = -\frac{v_{RS}}{2\pi\varepsilon_0 c^2 r} I(0, t - r/c) \tag{1.14}$$

This relation further assumes that the RS model be the simple TL (transmission line) model with constant velocity of the RS wave front and the wave front has not arrived yet at the top of the channel. Despite the assumptions and strong simplification, this relation, respectively a slight modification of it considering lossy ground, is heavily used as a rule of thumb to estimate the peak current of the stroke. If a remote E-field recording exists, the distance is known and v_{RS} is assumed between $1 \cdot 10^8 - 2 \cdot 10^8\,\mathrm{m/s}$, then the peak current can be estimated by means of the remote E-field. On the opposite, if the channel base current can be measured, the peak of the radiation field component can be estimated by this formula. For instance, with an assumed return stroke speed of $1.5 \cdot 10^8\,\mathrm{m/s}$, a peak current of $1\,\mathrm{kA}$ produces $E_{z,peak}^{rad}(100\,\mathrm{km}) = -\frac{1.5 \cdot 10^8}{2\pi\varepsilon_0 c_0^2 \cdot 100 \cdot 10^3\,\mathrm{m}} \cdot 1\,\mathrm{kA} \approx 0.3\,\mathrm{V/m}$ at a distance of 100km. This value is also used for a course plausibility check of simulated fields in later chapters.

1.5 Numerical methods: FDTD, MoM, ...

As already mentioned earlier, current surges of lightning return strokes, more specifically any charge distribution that changes in time, causes electromagnetic wave propagation due to the governing Maxwell equations. Since obtaining analytical solutions of those for more complex problems is often a daunting or even infeasible task, numerical computation methods have to be applied. Modern computers are able to perform billions of calculations per second and store the results in large random access memory (RAM) for further steps. A variety of methods with different approaches exist, and some of those shall now be briefly discussed. Note that while some RS models have the purpose of obtaining current distributions along the RS channel, this work is laid out more to the computation of propagating electromagnetic fields that interact with structured environments. In the latter case, the return stroke model is assumed to be known by employing an engineering model.

Method of Moments: The *Method of Moments (MoM)* is a boundary element method (see [19]), where a discretized mesh is laid over the structure where field values shall be simulated. Then it is tried to fit the boundary conditions of the cells (elements) into the integral equations in which the partial differential equations are formulated, hence resulting in Maxwell equations in boundary integral form, often called electric field integral equation (EFIE). Boundary conditions can for example be a given prescribed potential or for perfect conductors the E-field that has only a normal component to the surface and zero tangential field. This method exists in time and frequency domain formulation. The attempt is to develop a solvable set of equations in the unknowns by means of the related integro-differential equations. Very detailed derivations for both domains and numerical solution approaches are given in Chapter 9 of [6]. The *advantages* are that for a small surface/volume ratio, a lot of computation time can be saved, since the volume where fields propagate is not needed for the calculation. Further, non-linearities are easy to implement. On the other hand, the resulting matrices can be fully populated and not sparse, which could be solved easier. This causes the complexity to grow in the square of the size (because the surface mesh is 2D), rendering the initial advantage a *disadvantage*. Another disadvantage is that for the time domain formulation, lossy ground cannot be incorporated. In frequency domain again, lossy ground is allowed. Well known NEC-2 and NEC-4 (numerical electromagnetic codes, see) software for EM field simulation are based on frequency domain MoM.

Partial-Element Equivalent-Circuit (PEEC): In its approach, the *PEEC method* is similar to the MoM described above. It can be formulated in the time and frequency domain. The difference is that based on the exact field theory, equivalent circuits are derived ('circuit domain'). The problems are transformed to linear equation formulations with matrices relating currents and voltages instead of fields that have to be solved. The formulation of such a problem in PEEC and its application can be found in a very recent publication by [20]. The *advantage* is that it is also a full-wave solution based on integral equations, providing fast solutions for transient processes and being more efficient than FDTD.
Transmission Line Modeling (TLM): This principle builds on the analogy between electromagnetic wave propagation and voltage wave propagation through a discrete 3D grid which consists of so called symmetrical condensed nodes (SCNs), which are concatenated to resolve the 3D space. They represented short transmission lines that are described by scattering matrices, which in turn determine the transmission and reflection of incident voltage or power waves. Dielectric, magnetic and lossy materials can be included. The similarity to FDTD is that non-linear effects and arbitrary geometries can be handled. The *advantage* is that the numerical dispersion is less than in FDTD. The big *disadvantage* is the computational cost, which is higher than for 3D-FDTD. For 2D-cylindrical the computational cost is comparable.

Finite difference time domain (FDTD): The FDTD method is fully described in the following sections, therefore only the advantages and disadvantages in regard to the other described methods shall be mentioned here.

As *advantages* can be named:

- Easier implementation of the procedure compared to other methods
- Complex geometries and inhomogeneities possible
- Non-linear effects implementable
- Inherent wide-band analysis with a single simulation by transforming the impulse response into the frequency domain

- Numerous publications since the first peer-reviewed paper was published on the investigation of lightning surges in 2001 exist

There are several *disadvantages*. First, the computational costs (though becoming more manageable with stronger computation power) are higher compared to MoM. Second, the staircase approximation of oblique surfaces causes the 'stairstepping effect' (as discussed later) in a standard orthogonal grid and complex implementation of dispersive materials. The latter, though, has already been developed and is part of the software package MEEP (see section 2.2).

Other numerical methods: Other methods, that have already been applied to lightning electromagnetic surge simulations, are for instance the *Finite Element Method (FEM)* or the *Constrained Interpolation Profile Method (CIP)*. Sometimes a selected combination of the techniques is used in order to profit from the advantages at the cost of increasing programming effort. All the above named techniques are summarized in [21] along with a list of publications where they were applied.

Chapter 2

Simulating wave propagation with the FDTD approach

As described at the end of the previous section, there are several ways to simulate fields and their propagation in space and time. That thesis examines the FDTD method. Results and findings throughout that work are obtained by simulations with the open-source software package MEEP [22]. It implements the finite difference time domain (FDTD) approach, which discretizes the Maxwell equations in time and space for a large cell with grid points. These grid points contain all the electric field (E) and magnetic field (H) quantities. No additional assumptions are imposed. That's why results of this method are often called a 'self-consistent full-wave solution' (see for example [21]). In the following section, the theory of the FDTD approach shall be introduced.

2.1 FDTD - Finite Difference Time Domain

The theoretical background of the finite difference time domain method is thoroughly treated in [17] and advanced developments are presented in [23] , which especially targets the micro- and nano-structure world of photonics. A very recent, more specific treatment with applications to lightning surges can be found in [21], which reaches from the FDTD derivation in 3D and cylindrical coordinate systems to engineering approaches for modeling problems like lumped elements, linear and non-linear elements, towers, overhead power transmission lines and more. Since the Maxwell equations are scale invariant (see also [24], Chapter 2), an up-scaling of the spacial unit by a factor causes proportional down-scaling of the resulting frequencies by that factor. For example a geometric structure which is described in the order of millimeters may have a resonance frequency at 300 GHz. If the same structure is simulated in the order of kilometers (factor 10^6), the corresponding resonance frequency will be 300 GHz/10^6 = 300 kHz. The parameters of the macroscopic model of continuous media - i.e. electric permittivity ε, electric conductivity σ and magnetic permeability μ - remain unchanged in the re-scaling.

2.1.1 3D Yee Cell

A way to solve the coupled Maxwell curl equations numerically is to discretize them in (the euclidean) space by $i\Delta x$, $j\Delta y$, $k\Delta z$ and in time with multiples of a time increment $n\Delta t$, with $i, j, k \in \mathbb{Z}$ and $n \in \mathbb{N}_0$. The field components $u(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$ can also be denoted as $u^{n\Delta t}_{i\Delta x, j\Delta y, k\Delta z}$ or in a shortened way also as $u^n_{i,j,k}$, which will later be the preferred notation. Solving the equations for discrete points will lead to the computation of the complete grid of points for every time increment Δt, where the field values are *updated* with dependence of the neighboring E and H field values. To be precise, there are two separate grids for the

H-field and E-field, which are not perfectly aligned but offset by half a spatial increment, which is advantageous at the procedure of the *finite difference* method. This will be mentioned again later in the course of deriving the update equations. The grids contain all available field components at every point and further grids contain the material parameters ε, μ and σ assigned to those points. This makes an implementation of inhomogeneous material possible by simply specifying an array with changing parameter values from point to point.
First, we will start writing out those components from the vector equations 1.5 and 1.6. After plugging in the material equations 1.7 and 1.8 into 1.5 and 1.6, one gets six coupled equations, that need to be discretized. For two of those components, H_z and E_z, the partial derivative equations appear as follows:

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left[\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} - M_z\right] \tag{2.1}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\varepsilon}\left[\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - J_z\right] \tag{2.2}$$

In the following steps, the method for deriving the update equations will be demonstrated for these two components from above. The discrete field components $u^n_{i,j,k}$, are $E^n_{i,j,k}$ and $H^n_{i,j,k}$. The spatial and time derivatives will be approximated by a *finite difference*. The central difference approximation method was first developed by Yee ([25]), which leads to second order accuracy, when the higher order terms (denoted by the Landau-symbol $\mathcal{O}$) are dropped:

$$\frac{\partial H_z}{\partial t} = \frac{H_z\big|^{n+\frac{1}{2}}_{i,j,k+\frac{1}{2}} - H_z\big|^{n-\frac{1}{2}}_{i,j,k+\frac{1}{2}}}{\Delta t} + \mathcal{O}\left[(\Delta t)^2\right] \tag{2.3}$$

$$\frac{\partial E_y}{\partial x} = \frac{E_y\big|^{n}_{i+\frac{1}{2},j,k+\frac{1}{2}} - E_y\big|^{n}_{i-\frac{1}{2},j,k+\frac{1}{2}}}{\Delta x} + \mathcal{O}\left[(\Delta x)^2\right] \tag{2.4}$$

$$\frac{\partial E_x}{\partial y} = \frac{E_x\big|^{n}_{i,j+\frac{1}{2},k+\frac{1}{2}} - E_x\big|^{n}_{i,j-\frac{1}{2},k+\frac{1}{2}}}{\Delta y} + \mathcal{O}\left[(\Delta y)^2\right] \tag{2.5}$$

Since the curl equations 1.5 and 1.6 have to be satisfied for every computation point, the index in the above equations was chosen arbitrarily to be $i, j, k + \frac{1}{2}$, such that they match the example graph shown in Fig. 2.1 and 2.2. Plugging these spatial and time derivatives of the field components in 2.3 - 2.5, and $M_z = M_{src,z} + \sigma_H\ H_z\big|^n_{i,j,k+\frac{1}{2}}$ into 2.1, we get:

$$\frac{H_z\big|^{n+\frac{1}{2}}_{i,j,k+\frac{1}{2}} - H_z\big|^{n-\frac{1}{2}}_{i,j,k+\frac{1}{2}}}{\Delta t} = \\ = \frac{1}{\mu}\left[\frac{E_y\big|^{n}_{i+\frac{1}{2},j,k+\frac{1}{2}} - E_y\big|^{n}_{i-\frac{1}{2},j,k+\frac{1}{2}}}{\Delta x} - \frac{E_x\big|^{n}_{i,j+\frac{1}{2},k+\frac{1}{2}} - E_x\big|^{n}_{i,j-\frac{1}{2},k+\frac{1}{2}}}{\Delta y} - M_{src,z}\big|^n_{i,j,k+\frac{1}{2}} - \sigma_H H_z\big|^n_{i,j,k+\frac{1}{2}}\right] \tag{2.6}$$

In the equation above we see that the time indices of the H-field on the left-hand side ($n + \frac{1}{2}$ and $n - \frac{1}{2}$) do not coincide with the time index of the H-field on the right-hand side (n). This can be resolved by introducing the time mean value $H_z\big|^n_{i,j,k+\frac{1}{2}}$ between $(n - \frac{1}{2})\Delta t$ and $(n + \frac{1}{2})\Delta t$:

$$H_z\big|^n_{i,j,k+\frac{1}{2}} = \frac{H_z\big|^{n+\frac{1}{2}}_{i,j,k+\frac{1}{2}} + H_z\big|^{n-\frac{1}{2}}_{i,j,k+\frac{1}{2}}}{2}$$

Plugging this into Eq. 2.6 and resolving for $H_z|_{i,j,k+\frac{1}{2}}^{n+\frac{1}{2}}$, the final update equation of H_z for the point $i, j, k + \frac{1}{2}$ reads:

$$H_z|_{i,j,k+\frac{1}{2}}^{n+\frac{1}{2}} = \xi_{H,H_z} H_z|_{i,j,k+\frac{1}{2}}^{n-\frac{1}{2}} + \xi_{E,H_z} \left\{ E_x|_{i,j+\frac{1}{2},k+\frac{1}{2}}^{n} - E_x|_{i,j-\frac{1}{2},k+\frac{1}{2}}^{n} - \right.$$
$$\left. - E_y|_{i+\frac{1}{2},j,k+\frac{1}{2}}^{n} - E_y|_{i-\frac{1}{2},j,k+\frac{1}{2}}^{n} - \Delta \cdot M_{src,z}|_{i,j,k+\frac{1}{2}}^{n} \right\} \tag{2.7}$$

where

$$\xi_{H,H_z} = \frac{1 - \eta\Delta t}{1 + \eta\Delta t}$$

$$\xi_{E,H_z} = \frac{\frac{\Delta t}{\Delta \cdot \mu_{i,j,k+\frac{1}{2}}}}{1 + \eta\Delta t}, \text{ with } \eta = \frac{\sigma_{H,i,j,k+\frac{1}{2}}}{2\mu_{i,j,k+\frac{1}{2}}}$$

The factors ξ_{u,u_z} where u is the field, depend on the material parameters at the particular computation point and the time increment Δt. The first letter of the index indicates which field types (E or H) the expression consists of that the factor ξ is multiplied with. The second letter indicates the field component which is updated. Further it was assumed, that the space increment $\Delta x = \Delta y = \Delta$.

The same is now done for 2.2:

$$E_z|_{i-\frac{1}{2},j+\frac{1}{2},k+1}^{n+\frac{1}{2}} = \xi_{E,E_z} E_z|_{i-\frac{1}{2},j+\frac{1}{2},k+1}^{n-\frac{1}{2}} + \xi_{H,E_z} \left\{ H_y|_{i,j+\frac{1}{2},k+1}^{n} - H_y|_{i-1,j+\frac{1}{2},k+1}^{n} - \right.$$
$$\left. - H_x|_{i-\frac{1}{2},j+1,k+1}^{n} - H_x|_{i-\frac{1}{2},j,k+1}^{n} - \Delta \cdot J_{src}|_{i-\frac{1}{2},j+\frac{1}{2},k+1}^{n} \right\} \tag{2.8}$$

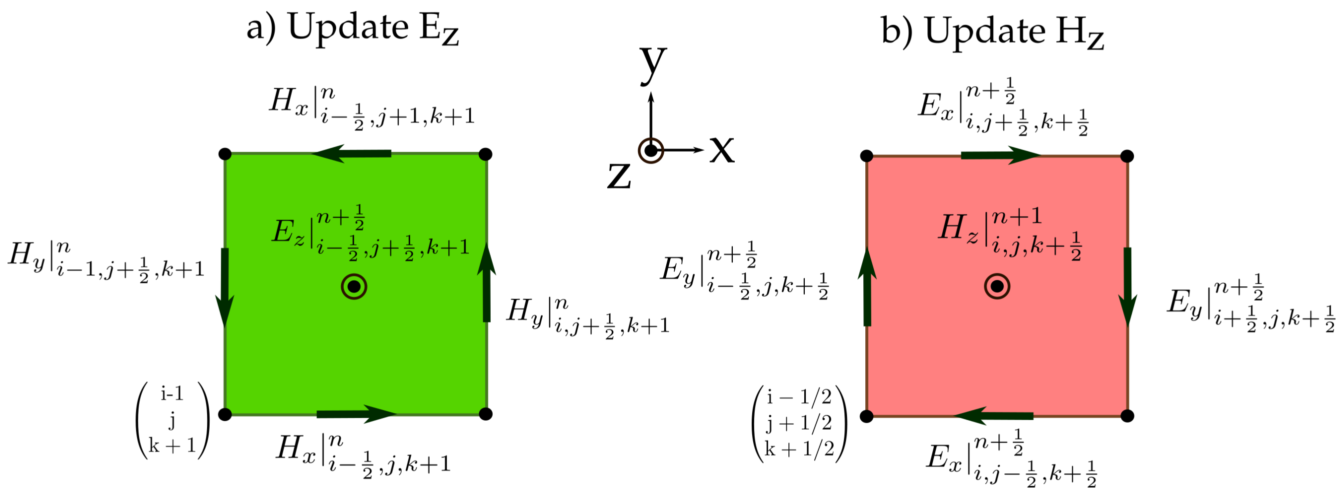
where

$$\xi_{E,E_z} = \frac{1 - \eta\Delta t}{1 + \eta\Delta t}$$

$$\xi_{H,E_z} = \frac{\frac{\Delta t}{\Delta \cdot \mu_{i-\frac{1}{2},j+\frac{1}{2},k+1}}}{1 + \eta\Delta t}, \text{ with } \eta = \frac{\sigma_{E,i-\frac{1}{2},j+\frac{1}{2},k+1}}{2\varepsilon_{i-\frac{1}{2},j+\frac{1}{2},k+1}}$$

Equations 2.7 and 2.8 are called the update equations. They are fully explicit, which means that they do not need any further parallel computations. These two equations are computed consecutively, where the latter equateion uses the field values which were just computed in the former. What can be seen is that the right-hand side of Eq. 2.8 contains H-field values at time $n\Delta t$, which are not available yet from Eq. 2.7. Therefore, depending on the order how the equations shall be executed, the time index is shifted in either of the equations, for example in Eq. 2.7, $n \rightarrow n + \frac{1}{2}$, which will result in a field function $H_z|_{i,j,k+\frac{1}{2}}^{n+1}$. The indexing is a crucial part in the derivation of the update equations of the FDTD, since the field components of the coupled partial derivative equations always depend on

1. the field value of the previous time step itself, and
2. the surrounding field values related to the curl equations.

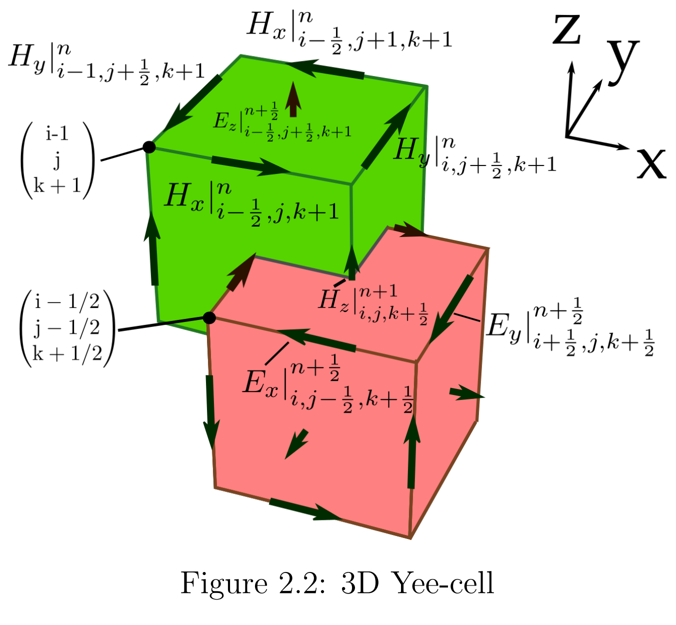


a) Update E_Z
b) Update H_Z
y
z
x
$H_x|^{n}_{i-\frac{1}{2},j+1,k+1}$
$E_z|^{n+\frac{1}{2}}_{i-\frac{1}{2},j+\frac{1}{2},k+1}$
$H_y|^{n}_{i-1,j+\frac{1}{2},k+1}$
$H_y|^{n}_{i,j+\frac{1}{2},k+1}$
$\begin{pmatrix} \text{i-1} \\ \text{j} \\ k+1 \end{pmatrix}$
$H_x|^{n}_{i-\frac{1}{2},j,k+1}$
$E_x|^{n+\frac{1}{2}}_{i,j+\frac{1}{2},k+\frac{1}{2}}$
$H_z|^{n+1}_{i,j,k+\frac{1}{2}}$
$E_y|^{n+\frac{1}{2}}_{i-\frac{1}{2},j,k+\frac{1}{2}}$
$E_y|^{n+\frac{1}{2}}_{i+\frac{1}{2},j,k+\frac{1}{2}}$
$\begin{pmatrix} i-1/2 \\ j+1/2 \\ k+1/2 \end{pmatrix}$
$E_x|^{n+\frac{1}{2}}_{i,j-\frac{1}{2},k+\frac{1}{2}}$

Figure 2.1: Yee-cells; components of the update equations of a) the E-field and b) the H-field

The contributing components from Eq. 2.7 and Eq. 2.8 are shown graphically in Fig. 2.1 a) and b). The indexing may appear somewhat confusing, but can still be understood clearly when coordinates (indices) of the lower left corners are marked.

As mentioned earlier, the grids of the E and H fields are not aligned by design. The offset by half a spatial step size $\frac{\Delta x}{2}$, $\frac{\Delta y}{2}$, $\frac{\Delta z}{2}$ is chosen since, for instance, the E-field update needs the neighboring H-field grid-points vice versa. This is best implemented by shifting the grids between those field components by half a grid step[1]. The half spatial step offset is shown in Fig. 2.2, where the Yee-cells from Fig. 2.1 are depicted as boxes which are interlaced by half a spatial step. Not all components from Fig. 2.1 are depicted, but it shall give a feeling how the update equations work and how the different components are influenced by them. Further implementations of FDTD are discussed in the subsequent sections.



$H_x|^{n}_{i-\frac{1}{2},j+1,k+1}$
$H_y|^{n}_{i-1,j+\frac{1}{2},k+1}$
z
y
x
$E_z|^{n+\frac{1}{2}}_{i-\frac{1}{2},j+\frac{1}{2},k+1}$
$H_y|^{n}_{i,j+\frac{1}{2},k+1}$
$\begin{pmatrix} \text{i-1} \\ \text{j} \\ k+1 \end{pmatrix}$
$H_x|^{n}_{i-\frac{1}{2},j,k+1}$
$\begin{pmatrix} i-1/2 \\ j-1/2 \\ k+1/2 \end{pmatrix}$
$H_z|^{n+1}_{i,j,k+\frac{1}{2}}$
$E_y|^{n+\frac{1}{2}}_{i+\frac{1}{2},j,k+\frac{1}{2}}$
$E_x|^{n+\frac{1}{2}}_{i,j-\frac{1}{2},k+\frac{1}{2}}$

Figure 2.2: 3D Yee-cell

[1]Due to this, the fields are not synchronized. If both fields are needed at the same spatial coordinate, e.g. for calculating the Poynting vector $\mathbf{S} = \mathbf{E} \times \mathbf{H}$, the fields' phases need to be synchronized at the time and sampling location, since they will differ by a phase shift caused by the travel time of half a step size. MEEP offers a function for field synchronization

2.1.2 2D planar

A purely two-dimensional planar FDTD (in Cartesian coordinates) has similar governing update equations as the 3D case derived in the previous section, just that some components into a third dimension are assumed constant and the derivatives are zero. In effect this leads to TEM (transversal electromagnetic) wave computation. It is a feasible, computationally less expensive FDTD where plane wave simulations for axial or planar symmetric structures are of interest, such as in simulations of microstrip lines, waveguides or photonic devices. A point source will in fact be a wire pointing orthogonally into the 2D simulation plane. If many of those point sources will be activated next to each other, an un-attenuated plane wave will be generated. When these sources are activated at different times, the plane wave will change the direction angle but remain un-attenuated. This fact was tested and is plotted in Fig. 2.3 for such a source current 'wall'. The colored waveforms are readouts at increasing distances, thus arriving later in time. The wave remains un-attenuated, thus showing that the planar 2D FDTD is not appropriate for current propagation models in lightning research which are treated in later sections. The currents were of arbitrary nature and the resulting amplitudes are irrelevant for the demonstration of the impracticality.

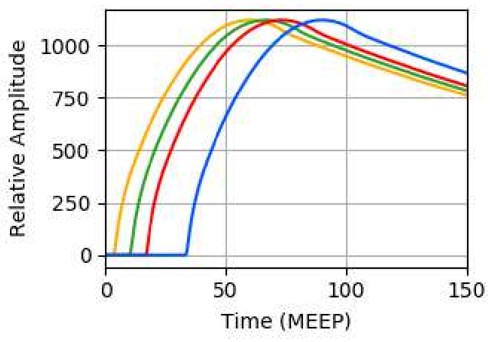


Figure 2.3: Planar 2D FDTD

2.1.3 2D cylindrical symmetry

This case differs from the planar 2D case in the choice of the coordinate system. Here a cylindrical coordinate system is chosen for the differential operators. A derivation, which will not be shown here, leads to update equations of the E_z and H_ϕ components. The special form of these update equations can be looked up in [21]. This realization of FDTD has the benefit of being only quadratic in complexity, which means that the memory usage will increase with the power of two of the resolution, as opposed to third power increase in 3D. That way certain simplified problems can be computed in a reasonable time.

2.1.4 Sources to generate fields in FDTD

Electromagnetic fields in nature follow the principle of causality. Therefore some stimulus has to exist, that excites the propagation of an electromagnetic wave. The components to introduce stimuli can be found directly in the Maxwell equations. Since the two curl equations (Eq. 1.5 and Eq. 1.6) are coupled, any of the field components **E** or **H** and any electric or magnetic source **J** or **M** will stimulate all field quantities. Based on the specific way how this is realized in FDTD, a distinction can be drawn between:

- **Hard sources:** They are represented by E-field or H-field values written into a grid point at every time instance (which is non-physical since they are not transparent, but a viable way to introduce correct fields). It 'dictates' a value independent of anything else, thus such a source overwrites the influence of the fields from the previous time step and neighboring grid points. Physically this renders the point a perfect electric conductor, because the incident tangential field is set to zero and only the source term is assigned. This effect is often unwanted, but can be used intentionally, representing for example a conductor probe that excites a waveguide or microstrip (in 2D or 3D).

- **Current densities M and J:** These components can be found in the update equations 2.7 and 2.8. They are transparent to any fields, since they manipulate the field update additively, and do not prescribe a field value at a specific grid point like a hard source does. Hence they do not cause any scattering. In 3D-FDTD a thin perfectly conducting wire, like a lightning channel with propagating current can be implemented as concatenated current source dipoles. The way how the current is practically implemented is shown in section 2.2.8. If the source has a non-zero time integral, due to Eq. 1.6 [2], charge will accumulate. This is responsible for the electrostatic component of E-fields caused by nearby lightning strikes and correctly represented in simulation results obtained by the FDTD method.

Any implementation of FDTD discretizes the Maxwell equations approximately with central differences in the used coordinate system (e.g. 3D Cartesian or 2D cylindrical). Due to the application of central differences the results will represent the physical fields within second order accuracy. Field propagation in FDTD is stimulated by sources. One way is to inscribe hard sources into the grid. The other way is to use soft sources, which are lumped current sources, see [21]. In 3D this is straight forward, but care has to be taken though, how the source current is introduced into the cylindrical symmetry 2D FDTD.

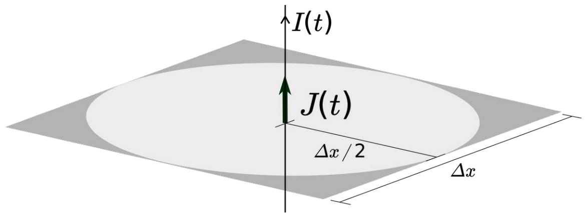


Figure 2.4: Source current density J.
Influence of the geometry (3D vs. 2D cylindrical) on the corresponding $I(t)$.

These source currents are introduced into the update equations. As shown in Fig. 2.4, the geometrical conditions of the unit area with respect to the grid resolution have to be considered. The following equation derives the factor by which the resulting integral currents I for the two coordinates differ. If the grid step size is called Δx in 3D, then 2D cylindrical has step size $\Delta r = \frac{\Delta x}{2}$, and we get:

$$\begin{gathered} J_{cyl} = J_{3D} \\ \frac{I_{cyl}}{(\frac{\Delta x}{2})^2 \pi} = \frac{I_{3D}}{(\Delta x)^2} \\ I_{cyl} = I_{3D} \cdot \frac{\pi}{4} \end{gathered} \tag{2.9}$$

Therefore also any field u_{cyl} obtained from the simulation will by smaller by this factor in cylindrical coordinates and either the source current density or the field sample in 2D cylindrical has to be scaled by the inverse of this factor in order to get matching results.

$$u_{3D} = u_{cyl} \cdot \frac{4}{\pi} \tag{2.10}$$

Therefore the expected field results in 2D cylindrical will be smaller by a factor of $\frac{\pi}{4}$, when the exact same function is used for the J component in both 3D and 2D cylindrical.

[2]Because $\nabla \cdot \nabla \times \mathbf{H} = 0$ and $\nabla \cdot \mathbf{D} = \rho$ (Gauss's law in differential form), the expression $\nabla \cdot \frac{\partial \mathbf{D}}{\partial t} = \nabla \cdot \nabla \times \mathbf{H} - \nabla \cdot \mathbf{J}$ leads to $\nabla \cdot \mathbf{J} = -\frac{\partial \rho}{\partial t}$, which is known as the continuity equation. Integration leads to $\rho(t) = -\int_{-\infty}^{t} \nabla \cdot \mathbf{J}(\tau)\, \mathrm{d}\tau + \text{const}$.

2.1.5 Numerical stability, time step and grid resolution

In this section, two important phenomena will be explained, which an engineer will encounter when working with the FDTD - or more generally with numerical solvers dealing with partial differential (wave) equations:

1. Instability due to the improper proportion of spatial (Δx) to time (Δt) resolution, leading to artificial oscillations growing with time.
2. Numerical dispersion caused by too coarse spatial resolution, which leads to propagation speed deviations and therefore to phase shifts. The latter in turn can be observed as a ringing at sharp edges of the waveform.

For the following considerations, the references [26] and [17] can be recommended. [26] is written from an abstract mathematical perspective, [17] with application to the wave equation.

Any numeric algorithm with the purpose to solve equations of that kind is prone to **instability**, meaning that when the (spatial or temporal) step sizes of the solving algorithm are not chosen appropriately, the solution may diverge. In [26], stability conditions are derived. From any linear partial differential equation, such as the Maxwell equations, algebraic equations can be obtained, replacing the derivatives by finite differences. The procedure was shown in section 2.1.1. In [26] the proof, that for a mesh of points the solution of the finite difference domain will converge to the partial difference equation solution for infinitesimally small grid size, is given. In some cases, e.g. for elliptic equations, the convergence is independent of the mesh. In other cases, such as the hyperbolic case, the mesh width (ratio in different directions) has to be chosen appropriately in order to obtain convergence. By solving the Maxwell equations for a certain component u, one will obtain the wave equation of the hyperbolic form:

$$\frac{\partial^2 \mathbf{u}(\mathbf{x},t)}{\partial t^2} = c^2 \nabla^2 \mathbf{u}(\mathbf{x},t)$$

where $c = \frac{1}{\sqrt{\mu\varepsilon}}$. By discretizing this in space and time, one has to choose a step sizes Δt and $\Delta x, \Delta y, \Delta z$. This discretization causes instability when a certain condition is not fulfilled. This is the well-known Courant-Friedrichs-Lewy (CFL) condition and in general reads:

$$CFL = \Delta t \left(\sum_{i=1}^{n} \frac{c_{x_i}}{\Delta x_i} \right) \leq CFL_{max} \tag{2.11}$$

for n dimensions, where $c_{x,i}$ is the speed of the wave into the different directions. CFL_{max} depends on the special numerical method. In [17] it is shown for the one dimensional case ($\frac{\partial^2 u_x(x,t)}{\partial t^2} = c^2 \frac{\partial^2 u_x(x,t)}{\partial x^2}$), that numerical instability occurs if $\Delta t \frac{c_x}{\Delta x} \leq 1$ is not fulfilled. By looking at the rearranged equation more closely, it has a figurative interpretation: If $c_x > \frac{\Delta x}{\Delta t}$, the wave passes the next grid point before the time increment takes place. Therefore the node 'misses' parts of the wave. By means of the numerical dispersion relation, it can be shown that the frequency ω becomes complex for an improper choice of Δt, resulting in instability. For the tree dimensional case, if $\Delta x_1 = \Delta x_2 = \Delta x_3 = \Delta x$, a wave traveling into the direction of the diagonal of one Yee-cell has equal velocity components into all directions with the resulting speed $c = \sqrt{c_{x_1}^2 + c_{x_2}^2 + c_{x_3}^2} = \sqrt{3}\, c_x$. This results in a maximum CFL in 2.11. Inserting $c_x = c/\sqrt{3}$ and $CFL_{max} = 1$, the CFL condition reads:

$$CFL = \Delta t \left(\sum_{i=1}^{n} \frac{c/\sqrt{3}}{\Delta x_i} \right) = \Delta t \frac{1}{\sqrt{3}} \frac{3\,c}{\Delta x} \leq CFL_{max} = 1 \;\rightarrow\; \Delta t \leq \frac{1}{\sqrt{3}} \frac{\Delta x}{c} = S \cdot \frac{\Delta x}{c} \tag{2.12}$$

where S is the so-called Courant factor, which in general equals $S = \frac{1}{c\sqrt{\sum_{i=1}^{n} \frac{1}{\Delta x_i^2}}}$.

The **numerical dispersion** in 3D can be derived by using in a monochromatic plane wave approach for fields, for example $E|^n_{i,j,k} = E_0 \exp\big(j[\omega\, n\, \Delta t - k_x\, i\, \Delta x - k_y\, j\, \Delta y - k_z\, k\, \Delta z]\big)$ and plugging it into the discretized curl equations of type Eq. 2.6). If $\Delta = \Delta x = \Delta y = \Delta z$, the result is:

$$\left[\frac{1}{S}\sin\left(\frac{\omega\Delta t}{2}\right)\right]^2 = \sin^2\left(\frac{k_x\Delta}{2}\right) + \sin^2\left(\frac{k_y\Delta}{2}\right) + \sin^2\left(\frac{k_z\Delta}{2}\right) \tag{2.13}$$

relating the frequency ω to the wave numbers k_x, k_y, and k_z. The Courant factor $S = \frac{c\Delta t}{\Delta}$ appears in this equation as well. This deviates from the ideal (continuous) dispersion relation $\left(\frac{\omega}{c}\right)^2 = k_x^2 + k_y^2 + k_z^2$. For example, for a major grid axis (e.g. a wave propagating into x-direction) Eq. 2.13 can be simplified and the numerical phase velocity can be found as $c_p = \frac{\omega}{k_x} = c \cdot \pi / \{N_\lambda \sin^{-1}\big[\frac{1}{S}\sin\big(\frac{\pi S}{N_\lambda}\big)\big]\}$, where $\Delta = \lambda_0 / N_\lambda$. N_λ is the number by which the smallest occuring wavelength λ_0 is divided in order to find an appropriate grid resolution Δ. This evaluates for any omega to a numerical phase velocity which is smaller than the real propagation velocity c. This is the so-called *numerical dispersion* phenomenon. It will cause a ringing in the fields signature for fast rise times (such as subsequent return strokes), where the numerical dispersion for high frequency components is large. Although lossy materials tend to attenuate higher frequencies and therefore the ringing, $N_\lambda \approx 20$ is recommended to be chosen.

2.2 MEEP - A flexible FDTD Tool

The acronym MEEP stands for MIT electromagnetic equation propagation. It is a free to use, easy to install open-source software package, which can be interfaced with several programming languages such as C/C++, Python and Scheme. It was developed by the Nanostructures and Computation Group around Stephen G. Johnson on the Massachusetts Institute of Technology since around 2003. The project is available on Github ([27]). By now, there is a comprehensive online documentation, which can be found in [22], with a rich set of examples for both, the Scheme and Python interface. It can output the fields either directly for points/plains or also collectively HDF5 data containers, which themselves can be conveniently opened and inspected with other tools, e.g. HDFviewer ([28]).

2.2.1 Overview and motivation

A good summary of how MEEP works internally is given in [29]. The authors, including some of the MEEP developers, shed light on some subtleties of MEEP, since it uses an advanced approach to implement the algorithms of the Yee-cell in order to become more applicable. For example, to mention one of them, a special sub-gridding technique is used, where the material objects' stair-stepped edges, caused by the discretization, are smoothed by a sub-pixel averaging algorithm. That way, by reducing the stair-step effect (see [30]), a better convergence can be achieved. MEEP employs the well proved and broadly applied concept of perfectly matched layers (boundary materials), which confine the simulation cell with as little reflections as possible (ideally no reflections at all). Also, with MEEP an efficient export of the field values in form of an HDF5 container as a .h5 file is possible. Not only it can be inspected by HDFViewer, but along with the installation comes a tool called `h5topng`, which can convert fields contained in the .h5 to graphics. It can be used to make the field propagation visible by nice color schemes. Further, it supports parallel computing, where multiple cores (as many as desired) can be utilized to perform the simulation with faster total computation time. Because of those reasons, along with the fact of being free to use and open-source, MEEP is a good candidate to choose for many electromagnetic field calculation problems in lightning research. Since implementing the aforementioned details from zero is tedious and requires precise programming work, it saves a lot of initial work when one relies on the effort of engineers at the NanoComp team at the MIT. Since more than 15 years they implement and extend a full-blown FDTD framework for researchers, who can fully focus on their actual work, studies and their findings. Although bound to a certain learning curve, programming simulations with such an existing library is much faster, easier and more reliable than creating a stable framework oneself previous to any correctly working simulation. By means of all these advantages in the sake of convenience, performance and reliability, the entry barrier to use the FDTD method in lightning research can be dramatically lowered by utilizing the MEEP library. Once the applicability of MEEP in this research area is proved and demonstrated for a variety of standard problems in lightning research, researchers shall be able to exchange simulation scripts, verify implementations and results faster than it was possible so far. For scientists or students, which are new to the field, it is a nice extra in MEEP, that they can set up a variety of simulations very quickly and get a feeling for the matter by means of easily generated colored animations of electromagnetic fields, which are normally hard to imagine or to visualize graphically. Thus, it can also serve as a useful toolbox assisting lecturers to generate animations of lightning electromagnetic phenomena and their interaction with the environment. It can also be used as teaching material or be the subject of simulation laboratory units.

To summarize this motivational section: Once the path of MEEP is paved for lightning electromagnetic field computations, this tool can become a useful companion of professors, students and scientists in the academic field of lightning research.

2.2.2 Simulating in MEEP: Core code

Before discussing all the principles of MEEP more in detail, it appears practical to first demonstrate how a simulation is set up and run. The following Python code listing contains the most important parameters which are used. The full working core simulation setup code is given in Appendix A.1.

```
sim = mp.Simulation(dimensions=mp.CYLINDRICAL,
                    resolution=resolution,
                    Courant=courant_factor,
                    cell_size=cell,
                    extra_materials=[material_fct(mp.Vector3())],
                    eps_averaging=True,
                    sources=sources,
                    geometry=geometry,
                    boundary_layers=pml_layers,
                    )
```

Finally, the simulation is run by using the `Simulation.run(...)` method:

```
sim.run(
        *arg_list,
        **keyworded_args,
       )
```

In the following subsections, parameters that MEEP needs, dimensions that it calculates and also peculiarities or known issues will be discussed more in depth, in order to equip the lightning researcher with the necessary knowledge to set up a functioning FDTD simulation in MEEP without any surprises. Further, means to check plausibility of the obtained results or ideas how to make a sanity-check even during run time will be presented.

2.2.3 Dimensions and units

In lightning research, mostly SI units are used for analytical methods and initialization of numerical simulations. Physical quantities used in lightning research are for example time t, length l, current I, magnetic field H, electric field E, electric conductivity σ, dielectric permittivity ε, etc. MEEP is a simulation tool with dimensionless units, therefore the dimensions of the quantities occurring in the Maxwell equations are normalized. The units in MEEP can be brought into relation with SI units, which results in transformation factors for all quantities, which will be derived in this section. Multiplying the results obtained by MEEP will yield the results in the dimension of the SI-unit system. The SI base unit system contains seven units of measure, from which all other SI units can be derived (see [31]). All aforementioned quantities, which are relevant in the lightning electromagnetic environment, can be derived in a way that they consist of four SI base units, which are given in Table 2.1.

Table 2.1: Unit and dimension symbols of four of the seven SI base units.

SI BASE UNITS

	Time	Length	Mass	Electric current
Unit symbol	s	m	kg	A
Dimension symbol	T	L	M	I

As follows, the transformation factor to convert MEEP units into SI units will be derived. Conversion factors Θ_{unit} transform a result in MEEP units to the SI unit system: $q_{SI} = \Theta_{\text{unit}} \cdot q_{MP}$, where q denotes a physical quantity like time, frequency, electric field, conductivity, etc. The indices 'SI' and 'MP' denote the quantity in SI units and MEEP units, respectively. The normalization in MEEP is achieved by the following definitions:

- The MEEP length unit is defined as $[\mathrm{L}] := \mathrm{a}_{\mathrm{MP}}$. If $1{\cdot}\mathrm{a}_{\mathrm{MP}}$ equals 1000 m, then $\Theta_m = 1000\,\mathrm{m}$ is the conversion factor for meters (m).
- Speed: $[\mathrm{L/T}] := \mathrm{c}_{\mathrm{MP}}$. Speed of light c_0 is normalized to $c_{0,\mathrm{MP}} = 1$ and is unit-free. To get a propagation speed in SI units m/s, the quantity has to be multiplied with c_0: $c_{SI} = c_{MP} \cdot c_{0,SI}$, therefore $\Theta_{m/s} = c_{0,SI} = 2.998{\cdot}10^8\,\mathrm{m/s}$.
- The MEEP time unit can be found as follows: $[c] = [L/T] = \mathrm{a}_{\mathrm{MP}}/[T] = 1$ which means that $\mathrm{a}_{\mathrm{MP}} = [T]$ is also the unit of time in the (non-coherent) MEEP unit system. Light propagates a distance of $d \cdot \mathrm{a}_{\mathrm{MP}}$ in the time of $d \cdot \mathrm{a}_{\mathrm{MP}}$ in vacuum. To recover the time in the SI unit, second (s), the time quantity t_{MP} has to be multiplied with $\Theta_s = \frac{\Theta_m}{c_{0,SI}} = \frac{\Theta_m}{2.998{\cdot}10^8\,\mathrm{m/s}}$. If again $1 \cdot \mathrm{a}_{\mathrm{MP}}$ equals 1000 m, $\Theta_s = 3.33\,\mu\mathrm{s}$, which is the time in SI units that light needs to travel a distance of 1000 m.
- To have a fully specified unit system, it has to be defined, how the mass M relates to the current I with unit I_0 in the dimensionless and normalized unit system. This can be achieved by normalizing the field constants [3] ε and μ. Since $c_{0,MP} = 1$, and $c_0^2\varepsilon_0\mu_0 = 1$, then in MEEP units $\varepsilon_0\mu_0 = 1$. If either of both field constants is normalized, the other one will be normalized too: $\varepsilon_{0,MP} = \mu_{0,MP} = 1$. Then one can find that the unit of the mass transforms to $[M] = (I_0^2\ \Theta_m)/(\varepsilon_{0,SI}\ \Theta_{m/s}^4) = (I_0^2\ \Theta_m)/(\varepsilon_{0,SI}\ c_{0,SI})$ with Θ_m being the length transformation factor form above.

[3]Note that the relative field constants ε_r and μ_r are already dimensionless in SI units, therefore they are equal in SI and MEEP units.

The definitions above serve as the basis for unit transformations of further quantities in MEEP. The steps how to get to the transformation factor of the corresponding quantity are:

1. Insert the derived SI units representation for the all quantities which themselves are non-base SI units, for example the unit of the E-field V/m as $\frac{W\,s}{A\,s\,m} = \frac{kg\,m^2}{s^3\,A\,m}$, or expressed with dimension symbols: $[U] = [\frac{M\,L}{I\,T^3}]$.
2. Insert all the factors Θ_{unit} defined above for the time T, length L, current I and mass M, respectively their values that were chosen for that specific simulation. The result is the transformation factor of a quantity in MEEP units to a quantity in SI units.

In Table 2.2, the MEEP unit transformation factors for the most relevant quantities are summarized. The first column shows the physical unit which we normally are interested in. The second column The third column shows the dimension of the quantity in the corresponding base units.

Table 2.2: Unit transformation factors: MEEP units to physical (SI) units. Last column represents numerial transformation factor for $I_0 = 1000\,\text{A}$, $\text{a}_{\text{MP}} = 1000\,\text{m}$, $\text{c}_{0,SI} = 2.998{\cdot}10^8\,\text{m/s}$ and $\varepsilon_{0,SI} = 8.854{\cdot}10^{-12}\,\text{As/Vm}$. SI dimension notation: M is mass, L is length, T is time and I is electric current

Quantity	Unit	Dimension (SI)	Factor Θ	Value
Electric field E	$\frac{\text{V}}{\text{m}}$	$\frac{ML}{IT^3}$	$\frac{I_0}{a_{MP}\,\varepsilon_{0,SI}\,c_{0,SI}}$	$377\,\frac{\text{V}}{\text{m}}$
Magnetic field H	$\frac{\text{A}}{\text{m}}$	$\frac{I}{L}$	$\frac{I_0}{a_{MP}}$	$1\,\frac{\text{A}}{\text{m}}$
Conductivity σ	$\frac{\text{S}}{\text{m}}$	$\frac{I^2T^3}{ML^3}$	$\frac{c_{0,SI}}{a_{MP}}\varepsilon_{0,SI}$	$2.654 \cdot 10^{-6}\,\frac{\text{S}}{\text{m}}$

2.2.4 Resolution, Courant factor and time step

The resolution in MEEP is specified as the amount of grid points per MEEP unit length a_{MP}. If $a_{MP} = 1000\,m$, a resolution of 100 leads to a spatial resolution of 10 m. As described by the CourantFriedrichsLewy (CFL) condition, see section 2.1.5, the courant factor has to meet a special requirement for the algorithm to be stable. It then gives a relation between the spatial resolution and the necessary time increment, which reads

$$\Delta t = S \cdot \frac{\Delta x}{c}$$

where c is the propagation velocity. Taking the dimension-free MEEP units from the previous section where $c = 1$, Δx is $1/100 = 0.01$ for a grid resolution of 100. Therefore $\Delta t = 0.005$ MEEP time units. While the Courant factor is 0.5 by default, it need not be set explicitly as a parameter when instantiating the Simulation class. Though, in the seldom case that S = 0.5 is not sufficient for stability, it can be given as a parameter to the Simulation class constructor next to the resolution and the rest of the information that specifies the simulation: `Simulation(resolution=..., Courant=...)`

During each of these time increments E- and H-fields of each cell have to be calculated. A simulation time of one MEEP time unit will therefore require 200 full cell calculations. The total simulation time is then estimated by MEEP by the amount of time in seconds that it needed for performing one cycle (one time increment) of the simulation multiplied with total the number of cycles needed until the simulation terminates. The output which is printed to the console is shown in Fig. 2.5.

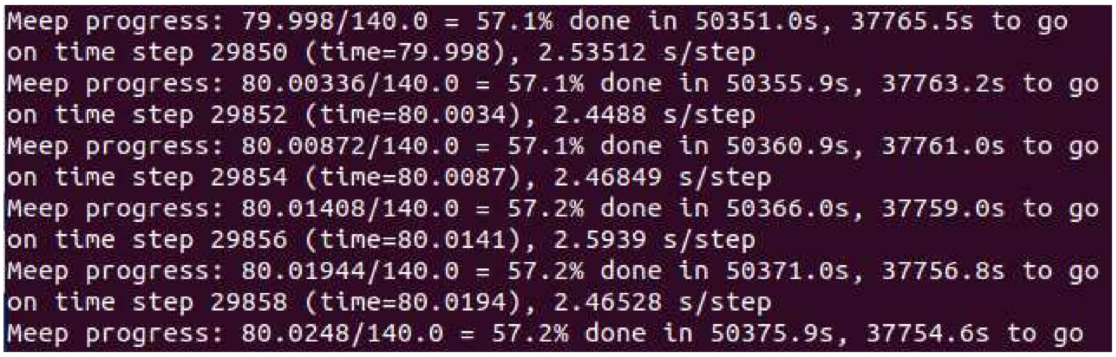

```
Meep progress: 79.998/140.0 = 57.1% done in 50351.0s, 37765.5s to go
on time step 29850 (time=79.998), 2.53512 s/step
Meep progress: 80.00336/140.0 = 57.1% done in 50355.9s, 37763.2s to go
on time step 29852 (time=80.0034), 2.4488 s/step
Meep progress: 80.00872/140.0 = 57.1% done in 50360.9s, 37761.0s to go
on time step 29854 (time=80.0087), 2.46849 s/step
Meep progress: 80.01408/140.0 = 57.2% done in 50366.0s, 37759.0s to go
on time step 29856 (time=80.0141), 2.5939 s/step
Meep progress: 80.01944/140.0 = 57.2% done in 50371.0s, 37756.8s to go
on time step 29858 (time=80.0194), 2.46528 s/step
Meep progress: 80.0248/140.0 = 57.2% done in 50375.9s, 37754.6s to go
```

Figure 2.5: MEEP simulation progress

2.2.5 3D FDTD

In the `mp.Simulation( ... )` class, the argument `dimensions=3` will initialize the simulation in 3D, which is the default values, if `mp.Vector3(x,y,z)` is used with $x, y, z \neq 0$. As described in the subsection above, the Courant factor S can be defined separately by the `Courant=...` parameter and is 0.5 by default. This value already fulfills the requirements for stability in 3D ($S \leqslant \frac{1}{\sqrt{3}}$).

2.2.6 2D cylindrical

In the `mp.Simulation( ... )` class, the argument `dimensions=mp.CYLINDRICAL` will tell the simulation class to be in cylindrical symmetry. Any geometry, which is initialized, will be rotated around the origin, which can have some artificial effects on the wave propagation. This has to be considered, when simulation results are interpreted. For the 2D simulation in `mp.Vector3(x,y,z)`, the y-component has to be set to zero. x represents r and z represents z. The Courant factor S for stability in 2D cylindrical symmetry has to fulfill ($S \leqslant \frac{1}{\sqrt{2}} \approx 0.7$). Therefore setting the value greater than the default S = 0.5 will lead to faster total simulation time. For example setting S = 0.67 saves 25% of computation time.

2.2.7 PML

As the full simulation cell as a grid of Yee-cells is implemented in form of an algorithm, some conditions or the behaviour of the boundary layers (confining planes of the simulation) have to be defined. Since especially at the border, one side of the confining area of the cell contains dynamically changing field values and the other side is always zero, the walls can be interpreted as having the characteristic of a perfectly electric conducting (PEC) material. This can be a wanted property, but often as well unwanted. For example it does not represent the real behavior of a system border, which should be perfectly transparent (vacuum). This means that the so called "return loss" should be very high (ideally infinite), such that nothing is reflected, or in other words all portions of a possibly reflected wave are lost into this medium. Instead an untreated boundary will reflect the waves back and just for an infinitely large cell these waves can be avoided, which in turn is infeasible. Due to this crucial fact, a considerable engineering effort has been invested into the design of boundaries which have an acceptable interaction with the fields.

One method, which is well described in [17], are absorbing boundary conditions. ABCs, as the name suggests, absorb the fields, therefore having the desired characteristic as a neutral boundary of the simulation cell. Perfectly matched layers (PMLs) are materials with a high degree of loss and have the purpose of suppressing reflected waves from the walls by dissipating the fields in that material within as short distance as possible.

2.2.8 Source currents

In the literature of lightning research, one often finds current waveforms $I(t)$ consisting of multiple exponential functions with parameters for typical return stroke (RS) models, as presented in the first chapter of the thesis. Since the results of theoretical analyses or numerical simulations are often dimensional (ratios are rather seldom), the E- and H-fields are compared with respect to certain aspects of the propagation conditions. Therefore, the magnitude and dimension of the result matters. As input that stimulates the fields, MEEP expects current sources J for the respective E-field or H-field components in x, y and z direction in the 3D case. Respectively, in cylindrical symmetry the components are r, φ and z. It shall be stressed again, that in the 2D cylindrical symmetry FDTD, the source current density has to be scaled up by the factor $4/\pi$ to obtain the same results as in 3D FDTD[4]. The following listing shows roughly, how an array of phased (time retarded) current sources for representing a lightning channel can be constructed:

```
sources = [] # instanciate empty list
# Sources per MEEP unit. 1 per grid point, if set equal to resolution:
sources_per_unit = resolution
# distance between two sources for given sources_per_unit:
source_extent = 1/sources_per_unit
# total nr of sources along the whole channel:
total_nr_src = int(stroke_length * sources_per_unit)
# distance of vertical channel from the origin. src_x=0 is the z-axis:
src_x = 0
# Height of ground strike point at src_x (corresponding to terrain model):
src_z = 0
src_component = mp.Ez # Ez is the only practical component!
# loop through channel and append current fct to sources:
for i in range(total_nr_src):
  h = i*source_extent # steps up the channel by int numbers of source_extent
  start_time = h/(v_rs/self.mp_units.c0) # time retardation term in i(t-h/v_rs)
  sources.append(mp.Source(mp.CustomSource(
                                            # CustomSource evaluates this function:
                                              _current_fct(height=h, t_shift=start_time),
                                            # before start_time CustomSource returns 0!
                                              start_time = start_time
                                                  ),
                                  # component stimulated in the update equation
                                  component=src_component,
                                  center=mp.Vector3(src_x,0,src_z+h), # location
                                  size=mp.Vector3(0,0,source_extent)# source extent
                                  )
                    )
```

Listing 2.1: Source currents

[4]IMPORTANT NOTE: Additional to the $\frac{4}{\pi}$ factor, the fields have to be multiplied with the simulation `resolution` variable in the 2D case, defined in the MEEP simulation class. At the time of writing, the author has not found any technical reason why this multiplication has to be performed, yet it is necessary to obtain consistently agreeing results.

The `_current_fct` is given in Appendix A.2. It implements the Heidler function representing the return stroke channel base current and wraps it into a form that `mp.CustomSource` can interpret.

2.2.9 Geometries

MEEP provides geometry primitives `GeometricObject` for material objects, that have some electric properties like permittivity ε, permeability μ and conductivity σ (specified by the property `material` = `mp.Medium(...)`) and are located at `center` = `mp.Vector3(...)`. These primitives (with parameters in parenthesis) are **`mp.Sphere(radius)`**, **`mp.Cylinder(radius, height, axis)`** , **`mp.Cone(radius, radius2)`**, **`mp.Block(size)`**, **`mp.Ellipsoid(axes_size)`**, and **`mp.Prism(...)`**, where the prism parameters are vertices as a list of Vector3 points that make up the prism, height, axis and center.

The advantage of these primitives is that one can make use of the subpixel-averiging property, where the stair-stepped approximation of materials with curvature (e.g. Spheres, Ellipsoids, Prisms ...) is smoothed out. For the above mentioned primitives this subpixel-averaging is very efficient. For a user material function on the other hand, which is described in the next section, this is rather slow because it makes use of integrative methods.

2.2.9.1 User material function

As will be seen in later sections, terrain models with specific ground parameters are often of interest in lightning research. Besides initializing the `geometries` by primitives such as `mp.Blocks` or `mp.Prisms` in the simulation definition, these objects can also behave following a user specific material function. Such a function depends on certain conditions regarding its geometry and can be defined in a form as given in Listing 2.2.

```python
def material_fct(p):
  h_threshold = ground_level(p.x)
  if (p.z<=h_threshold):
    return ground_material
  else:
    return air_material
```

Listing 2.2: MEEP material function as coordinate dependent ground model

This `material_fct` accepts a parameter p, which is a `mp.Vector3()` describing the location in the simulation cell. The object is given to the function by MEEP at the time, when the simulation grid is initialized. Inside the function, a `ground_level` function can serve, for instance, as a distance dependent height model, evaluating `p.x` to a threshold value, which is done for simulations with propagation over some terrain profile later in that work. If the `p.z` values are smaller than the threshold, `ground_material` is returned and the `mp.Block` behaves like this material below the threshold. `air_material` is returned, when the `p.z` lies above the threshold, leading to a transparent behavior of the `mp.Block` in this region. Two important things to point out are:

- The `mp.Block` has to be at least as thick as the height difference between the highest and the lowest point in the user material function, otherwise the structure will be cut off.
- The threshold comparison w.r.t. `Vector3` point `p` is performed in *absolute* MEEP coordinates and *not relative to the bottom of the block*. Therefore it is practical to always keep track of the absolute coordinates of the surface (or bottom) of blocks. The absolute MEEP coordinate of the `material_fct` surface can then be calculated easily[5].

[5]Note: The x-bounds of the simulation cell in `mp.CYLINDRICAL` reach from `[0, cell_x]` and the z-bounds from `[-cell_z/2, +cell_z/2]`.

Sub-pixel averaging can be activated for such a material function, but is turned off by default. This will lead to a stair-stepped approximation of the material function below its curve without any smoothing of edges.

2.2.9.2 Prism model

Another, more efficient, way to initialize a terrain model is to use `mp.Prism(vertices,...)`. The `vertices` is a list of `Vector3(...)` points. In the 3D simulation they have to represent a plain that lies perpendicular to the `axis`. In 2D cylindrical it will be plain that is rotated around the z-axis, thus forming a torus. An example of a vertex with four (`mp.Vector3`) points in 2D cylindrical is given in Listing 2.3, whereby the first and the last `Vector3` point in the `vertices` list will be connected.

```
vertices = [mp.Vector3(0,0,0),
            mp.Vector3(0,0,1),
            mp.Vector3(1,0,2),
            mp.Vector3(1,0,0)]
geometry = [mp.Prism(vertices,
                     axis = mp.Vector3(0,1,0), # direction of the prism
                     height = 0, # height of the prism
                     material = mp.Medium(epsilon = ..., D_conductivity = ...))]
```

Listing 2.3: Construction of an mp.Prism GeometryObject

This, for instance, can serve as a small building block that can approximate a small segment of some terrain, where the points represent tangent points of some extracted terrain profile. Many concatenated prisms then form the overall ground profile. The big advantage over the user material function is, that one can make use of the epsilon averaging, which will smooth out the stair steps inherent to any FDTD implementation. This in turn improves the accuracy of results in the presence of objects exhibiting some curvature (like hilly terrain). Another method is to initialize only one large prism with all vertex points that make up the surface of some structure. Due to the very long sub-pixel (epsilon) averaging time and larger RAM consumption, while yielding equal results as the previous prism method, the latter prism approach should be disregarded.

2.2.10 Functionality of MEEP

In this section the most important features and functions used in the simulations will be discussed.

1. Run function: The `Simulation` class method `sim.run(*arg_list, **keyworded_args)`. By means of the `*arg_list`, step functions (see next point) can be introduced. They will be called by functions like `mp.at_beginning(fct)` or `mp.at_every(time=..., fct)` which tell the simulation, when the step functions shall be exactly executed. One of the `keyworded_args` normally used is the "until" keyword, which determines, how the MEEP simulation it timed. It specifies the termination of the simulation when the sources are declined below a pre-defined threshold, but also for an absolute simulation time give in MEEP time units.

2. Step functions: As shown in the previous point, they are given to the `sim.run(...)` method and are executed at certain single or periodic time instances. Most commonly they are utilized for adding or removing sources, changing the material properties and location (e.g. for describing moving media), reading out material, field, flux or other values, and other purposes. A few examples of step functions are:

  - Outputting: Commonly used functions are for example `output_epsilon` which is normally done at the beginning (`mp.at_beginning(mp.output_epsilon))`) or field readouts such as `mp.output_efield_z(...)` which are usually called periodically every time increment dt as `mp.at_every(dt, mp.output_efield_z(...))`. Further examples of step functions are: `mp.integrate_field_function` and `mp.max_abs_field_function`. They evaluate the integrals over or the maximum absolute field value of a certain segment or area.
  - User specified functions in the form `user_fct(sim)`. MEEP need to have the simulation instance as argument in these functions. They can be used for example to sample values, manipulate objects and current sources, etc. With sim.time() the current MEEP time can be retrieved in such a function.

3. Subpixel averaging: In the publication that describes the functionality of MEEP ([29]), it is explained, what the sub-pixel averaging technique is, what purpose it has and how it is implemented. It works most efficiently with primitives, such as prisms presented in the previous section, but also, although being very slow, with material functions.

4. Grid interpolation: In MEEP, if a source is specified at a point between grid points, the source is distributed according to an inverse bilinear (2D) or trilinear (3D) interpolation, hence it is distributed across the neighboring grid points. Accordingly, a field value at a certain point is found as the bi- or trilinear interpolated value of the neighboring grid points' field values.

2.2.11 Possibilities and Limits

This section compiles the pros and cons of the software package MEEP. The **drawbacks** or limits are:

- Complex code base: Since the early days of the project, at least 15 years of development have passed. The complexity has increased enormously with the amount of features, parallelization, etc. Extending or customizing it for special needs is very difficult and needs expert knowledge. A self-developed FDTD implementation may be more advantageous in special cases.
- Dimensionless: As discussed earlier in that chapter, MEEP calculates with dimension-free quantities and fields. The conversion of those to (field) quantities and parameters with units can be tedious since it needs to be derived for every unit individually. This is an advantage in photonics but can be counted as a disadvantage in fields of research where dimensions are of main interest.
- Sources: In lightning research, current sources are often specified in terms of H_x and H_y (or H_ϕ) fields, which are directly calculated from the current I (Amperes law) and impressed in the closest surrounding grid points (see [21], Eq. 3.35). In MEEP one is confined to use current sources M (e.g. `mp.H_y`) and J (e.g. `mp.Ez`). One has to be aware which source component will lead to the correct desired field. In the application of MEEP for lightning research, `mp.Ez` is sufficient[6].
- Uniform (cartesian) grid: The mesh is only uniform. Rectilinear or curvilinear meshes are not implemented.

[6]It is important to note that the components `mp.Hy` or `mp.Hphi` *shall not be confused* with the corresponding hard sources H_y or H_ϕ which are often calculated by Amperes law and used to represent a channel current ([21], Eq. 3.35). Instead, they are soft current sources applied to Eq. 2.7!

- Bugs: Unexpected behavior due to bugs is always possible in a large software project. Without the necessary experience, it is hard to discover possible bugs. The large amount of features makes it likely that many bugs are still undetected.

As **advantages** can be named:

- Easy entry: The entry barrier to FDTD is quite low. Installing MEEP, or parallel MEEP, via `conda` installer, within some minutes one has a ready-to-use FDTD solver.
- Good documentation: In the docs ([22]), many use cases are covered with example scripts ready to execute. The problem with respect to fields outside of photonics is, that there is rather little information available.
- Python & C++: Since late 2017 simulations are fully scriptable in Python. The previous interfacing language Scheme was rather sophisticated. C++ can also be used.
- Anisotropic dispersive materials: As mentioned in [21], dispersive materials can be implemented into an FDTD algorithm with high effort, hence in FDTDs for lightning research seldomly encountered. Since 2012, anisotropic dispersive materials are fully supported along with appropriate perfectly matched layers adjacent to the material.
- PMLs: Perfectly matched layers (PMLs) can be implemented by one line of code. They are customizable with additional parameters.
- Sub-pixel averaging: This special subgridding technique increases accuracy at coarser spatial resolution ([29]). Unfortunately for user defined materials (see Listing 2.2) this averaging technique is turned off by default due to long computation times of the averages by means of numerical quadrature (integration). If objects following some custom functions in combination with sub-pixel averaging are needed, the problem can be mitigated by using prism primitives, as shown in Listing 2.3.
- Parallelization: MEEP comes out of the box with a parallelization by using distributed processing and memory by the MPI (Message Passing Interface) standard.
- Amazon Web Services (AWS): The Simpetus platform in the interface to perform MEEP simulations in clusters of multicore units that can be rented via AWS. This way of utilizing MEEPs parallelization feature can increase the performance, respectively lower the computation time, of large problems considerably by exploiting top-notch technology deployed by AWS. Also here a variety of useful (video) tutorials can be found on the web (for example [32]–[34]).
- Runs on Windows© 10 by utilizing the Ubuntu App, which runs on the basis of the Windows-Linux subsystem. It can be installed directly from the Windows Store. That way users can run simulations on this platform with a low entry barrier.
- Publications: A lot of scientific work has already been done and published based on MEEP.
- Community: A big community exists which interacts via mailing list. Many questions will be answered, even if they are rather basic. The mailing list archive, though, is rich of answers to commonly asked questions in addition to the FAQ in the docs [22].

2.2.12 Known issues and bugs in MEEP versions

In the MEEP Github repository ([27]) under 'Issues', a list of bugs and their fixes can be inspected. One important bug that was found in the course of this work was related to the user material function of Listing 2.2 in section 2.2.9.1. Whenever a conductivity σ (`D_conductivity`) is specified as a material parameter in a user defined material function, the conductivity gets ignored in case that the conductivity of the other material (in the if-else clause) is *zero* (such as in `mp.air`) *AND* the threshold is compared to a number smaller or equal a negative number. In this case, D_conductivity is zero and only a permittivity ε_r remains, making the material transparent instead of lossy. A quick fix is to specify a D_conductivity of `1e-20` and `varepsilon=1` instead of selecting `mp.air`. At the time of writing, the bug has been reported (issue #892) together with some example code for reproduction, but has not yet been fixed.

Chapter 3

Application of MEEP for lightning research

All simulations in that thesis were performed with 2D cylindrical symmetry FDTD. The ground profiles are therefore rotationally symmetric objects around the origin. Any occurring reflections will only exist in radial direction. In a 3D scenario, this represents objects that are infinitely extended into the y-axis and the propagating field caused by lightning has plane-wave character with 1/r decay. The possible influence of that special geometry has to be considered in the results. For the following simulations, a standard resolution of 4 m was chosen. For a MEEP unit of 1 km this means a resolution of 250 grid points per MEEP unit. This is still feasible for a cell of 120 km × 12 km (effectively 121.5 km × 15 km including PMLs of thickness 1.5 km), with a total simulation time of roughly 24 h. Detailed performance analyses are given in a dedicated subsection at the end of this chapter. If not explicitly stated differently, the return stroke model MTLE (Eq. 1.3) was used with Heidler type base currents, as given in Eq. 1.4, with parameters listed in Table 3.1 for all simulations with MEEP throughout the thesis. The corresponding plots of the return stroke currents are shown in Fig. 3.1. Only one source per grid point is used, resulting in 250 current sources per unit. This appears to be sufficiently smooth for remote field simulations, see analysis and plots in Appendix B.2.1. Due to the MEEP's linear interpolation property, even several current sources could be distributed between Yee grid points. This is potentially necessary for calculation of close electric fields.

Table 3.1: Heidler current parameters for first RS (peak current about 30 kA) and subsequent RS (peak current about 12 kA).

	I_1 (kA)	τ_{11} (μs)	τ_{12} (μs)	n_1	I_2 (kA)	τ_{21} (μs)	τ_{22} (μs)	n_2
first RS	28	1.8	95	2	–	–	–	–
subsequent RS	10.7	0.25	2.5	2	6.5	2	230	2

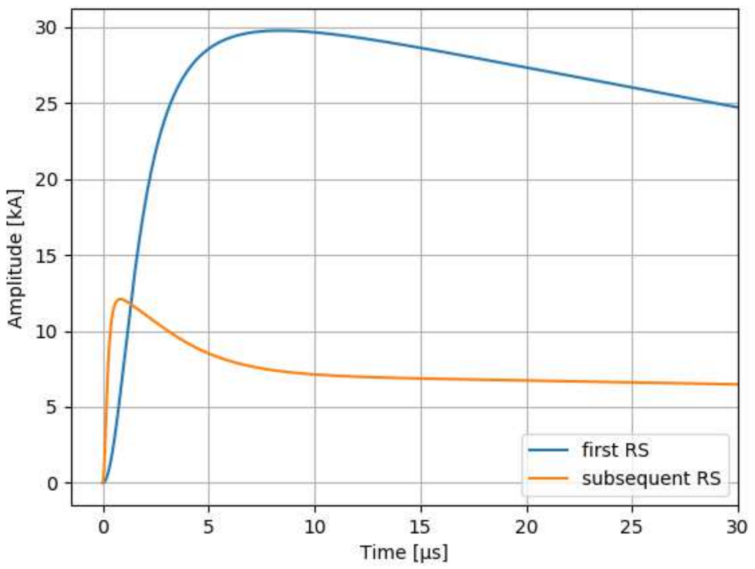


Figure 3.1: Heidler current functions

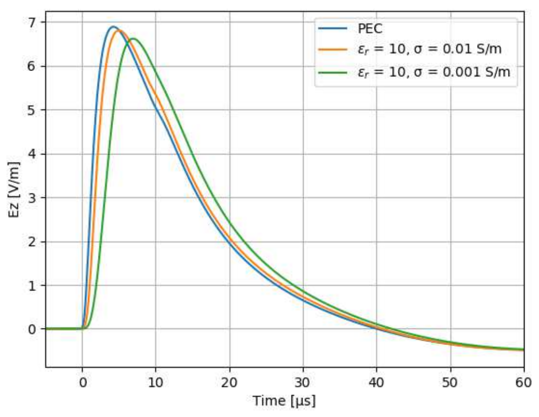
3.1 Theoretical scenarios

In this section theoretical scenarios are simulated. The simulations were performed for RS parameters in Table 3.1 for flat and harmonically changing ground with perfectly electric conducting (PEC) and lossy material with constant dielectric permittivity $\varepsilon_r = 10$ and conductivities of $\sigma = 0.001\,\mathrm{S/m}$ and $\sigma = 0.01\,\mathrm{S/m}$. The results were taken $0.005 \cdot a_{MP}$ above ground, which means 5 m. The reason is, that the permittivity does not change abruptly in the transition from the material to air, but gradually until it reaches its correct physical value $\varepsilon_r = 1$. Therefore a sample directly at the border of the two materials will lead to wrong E-field results, since MEEP scales the D-field from the grid with an average permittivity by $E = D \;/\; \varepsilon_r$. Details about the material behavior and averaging are provided in the Appendix B.1 and B.2.3.3.

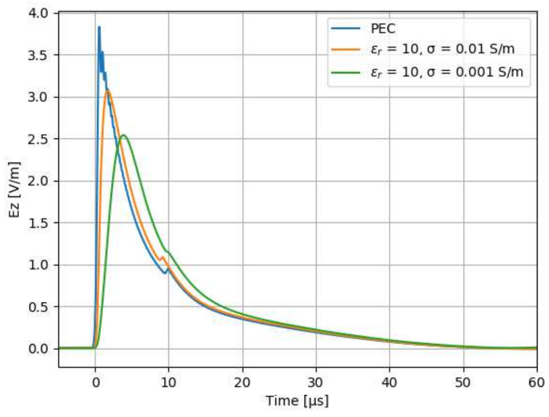
3.1.1 Flat ground

To begin with a simulation that serves as a reference for more complex ground structures, flat ground shall be considered first. In Fig. 3.2a and 3.2b the waveform of a first RS and subsequent RS at a distance of 100 km are depicted. In comparison to the first RS, a stronger influence of the finite ground conductivity on the amplitude can be observed for the case of the subsequent RS. The higher frequency components are attenuated stronger, which in addition leads to longer rise times. This relation can be verified by the Norton approximation method [35] where a field propagating over lossy ground can be interpreted as a low-pass filtered field of a wave traveling along perfectly electric conducting (PEC) ground. The filter parameters are determined by the distance, relative permittivity and electric conductivity. In the graphs it is obvious that for both cases with an increasing value of conductivity, the field approaches the PEC case. In the subsequent RS simulation, Fig. 3.2b, the high frequency components lead to an artifact called 'numerical dispersion' for PEC. It becomes apparent in form of ringing. The higher the resolution is chosen, the smaller the ringing will become [1].

[1]But at the same time the problem size (RAM) grows quadratically (2D)or with the 3rd power (3D) and the simulation time even with about one order more. Further, choosing an inappropriate resolution, e.g. 1/300 (3.33 m) in spite of 1/250 (4 m), the ringing will become higher even though the resolution was increased. This phenomenon was observed, but has not been shown mathematically in the course of the thesis.



(a) First RS



(b) Subsequent RS

Figure 3.2: Fields in a distance of 100 km for flat, perfect electric conducting (PEC) ground and ground with finite conductivity.

3.1.2 Periodic ground

In this section a periodically changing ground surface was implemented by means of a cosine with period 10 km and 5 km and a peak-to-peak height of 100 m and compared to flat ground, which is depicted in Fig. 3.3. The motivation behind this was to investigate a possible effect of rough ground with different spatial variability and conductivity.

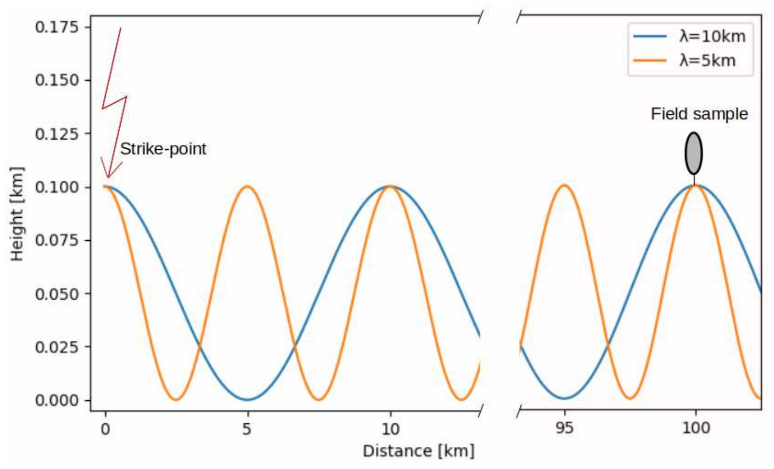
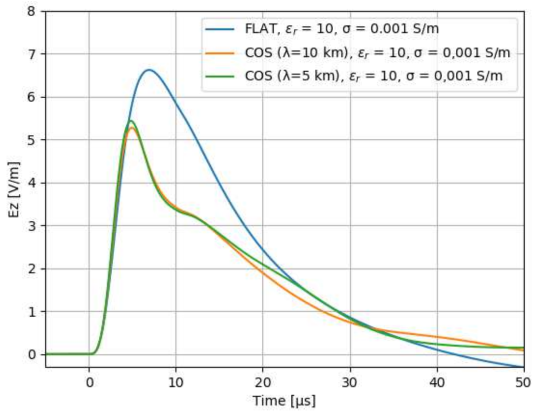
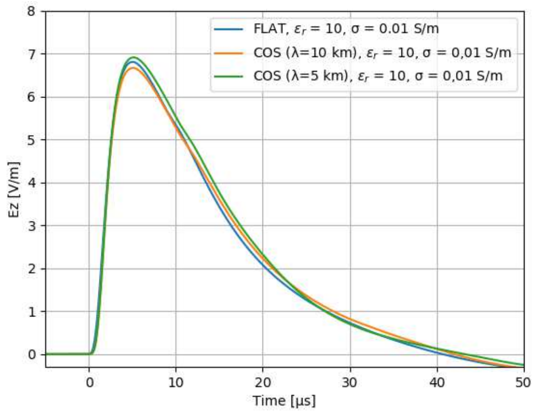


Figure 3.3: Terrain profile of periodic ground for a spacial wavelength of 5 km and 10 km

This was done for first RS and subsequent RS with $\sigma = 0.001\,\mathrm{S/m}$ and $\sigma = 0.01\,\mathrm{S/m}$. The results for the first RS are plotted in Fig. 3.4. In Fig. 3.5a, a reduction of the amplitude and a slight change in the signature can be observed. For higher conductivity, Fig. 3.5b, the change in the amplitude is again negligible.



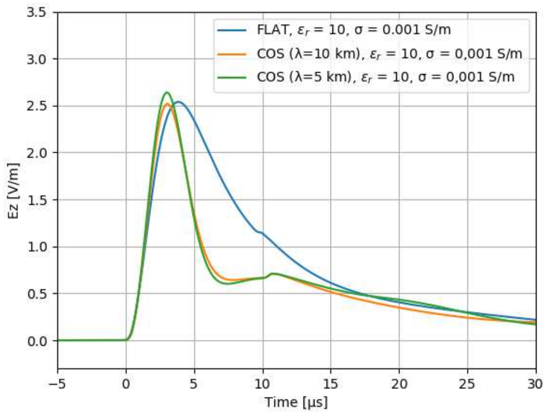
(a) $\sigma = 0.001\,\mathrm{S/m}$



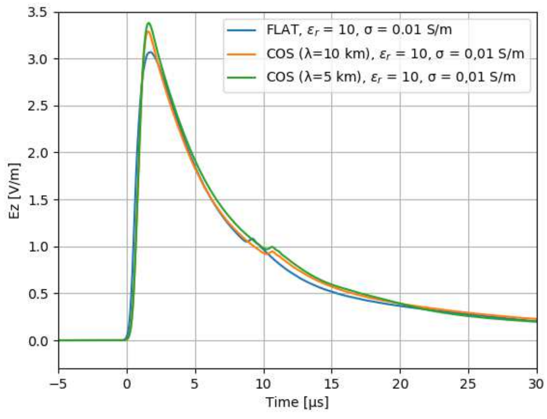
(b) $0.01\,\mathrm{S/m}$

Figure 3.4: First RS in a distance of 100 km for cosine shaped ground with periodicity of 10 km and 5 km and flat ground for finite conductivity

The results for the subsequent RS is shown in Fig. 3.5. Here the conductivity has dominantly an effect on the signature, meanwhile the influence on the amplitude is negligible for both conductivities.



(a) $\sigma = 0.001\,\mathrm{S/m}$



(b) $0.01\,\mathrm{S/m}$

Figure 3.5: Subsequent RS in a distance of 100 km for cosine shaped ground with periodicity of 10 km and 5 km and flat ground for finite conductivity

There are various possible setups with respect to the combinations of the parameters λ, σ, peak-to-peak height, strike-point on top or in a valley of the hills, sampling point on top or in a valley of the hills, etc. Hence, interpretations have to be taken with care.

3.2 Practical scenarios

In the practical part, models of the terrain along the propagation path from the Gaisberg (GB) mountain (Salzburg, Austria) to Neudorf (ND) are established. This model, further on called GB-ND, will be simulated both without and including a tower model implementation. The goal is to simulate real world scenarios with real measured currents and to obtain findings about the effect of the mountain and the tower, respectively. Further these simulations are compared to remote real E-fields recorded at a distance of 108.8 km in Neudorf. If not stated differently, the currents that were used for the simulations comply with a real measured current at the GB. The plots are shown in Fig. 3.6. The real current measured at the top of the Gaisberg tower by means of a shunt ([36]) was filtered with a 1 MHz low-pass filter. Further, the return stroke speeds were chosen to be primarily $v_{RS} = 1.5{\cdot}10^8$ m/s complying with the dominantly used value in literature. In the publication [37] a RS speed of $v_{RS} = 1.2{\cdot}10^8$ m/s can be found. This is motivated by the fact, that in the EUCLID network this value is used in the transmission line model (TLM) for the estimation of the stroke peak current.

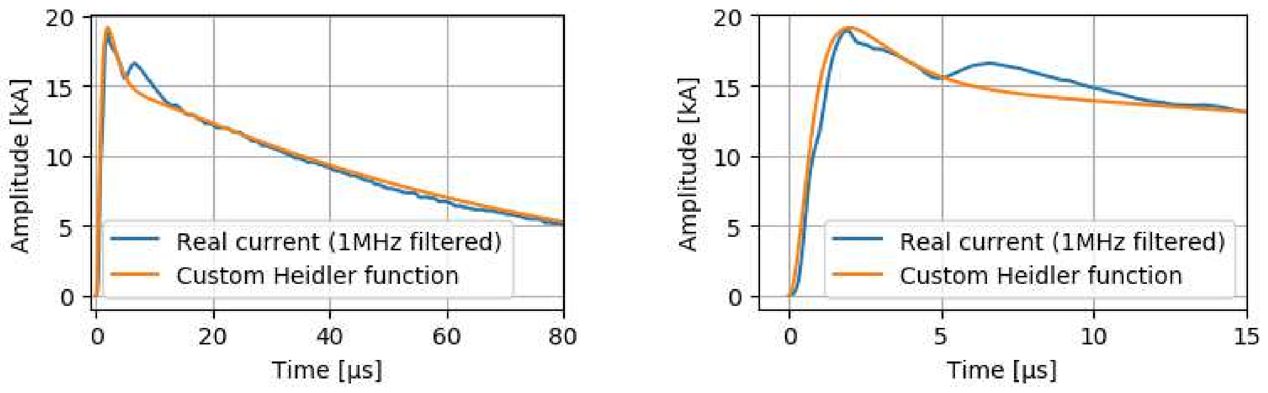


Figure 3.6: Real measured vs. custom Heidler current (left: zoomed, right: extended plot), used for simulations in section 3.2. The custom current consists of two Heidler type terms with parameters $I_1 = 15\,\text{kA}$, $\tau_{11} = 1.1\,\mu\text{s}$, $\tau_{12} = 1.7\,\mu\text{s}$ and $I_2 = 13\,\text{kA}$, $\tau_{21} = 2\,\mu\text{s}$, $\tau_{22} = 70\,\mu\text{s}$.

For the sake of comparison of later results, the resulting fields are simulated for flat, lossy ground. The fields in a distance of 109 km (which is approximately the distance GB-ND) are depicted in Fig. 3.7. The field signatures of the real measured current and the fitted custom Heidler function are in good agreement. The custom Heidler can be used for speed optimization, because analytical expressions, as custom Heidler, are evaluated faster by MEEP, see 3.3.

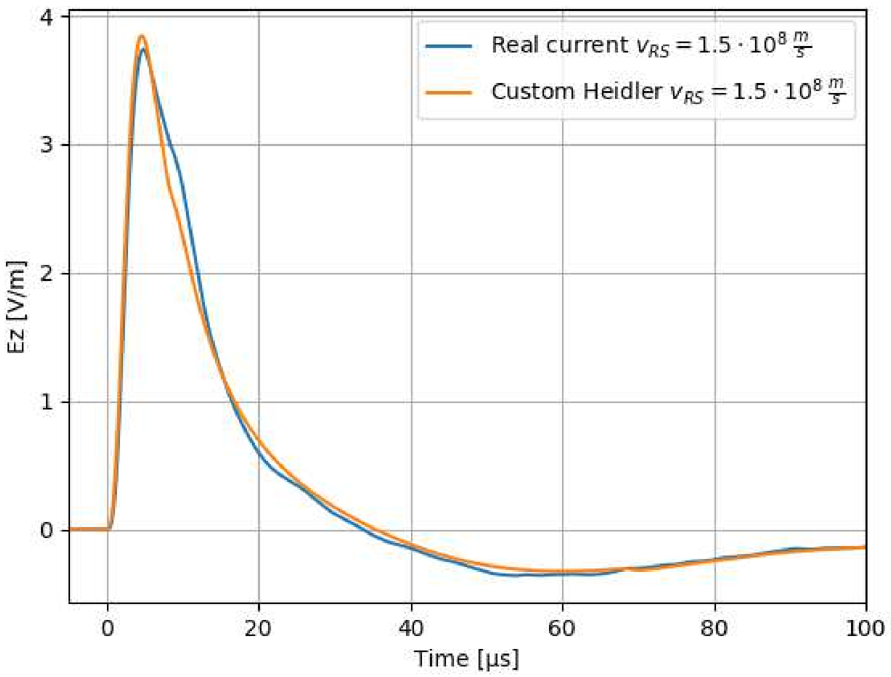


Figure 3.7: Vertical E-field over flat, lossy ground, $\varepsilon_r = 10$, $\sigma = 0.001\,\mathrm{S/m}$ with $v_{RS} = 1.5{\cdot}10^8\,\mathrm{m/s}$ in 109 km distance.

3.2.1 Terrain model

In this chapter, results of EM-fields caused by lightning discharges over a real terrain profile for some geographical area are presented. This terrain model considers lossy ground with $\varepsilon_r = 10$ and $\sigma = 0.001\,\mathrm{S/m}$. For the terrain model, the height above sea level of Austria was obtained from `https://data.gv.at` ([38]). The geo.tiff file contains a grid of $10\,\mathrm{m} \times 10\,\mathrm{m}$ tiles with one associated numerical value each, representing the average height above mean sea level at each grid cell. The extraction of the height profile over the line-of-sight was done with the open-source geographic information system QGIS ([39]), more specifically by utilizing the `Profile tool` plugin ([40], [41]). It can be downloaded and installed from the official QGIS repository accessible over the QGIS extensions menu. After loading the geo.tiff file into the `Profile tool` and connecting the points by marking them, the profile along the connecting line was calculated[2].

Further, the data table was exported as a .csv file, which in turn could be imported into `Python` with the `csv` library. At this point, the data is available as an array of tuples `(distance, height)`. Those points can then be fed into MEEP as prism primitives or an interpolation function has to be employed, which then will serve as the terrain model in MEEP, utilizing a coordinate-based material function. This was done by the `Python` module `scypi.interpolate`, using the function `interp1d(x,y, kind='cubic')`. The latter is a one-dimensional interpolating function that interpolates with a polynomial of 3rd order. It accepts as arguments an x- and y-array (vector) of same length and returns an object, stored as `terrain_fct`, that can be called like a function (see also Listing 2.2). It is fed with a distance value p, which is provided by MEEP, as it initializes the geometries and materials at the beginning of the simulation:

```
ground_level = interpolate.interp1d(distance_data, height_data, kind='cubic')
height = ground_level(distance)
```

[2]The complete terrain profile extraction was added to the Appendix B.2.2 (Fig. B.4).

The interpolated version `ground_level(p)`, which is used in the user defined `material_fct`, is plotted in Fig. 3.8. Three terrain functions were used, the full terrain height profile, an averaged terrain profile, obtained by filtering the original terrain profile with a running average of 2 km and a version with the Gaisberg only followed by flat ground. These profiles are shown for the first 15 km. A custom Heidler current was used to simulate the E-fields. At the beginning of the simulation, this computationally more expensive material function representing the terrain model is initialized by MEEP. The process (in the console outputted as 'subpixel-averaging') takes up to about half an hour. The specific simulation setups and their results are discussed in the following two subsections.

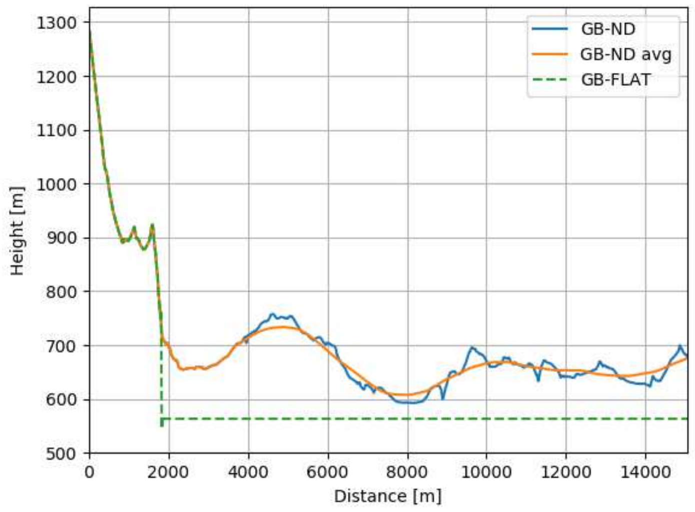


Figure 3.8: Terrain profiles used in simulations. Gaisberg-Neudorf (GB-ND), GB-ND avg (2 km gliding average), GB only with consecutive flat ground only (GB-FLAT).

The influence of 2D cylindrical symmetry, where the full terrain is a torus around the origin, on the results is neglectable, since the lightning strike point is the highest of the overall terrain and no tall objects are in line-of-sight to the observation point. The results are shown in Fig. 3.9. It can be seen the terrain enhances the field in comparison to flat ground significantly. On the one hand the strike to the mountain top leads to the enhancement of the field peak. On the other hand, the difference between the results of the averaged terrain model and the original terrain model is not big. The local terrain structure where the field is sampled influences the scaling of the waveform, where the averaged version shows less field enhancement than the unaveraged hill at 108.8 km[3]. The averaged terrain profile and GB followed by purely flat ground are almost equal. This suggests, that the strike to the mountain top causes the biggest part of the field enhancement rather than the local terrain elevation.

3.2.2 Fields without tower

In this chapter, simulations of the strike to the mountain without tower are now compared to real measured remote E-field, corresponding to that measured current. The E-field measurements were performed on top of a building in Neudorf in a distance of 108.8 km. For this building a field

[3]Though, special care has to be taken when using the terrain model, which will exhibit a stair-stepping approximation of the ground structure. One will get severe scaling errors in the fields when reading out at locations close to corners of a step. This is treated in the Appendix B.2.3, where a plot of the terrain model together with the maxima of the return stroke associated fields can be found. It compares the sensitivities of original, averaged and flat terrain profiles along with the stair-stepping effect.

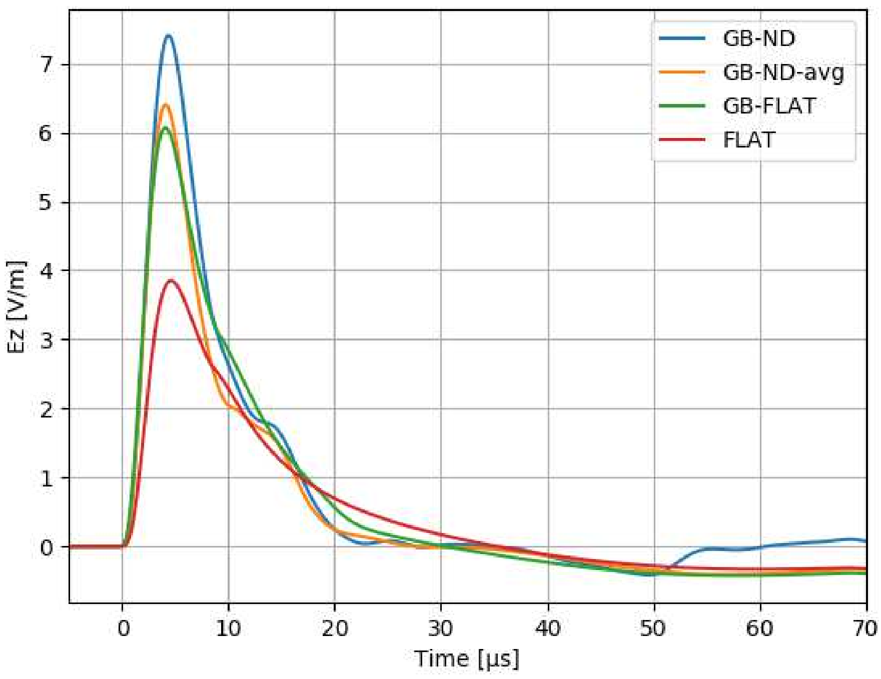


Figure 3.9: Results of simulations with the original (GB-ND), averaged (GB-ND-avg) and mountain followed by flat ground (GB-FLAT) terrain profiles from Fig. 3.8. The ground parameters are $\varepsilon_r = 10$ and $\sigma = 0.001\,\mathrm{S/m}$. The RS current according to the custom Heidler function in Fig. 3.6 with $v_{RS} = 1.5{\cdot}10^8\,\mathrm{m/s}$ is used. For comparison, flat lossy ground (FLAT) was included.

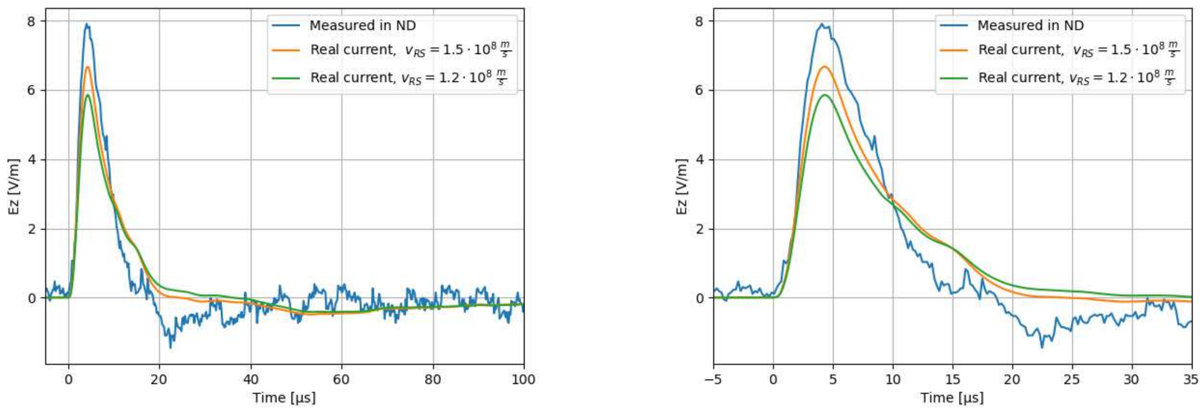
enhancement factor of 2.5 was determined. The measured remote E-field was therefore scaled down by this value. The results for the case of the averaged terrain profile Gaisberg to Neudorf ('GB-ND avg') can be seen in Fig. 3.10. The results compared with the real measured remote field in Neudorf, show good agreement with respect to the field signature. The amplitudes of the simulated fields are slightly smaller than observed in the measured fields. If the zero crossing occurs or not depends on the used RS model (i.e. MTLE, which shows zero crossing contrary to the TL model), but also a dependence on the terrain is suspected[4]. In Fig. 3.10, the zero crossings can be identified at about 40 μs after the field onset, similar to the flat ground. The zero crossing is shifted to the right for slower RS front velocities v_{RS}.

3.2.3 Fields with tower

The model of the previous section is now extended by a tower on top of the Gaisberg. The implementation of the tower model is accomplished by means of a current source distribution considering reflected wave fronts along the tower surface ([42]).

$$\begin{aligned} I(z,t) &= P(z-h)i_0\big(h, t - \frac{z-h}{v*}\big) - \rho_t i_0\big(h, t - \frac{z-h}{c}\big) \\ &\quad + (1-\rho_t)(1+\rho_t)\sum_{n=0}^{\infty} \rho_g^{n+1}\rho_t^n i_0\big(h, t - \frac{h+z}{c} - \frac{2hn}{c}\big) \quad \text{for} \ \ h < z < H_0 \qquad (3.1) \end{aligned}$$

[4]This is not fully clear at the time of writing and has to be further investigated by a more detailed analysis of terrain models.



(a) Extended (b) Zoom

Figure 3.10: Results of simulations for an averaged terrain profile (GB-ND-avg) with $\varepsilon_r = 10$, $\sigma = 0.001\,\mathrm{S/m}$. RS currents: real measured current and custom Heidler, as shown in Fig. 3.6 for (a) extended and (b) zoomed waveform.

$$I(z,t) = (1-\rho_t)\sum_{n=0}^{\infty}\Big[\rho_t^n\rho_g^n i_0\big(h, t - \frac{h-z}{c} - \frac{2hn}{c}\big) + \rho_t^n\rho_g^{n+1} i_0\big(h, t - \frac{h+z}{c} - \frac{2hn}{c}\big)\Big] \quad \text{for} \quad 0 \le z \le h \tag{3.2}$$

where the channel height is H_0, the tower height is h, ρ_g and ρ_t are the ground and top reflection coefficients of the tower.

The implementation of the above equations is given in the Appendix A.2. The model was also validated (see Appendix B.2.4) as being precise enough to yield usable results with respect to the peak amplitude enhancement that it will cause.

The Gaisberg tower has a height of 100 m, but the reflection coefficients of the wave fronts at the tip and bottom of the tower are unknown. Therefore the values from literature from the 168 m tall Peienberg tower in Germany (see [42]) are used. In another simulation, the reflection coefficients at the top and the bottom of the tower‘ were both set to 0 in order to see the effect of just one faster downwards traveling wave along the tower.

The result of the latter setup shows a slight field enhancement in comparison to the model without the 100 m tall object. Thus it is almost in perfect agreement with the amplitude observed in the measured remote E-field, which can be seen in Fig. 3.11. A too high field enhancement in comparison to the measured remote field was obtained by the simulation with reflection coefficients determined for the Peienberg tower. The peak value is almost three times as high. In addition, no zero crossings appear in the case of a tower, which is an observation that can also also be found in literature (see [42]).

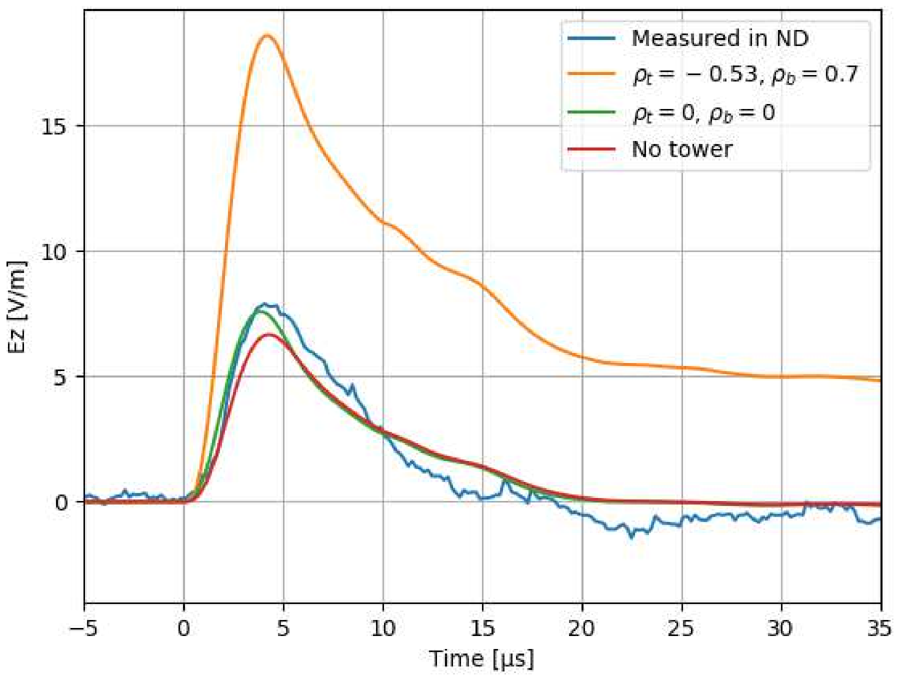


Figure 3.11: Tower models for real measured current sources and return stroke speed $v_{RS} = 1.5{\cdot}10^8$ m/s. Ground parameters like in Fig. 3.10. For comparison the terrains model GB-ND-avg without tower was included (red).

3.3 Performance

The performance of an FDTD simulation with respect to total computation time depends strongly on the components of the simulating system and the implementation of the algorithms from a computer architectural point of view. Most importantly, the performance depends on the following components: 1) CPU speed, 2) the speed and bandwidth of the random access memory (RAM) (GB/sec) and 3) the CPU cache size. While the CPU speed is responsible for working off the instructions, like arithmetic calculations, the RAM stores all the field values and material parameters (all as double precision in MEEP). A large bandwidth and speed of the RAM is necessary for avoiding bottle necks when data has to be retrieved from it. In the simulations of that thesis a maximum of about 26 GiB were observed in the simulations. The RAM is nowadays the limiting factor in finite-difference solvers, since it is considerably slower than the cache (L1, L2, L3) located close to the CPU. The cache usage is important, since data that are used often shall be stored there for avoiding unnecessary fetches (due to so-called 'cache misses') from the rather slow RAM. In the range of what is possible, MEEP has implemented its update algorithm loops, such that field values that serve as results for consecutive field calculations will be stored as closely as possible to reach cache locality (i.e. within the same computation loop).

3.3.1 Results

Since the algorithm, the CPU, RAM and cache determine the performance of an FDTD simulation, the problem size influences the total simulation time the most (doubling up the resolution in 2D leads to 2^2 times as much RAM demand and 2^3 times the simulation time). The cache is suspected to be the limiting factor if two or more simulations are runing in parallel. For two parallel simulations for instance it was observed, that the duration of a MEEP time step Δt_{MP} of both simulations nearly doubled up when a second simulation is started. This was independent of the number of cores that were spent for that process. A speed comparison of total simulation time and seconds per time step Δt_{MP} of a Amazon Web Services (AWS) instance vs. PC is given in Table 3.2. A test with a 'c4.8xlarge' AWS instance, that can utilize 36 cores, revealed that using this amount of cores did not increase the performance in this example (flat, lossy ground with real measured and interpolated currents as sources). With 36 cores the simulation is slower compared to 16 cores. Thus, the acquisition of an 'c4.8xlarge' AWS instance is not advantageous. Further, 8 cores performed equally well, if not minimally faster than 16 cores in the 'c4.4xlarge' AWS instance. This can be due to the special simulation structure (objects, PMLs, etc), where unnecessary splitting of the simulation cell into chunks just increases the overhead of chunk communication. For AWS simulations, the related costs for the flat ground simulation were roughly 10 EUR for an unclustered (single) 'c4.4xlarge' AWS instance at regular price.

Table 3.2: Simulation times of flat ground (121.5 km x 15 km) on a PC (Intel® Core ™ i7-8700, 4 GHz, 12MB cache) vs. AWS, 16 cores c4.4xlarge.

	PC, 1 core	PC, 4 cores	AWS, 16 cores
	(100%)	(63%)	(44%)
Total time	30.2 h	18.9 h	13.2 h
Per step (Δt_{MP})	2.08 s	1.3 s	0.91 s

A simulation of a harmonic lossy material function (see 3.1.2), which is comparable to the terrain model in its computational complexity, using 4 cores on a 6-core Intel CORE i7-9850H vPro (9th Gen) with 4.6 GHz in turbo mode and 12 MB cache took 17.5 h, which is slightly better than the PC performance results in Table 3.2.

Table 3.3: Simulation times of terrain model (121.5 km x 15 km) with analytical (custom Heidler) vs. interpolated currents sources (from .csv file) on a PC, 4 cores Intel® Core ™ i7-8700, 4 GHz, 12MB cache.

	Analytical current	Interpolated current
Terrain model	20.6 h	24.3 h
Per step (Δt_{MP})	1.42 s	1.68 s

When real measured current sources from .csv sheets were used, the construction and evaluation of the interpolated sources consumes more time than the evaluation of an analytical function. The difference in the total simulation time can be seen in Table 3.3. When further a lot more current sources need to be initialized, as for example a tower model demands (reflected currents are inserted as additional current sources), the simulation time using interpolated currents can easily extend to 72h.

RAM consumption for the terrain model (121.5 km x 15 km) with a user material function is approximately 26 GiB, whereas the prism model demands only about 16 GiB. This is due to a more efficient material parameter storage of `GeometricObject` primitives in comparison to custom objects. Since the prism model is more efficient while retaining the possibility of fast epsilon averaging, it should be favored over the user material function.

Chapter 4

Summary and outlook

4.1 Summary

At the beginning of that thesis, a short introduction to the atmospheric physics behind the phenomenon of lightning discharges guides the reader from charge separation over the initiation of discharges (initial breakdown and stepped leaders) till the return stroke (RS), which is the most relevant event for this work. The engineering RS model, which serves as the basis for lightning electromagnetic simulations, is elaborated along with the mathematical description of the RS current types, which are often used.

After that, the reader is introduced to the basics of electromagnetic field propagation, most importantly the governing Maxwell equations (1.5) - (1.8). The mathematical approach to solve these equations analytically is treated first and some applicable, important solutions in the field of lightning research are mentioned. Once analytical solutions are no more obtainable due to the complexity of the conditions, numerical methods like FDTD, Method of Moments and others are utilized to yield results for more general electromagnetic propagation environments.

The Finite Difference Time Domain (FDTD) method is derived in its basic form and elaborated with respect to the appliction for lightning electromagnetic simulations in chapter 2, where stability considerations form an important part. Section 2.2 introduces the software package MEEP (MIT electromagnetic equation propagation), which is a full, advanced implementation of an FDTD algorithm, which can be applied to simulating lightning electromagnetic fields. The demonstration of its applicability was the goal of that thesis.

After explaining how MEEP internally works and how it is interfaced by means of the programming language Python, in section 3 MEEP is applied to theoretical and practical scenarios. Some simple theoretical situations are given for flat and rough (periodically changing) ground with different ground parameters (permittivities and conductivities). These theoretical scenarios are followed by practical ones, where terrain height profiles are simulated for real measured currents (and mathematical approximations thereof) and related remote E-fields measured with a plate antenna in a distance of about 109 km. Given the limited parameters space and assumptions made for the simulation, the resulting waveforms from the simulation of a lightning strike to the Gaisberg (Salzburg, Austria) are in fairly good agreement with the measured remote E-field. In addition, a tower model was also implemented and tested for its influence to the peak value and signature.

The Appendix A comprises Python code listings of the used simulation scripts and classes and Appendix B gives further detailled information about terrain models and stair stepping.

4.2 Outlook and further work

MEEP is an ideal tool for the re-calculation and verification of existing FDTD implementations of regular complexity on the one hand. On the other hand it can also be used for exploring new electromagnetic environments that are affected by lightning electromagnetic pulses (LEMPs). A list of further ideas to be elaborated, that would have been too extensive for the scope of that thesis, is:

- 3D simulations: It is necessary to check how to implement terrain by means of prisms or user material functions the best way in 3D. Since the prisms can only be constructed by planes perpendicular to the axis, some trade-offs between stair-stepping and smooth height transitions on the surface of a 3D terrain model will be necessary.
- Lumped elements (resistance, inductance, capacitance) that influence the local amplitude and phase shift of the field corresponding to the element. These elements are substantial when phenomena like coupling of fields to power transmission lines are investigated. They must be implemented as current sources, since hard sources are not feasible in MEEP without manipulating the core code of MEEP. But since the field update coefficients in the update equations are changed substantially, the possibility to implement lumped elements in MEEP by means of current sources alone has to be investigated.
- Equivalent source current model, corresponding to some previously monitored fields at a certain distance, which can be injected closer to the observation point and reproduces the same expected fields (or good approximations). That way, computation times can be reduced crucially. The existence of such an equivalent model has to be shown.
- Non-linear and dispersive materials: Due to the support of those materials in MEEP, electromagnetic environments containing these materials can be investigated.

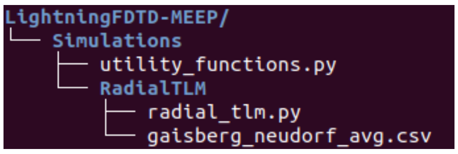
Following the open source philosophy, the author hopes, that this work will contribute to a progress towards a more uniform collaboration with respect to FDTD in lightning research, which so far appears somewhat scattered over many institutes around the world. The ideology behind this thought is to make the entry barrier lower and to significantly accelerate the process of gaining scientific knowledge and findings in lightning research.

Appendices

Appendix A

Python Code listings

In this chapter, a code base consisting of two scripts is listed. By means of those, a MEEP lightning EM-field simulation can be run. The first script named `radial_tlm.py` contains the setup of the simulation and `utility_functions.py` contains all necessary classes that construct the source current models. The two files are contained in folders corresponding to a directory as shown in Fig. A.1.



```
LightningFDTD-MEEP/
└── Simulations
    ├── utility_functions.py
    └── RadialTLM
        ├── radial_tlm.py
        └── gaisberg_neudorf_avg.csv
```

Figure A.1: Simulation folder tree

A.1 Core script: radial_tlm.py

The functionality of the script and generally procedure of setting up a simulation can roughly be summarized as follows:

1. Find the resolution, courant factor, cell size and simulation time, which is appropriate for your simulation in order to be stable and feasible in a reasonable simulation time.
2. Pass (read and parse) all relevant parameters to the script, either from the console directly or a text file, if not hard coded.
3. Construct the material objects by means of Blocks, Prisms or user material functions in the right simulation dimension and add them to a list.
4. Add the PMLs to a list.
5. Construct the sources and add them to a list.
6. Initialize the simulations with all the constructed lists of sources, materials and PMLs together with the parameters from 1. by setting up a dictionary (a Python object) `sim_config` that is passed at the `Simulation` instantiation to the class as `sim = mp.Simulation(**sim_config)`.
7. Decide which step functions shall be used (that are either called continuously with `mp.at_every(...)` or just at the beginning, using `mp.at_beginning(...)`), in order to output fields, etc. Include them into a `arg_list`.
8. Run the simulation (previously instantiated as `sim`) by passing the `arg_list` to `sim.run(*arg_list, **keyworded_args)`. `**keyworded_args` contains options such as `run_until`, which tells MEEP when to terminate the simulation.

The following Listing is an excerpt of a functioning core code, roughly according to the above enumeration of steps that are needed to set up and run a flat ground or terrain model simulation.

```
# ***********************************************************************
# *********************** radial_tlm.py ***************************************
# ***********************************************************************
# Import libraries
import sys, os
import datetime
sys.path.append("..")
script_filedir = os.path.dirname(os.path.realpath(__file__))
script_filename = os.path.basename(os.path.realpath(__file__))
output_dir = script_filedir+"/sim-output"

import meep as mp
import pandas as pd
import numpy as np
import math as m
import matplotlib.pyplot as plt
from Simulations.utility_functions import *

field_component_meep =  { # componint conversion to MEEP API syntax
                        "Ex": [mp.Er, mp.output_efield_r],
                        "Ey": [mp.Ep, mp.output_efield_p],
                        "Ez": [mp.Ez, mp.output_efield_z],
                        "Hx": [mp.Hr, mp.output_hfield_r],
                        "Hy": [mp.Hp, mp.output_hfield_p],
                        "Hz": [mp.Hz, mp.output_hfield_z] }
# ***********************************************************************
# *************** CONFIGURATIONS AND PARAMETERS ******************
# ***********************************************************************
a = 1000 # MEEP unit size in m
eps0 = 8.854e-12 # epsilon_0 dielectric constant
c0 = 299792458
i0 = 1 # Ampere
time_unit = a/c0 * 1e6 # propagation of one MEEP unit distance in microseconds
cyl_scaling_factor = 4/m.pi
e_conversion_factor = i0/(a*c0*eps0)

output_filename = "field_output" # name of csv file containing the field output

time = 140 # total simulation time (in MEEP units)
resolution = 250 # resolution of the unit cell (1km/resolution is one delta x)
courant_factor = 0.67 # is determined from chapter "Numerical Stability" 2.1.5

size_x = 120 # size of simulation cell in x direction (km)
pml_thickness = 1.5 # thickness of PML at the edge of the cell
cell_x = size_x + pml_thickness # extends cell by pml thickness at outer end
above_ground = 0.005 # measure above ground (km) to avoid averaged permittivity effects

# Lightning channel specific:
stroke_length = 10 # length of return stroke channel in km
v_rs = 1.5e8 # return stroke speed in m/s
# nr of sources per MEEP unit. 1 per grid point, if set equal to resolution:
sources_per_unit = resolution
# this is the distance between two sources for given sources per unit:
source_extent = 1/sources_per_unit
# nr of sources making up the whole channel
total_nr_src = int(stroke_length * sources_per_unit)
src_x = 0 # distance of vertical channel from the origin. 0 distributes sources at the z-(symmetry) axis
# field sampled at:
```

```
dist_from_src = np.r_[105:114:0.001]

# **************** TERRAIN MODEL INITIALITATION, IF NOT FLAT *****************
with_terrain = True # terrain model will be used
prism_model = True # terrain model as prism model, else terrain_fct (more severe stair stepping!)
epsilon_averaging = True # epsilon averaging of material leads to smoother corner edges
if with_terrain: # else just flat
terrain_filename = "./RadialTLM/gaisberg_neudorf_avg.csv"
# TerrainModel is a self-written class related to a special terrain model (analytical or real terrain)
terrain_model = TerrainModel(kind="file", cell_x=cell_x, resolution=resolution,
    filename=terrain_filename )

# field output function:
field_to_csv_at_every = int(time/100) # output of fields every time/100 time step

# ***************** MATERIAL PARAMETERS AND TERRAIN MODEL ********************
# Here, the material parameters are initialized
epsr = 10 # Remark: -1e20 (-infinity) corresponds to a mp.perfect_electric_conductor
E_conductivity = 0.001 # Conductivity in S/m
# convert to dimension-free MEEP D_conductivity; specifies lossy ground medium:
D_conductivity_mp = a/(eps0*epsr*c0)*E_conductivity
ground_material = mp.Medium(epsilon=epsr, D_conductivity=D_conductivity_mp)
# air medium (no conductivity: 1e-20. If set to 0, D_conductivity of ground_material (!) is set 0 too)
air_material = mp.Medium(epsilon=1, D_conductivity=1e-20)

# ************ GROUND LAYER OBJECT LOCATION *************************
# the outer layer is reserved for PML, which would otherwise just overwrite the geometry
obj_offset = pml_thickness # makes space for PML at the edge of the simulation cell
# set up cell extent in z-direction, including PMLs:
ground_thickness = 1
# the outer layer is reserved for PML, which would otherwise just overwrite the geometry, therefore offset:
obj_offset = pml_thickness
if not with_terrain: # flat ground
  ground_thickness = 1
  # set up cell extent in z-direction, including PMLs
  cell_z = stroke_length + 2*pml_thickness + ground_thickness
  geom_center_point = -cell_z/2 + obj_offset + ground_thickness/2
  geom_ground_point = -cell_z/2 + obj_offset # for keeping track of reference height levels
  surface_at_origin = geom_ground_point + ground_thickness
else: # for keeping the channel length constant. Otherwise it is cut off earlier in height: Artefacts!
  # add buffer of 0.5 to the maximum terrain elevation:
  ground_thickness = 0.5 + terrain_model.max()
  # set up cell extent in z-direction, including PMLs:
  cell_z = stroke_length + 2*pml_thickness + ground_thickness
  geom_center_point = -cell_z/2 + obj_offset + ground_thickness/2
  geom_ground_point = -cell_z/2 + obj_offset # for keeping track of reference height levels
  # evaluates terrain height at source location and shifts sources up:
  surface_at_origin = geom_ground_point + terrain_model.terrain_fct(src_x)
print("Ground level at origin: "+str(surface_at_origin))

# ********************************************************************
# Channel current source model: PARAMETER SPECIFICATION + MTLM + TOWER MODEL
# ********************************************************************
# 'stroke' is a Python dictionary (dict), initialized as "keyword": value: "i" is the current amplitude (kA)
# "tau1","tau2" are the time constants and n further coefficient of the Heidler functions
if stroke_type == "first":
  stroke = [{"i":28, "tau1":1.8, "tau2":95, "n":2}]
elif stroke_type == "subsequent":
  stroke = [{"i":10.7, "tau1":0.25, "tau2":2.5, "n":2},
             {"i":6.5, "tau1":2, "tau2":230, "n":2}]
```

```python
elif stroke_type == "custom":
  stroke = [{"i":15, "tau1":1.1, "tau2":1.7, "n":2},
            {"i":13, "tau1":2, "tau2":70, "n":2}]
elif stroke_type == "real":
  stroke = "./RadialTLM/801_3_1000kHz_I.csv"
else:
  print("Stroke type does not exist:", stroke_type); raise ValueError
# 'mtlm' is a dictionary holding modified transmission line model parameters:
# "type" can be "exp" (exponential) or "lin" (linear); "lambda" is the decay length of the channel currents
# in meters: in the exponential case [exp(-height/lambda)] and in the linear case [1-height/lambda]
mtlm = {"type": "exp", "lambda": 2000}
# write the source current-related data into a dict:
src_configs = {"src_x": src_x,
               # the surface height (z) of the src x location, which can be offset by src_offset:
               "surface_z": surface_at_origin,
               "v_rs": v_rs, # return stroke velocity
               "sources_per_unit": sources_per_unit,
               "stroke_length": stroke_length,
               "source_component":mp.Ez,
               "channel_base_current": stroke} # (Heidler) current parameters

with_tower = True
if with_tower: """ if True, initialize a tower tower dict with "H": tower size
   in km, "rho_t" and "rho_b" are top and ground reflection coeffs and n the
   number of channel injections """
      tower = {"H": 0.100, "rho_t":-0.53, "rho_g":0.7, "n":10}

""" RS_Model is a class, which takes as arguments all previously defined and
   described parameters. discontinuity = True would mean, that the return stroke
     front of the injected waves into the channel cannot overtake previous slower
     wave fronts """
rs_model = RS_Model(src_configs=src_configs,
                    tower=tower if with_tower else None, discontinuity=False,
                    t_ord_of_magnitude=1e-6, a=a, resolution=resolution)

# ***** MATERIAL FUNCTION (if with_terrain and prism model is not activated) *****
""" a geometry needs to be passed to MEEP as a list of blocks: [mp.Block(...)].
   This is done as follows for terrain (with prism model or terrain function)
   and flat ground. mp.Block takes its center and size as parameters and the
   material a mp.Medium either purely (like ground_material which was
   instanciated above at "MATERIAL PARAMETERS AND TERRAIN MODEL") or as a
   material function that returns mp.Medium. The latter is defined as
   material_fct(p) in the corresponding if-else condition below"""
if with_terrain:
  if prism_model: # build terrain with prism primitives
    geometry = terrain_model.prism_func(geom_ground_point)
  else:  # build terrain with material function
    def material_fct(p): # compares Vector3 type MEEP location p (like p.z) to height threshold
      h_threshold = geom_ground_point + terrain_model.terrain_fct(p.x)
      if (p.z<=h_threshold): # compare height p.z to h_threshold
        return ground_material # return ground material if below threshold
      else:
        return air_material # return air if above threshold
    geometry = [mp.Block(center=mp.Vector3(0,0,geom_center_point),
                         size=mp.Vector3(mp.inf,0,ground_thickness),
                         material=material_fct)]
else: # flat ground; mp.inf is infinity (1e20 is equivalent!)
  geometry = [mp.Block(center=mp.Vector3(0,0,geom_center_point),
                       size=mp.Vector3(mp.inf,0,ground_thickness),
                       material=ground_material)]
```

```
# ********************************************************
# *************** Field output ***************************
# ********************************************************
field_readout_components = ["Ez", "Ex", "Hy"]
readout_at = np.r_[108.5:109:0.002] # a NumPy array of distances in km, sampled every 2m
readout_at_every = 0.05 # MEEP time!
to_csv_at_every = int(time/100) # output of fields every time/100 time step
output_filename = "field_output" # name of csv file containing the field output
# construct field sampling vector:
_sampling_vectors = [mp.Vector3(d_f_s ,0 ,
                    geom_ground_point + above_ground+
                    (terrain_model.terrain_fct(d_f_s) if with_terrain else 0))
                    for d_f_s in readout_at]

fields = {} # this is a dict of dicts, where each key (e.g. "Ez", "Hy") is separated
for f in field_readout_components:
  fields[f] = {"Time": [], "MEEP_Time":[]} # both real Time (mu s) and MEEP_Time export
  for r in _distance_vectors:
    d = round(r[0], 3) # rounds distance to 3 comma values
    height = round(r[2]-geom_ground_point+above_ground, 3) # indicates real height (km)
    key = f+"_("+str(d)+","+str(height)+")" # this is the full field key of the table column
    fields[f][key] = []

def read_out_field(sim):
  """This function outputs the specified fields
       Important note: for sampling, a multiplication with
       MEEP resolution is necessary!!!
       (next to scaling due to MEEP cylindrical symmetry factor 4/pi
       + MEEP E-field conversion)
    """
  cyl_scaling_factor = 4/m.pi # scaling due to radial symmetry
  e_conversion_factor = i0/(a*c0*eps0) # scaling factor of dimension-free results
  mp_t = sim.meep_time()
  t_real = mp_t*time_unit # time with units (micro seconds)
  for field in field_readout_components:
    fields[field]["MEEP_Time"].append(mp_t) # append MEEP Time of current step
    fields[field]["Time"].append(t_real) # append real time of current step
    # for all _readout_vectors, construct keys for "fields"-dict, sample and append value to respective list:
    for r in _sampling_vectors:
      d = round(r[0], 3)
      height = round(r[2]-geom_ground_point, 3)
      key = field+"_("+str(d)+","+str(height)+")"
      # scaling of E-field due to unit-freedom (MEEP), no scaling of H-field:
      e_conv = e_conversion_factor if (field.find("E") != -1) else 1
            # multiplication with cyl_scaling_factor, E-conversion factor AND resolution necessary!!!
      fields[field][key].append(cyl_scaling_factor*resolution*e_conv*np.real(
          # pick the field value of interest (e.g. Ez or Hy):
          sim.get_field_point(field_component_meep[field][0],
          mp.Vector3(r[0],r[1],r[2]+above_ground))))

def write_fields_to_file(sim):
  """ This function writes field values from a dictionary to the a csv file
    using a pandas DataFrame"""
  for field in field_readout_components:
    data = pd.DataFrame.from_dict(fields[field])
    csv_path = output_dir + "/"+filename+"_"+field+".csv"
    data.to_csv(csv_path)


# ********END OF FIELD OUTPUT ************************

# Create arg_list to be passed to the sim.run(*arg_list, **keyworded_args)
arg_list = []
if _sampling_vectors != []:
    arg_list.append(mp.at_every(readout_at_every, read_out_field ))
    arg_list.append(mp.at_every(to_csv_at_every, write_fields_to_file))

""" Output h5 file ever h5_at_every MEEP time units """
output_h5 = ["Ez"]
h5_at_every = 0.5
for field in output_h5:
    arg_list.append(mp.to_appended(field, mp.at_every(h5_at_every,
     field_component_meep[field][1])))

# Create keyworded_args (kwargs), that can be passed to sim.run()
keyworded_args = {"until": time}

# instanciate Simulation class by passing keyworded arguments **sim_config to it
sim = mp.Simulation(**sim_config)
sim.use_output_directory(script_filedir+"/sim-output") # directory of the outputs
sim.run(*arg_list,**keyworded_args)
write_fields_to_file(sim)
```

Listing A.1: Core simulation script: "radial_tlm.py"

The ways to instantiate and run a MEEP simulation flexibly were shown in the last few lines of the above listing. In `sim = mp.Simulation(**sim_config)` and `sim.run(*arg_list, **keyworded_args)`, the arguments are given to the class constructor or `run` method as arbitrarily long lists with corresponding keywords, that can be constructed before starting the simulation. That makes it more flexible with respect to adjustability, when when only small parameter adaptions are needed.

A.2 utility_functions.py

This file contains the classes and their methods for generating the needed models of the RS and the terrain: Modified transmission line model (MTLM) of the RS with and without a tower model (class name: `RS_Model`), and the terrain model with analytical functions or based on a csv-file real terrain profile (class name: `TerrainModel`). Those are instanciated in the core simulation script `radial_tlm.py`.

```
# **********************************************************************
# ********************** utility_functions.py ***************************
# **********************************************************************
import numpy as np
from scipy import interpolate
import csv
import math as m
import matplotlib.pyplot as plt
if __name__ != "__main__":
    import meep as mp

class MeepUnits:
    """
    This class contains the configs and methods to convert the units
    from MEEP to dimensionful units, vice versa
    __init__ is called at the beginning of the instantiation and initializes the
     parameters
    a ... MEEP spatial unit
    resolution ... how often the MEEP unit 'a' is devided
    t_order_of_magnitude ... for correct conversions. Standard is 1e-6 (mu s)
    """
    def __init__(self, a, resolution, t_order_of_magnitude=1e-6):
        """ Initializes parameters of the class (assigns it
        by means of the 'self' variable that holds the class attributes) """
        self.c0 = 299792458 # speed of light (precise definition)
        self.a = a # MEEP spatial unit (e.g. 1000m)
        self.resolution = resolution
        self.t_ord_of_magn = t_order_of_magnitude

    def mp_scale_factor(self, t_order_of_magnitude):
        """
        returns time that it takes a wave to travels a
        unit distance 'a' (e.g. 3,3 mus)
        ord_of_magn must be 1e-3 if ms, 1e-6 if micro sec, etc
        used by methods mpt_to_t and t_to_mpt
        """
        return self.a/self.c0/t_order_of_magnitude #converts to "..s" and returns value

    def mpt_to_t(self, mpt, t_order_of_magnitude): # MEEP time to SI time
        return mpt*self.mp_scale_factor(t_order_of_magnitude)

    def t_to_mpt(self, t, t_order_of_magnitude): # SI time to MEEP time
        return t/self.mp_scale_factor(t_order_of_magnitude)

class RS_Model:
  """The following code extract is the implementation of the RS_model without
    and with a tall structure, incorporating reflections and multiple time
    retarded current injections into the channel. The most important function,
    called from the core script, is sources() which returns a list of sources
    that MEEP can interpret is part of the RS_Model Class.
    * Input params:
    src_configs ... dictionary described in the core script and below in the
    __init__ help
    tower ... is a dictionary with "H": tower size in km, "rho_t" and "rho_b"
              are top and ground reflection coeffs and n the
              number of channel injections. Example:
              tower = {"H": 0.100, "rho_t":-0.53, "rho_g":0.7, "n":10}
    discontinuity ... if True, injected waves cannot pass first wave front:
                      cutoff leads to discontinuity
                      False by default
    """
    def __init__(self, src_configs, tower=None, discontinuity=False,
                 t_ord_of_magnitude=1e-6, a=1000,resolution=250):
        """
        src_configs structure: the parameters are described in the core script
        src_configs = {"src_x": src_x,
                       # the surface height (z) of the src x location,
                        which can be offset by src_offset:
                       "surface_z": surface_at_origin,
                       "v_rs": v_rs, # return stroke velocity
                       "sources_per_unit": sources_per_unit,
                       "stroke_length": stroke_length,
                       "source_component":mp.Ez,
                       # Heidler current params. If string, then path to csv file
                       "channel_base_current": stroke,
                       }
        """
        self.mp_units = MeepUnits(a=1000, resolution=resolution)
```

```python

        """ i_params is a list of dicts (extracted from
            src_configs["channel_base_current"]), which contains the
            Heidler function parameters:
            {"i":1, "tau1":1.8, "tau2":95,"n":2,"t_ord_of_magn":"mus"} """
        self.i_params = src_configs["channel_base_current"]
        if self.i_params is str:  #If string, then path to .csv file of a real measured current!
            self.current_type = "real"
            self.filename = self.i_params
            self.readFile() # read the file
        else:
           self.current_type = "analytical"
           self.filename = None

        self.t_ord_of_magnitude = t_ord_of_magnitude
        self.src_configs = src_configs
        self.mtlm = self.src_configs["mtlm"] # read lambda
        self.mtlm["lambda"] = self.mtlm["lambda"]/self.mp_units.a # scale lambda
        self.tower = tower
        self.discontinuity = discontinuity

    def readFile(self):
        time = []
        amplitude = []
        with open(self.filename, "r") as f:
            data = csv.reader(f, delimiter=',')
            for row in data:
                time.append(float(row[0]))
                amplitude.append(float(row[1]))
        f.close()

        time = np.array(time) # create numpy array
        amplitude = np.array(np.multiply((-1),amplitude)) # invert amplitude
        # Interpolation function of real current:
        self._current_interp = interpolate.interp1d(time,amplitude,kind='cubic')

    def mtlm_factor(self, height):
        """
        This returns the MTLM factor (lin or exp decay of amplitude)
        height must be passed to the function in MEEP units
        """
        if self.mtlm == {}:
            return 1
        if self.mtlm["lambda"]==0:
            return 1
        if self.mtlm["type"] == "exp":
                return m.exp(-height/self.mtlm["lambda"])
        elif self.mtlm["type"] == "lin":
            fact = (self.mtlm["lambda"]-height)/self.mtlm["lambda"]
            if fact >0:
                return fact
            else:
                return 0

    def _current_fct(self, height, t_shift, t_shift_heaviside=0, channel=False,
            wall_src=False):
        """
        As t itself, t_shift comes in mp unit and must be converted to the
        correct base (mus, ms, ns,..)
        t_shift_heaviside is an extra time retardation function to check
        (t<t_shift_heaviside) when the source may activate (e.g. if two step
        functions corcur). Used for tower model discontinuity
        channel = True --> actual channel
        channel = False --> current is injected from the tower and doesn't get
        attenuated!
        wall_src is to activate all sources at the same time (experimental!)
        """
        def wrapper(t):
            """ this wrapper function is defined within the _current_fct
            where all parameters except the time are plugged in.
            It is returned to the mp.CustomSource, which expects an evaluable
            function where the MEEP time of the current step will be plugged in.
            """
            eta = [] # part of the Heidler function
            t_i = [] # this is just a variable substitution used for the function
            current_function = 0
            t_shift_t = self.mp_units.mpt_to_t(t_shift, self.t_ord_of_magnitude)
            t_orig = self.mp_units.mpt_to_t(t, self.t_ord_of_magnitude)
            t = t_orig-t_shift_t
            t_shift_h = self.mp_units.mpt_to_t(t_shift_heaviside,
                                               self.t_ord_of_magnitude)
            """ return values right away before even checking which type of
                function, if time certain criteria are satisfied"""
            if t<0:  return 0
            if t<t_shift_h-t_shift_t:  return 0

            # Set mtlm_factor (can also be checked in advance, since it is applicable to every model!)
            mtlm_factor = 1
            if channel:
                mtlm_factor = self.mtlm_factor(height)
            if wall_src:
                mtlm_factor = 1

            if self.current_type == "analytical":
                for idx, i_params in enumerate(self.i_params):
                    eta.append(m.exp(-i_params["tau1"]/i_params["tau2"]*
                               (i_params["n"]*i_params["tau2"]/
                               i_params["tau1"])**(1/i_params["n"])))
                    t_i.append((t/i_params["tau1"])**i_params["n"])
                    current_function += i_params["i"]/eta[idx]*t_i[idx]/\
                                (t_i[idx]+1) * m.exp(-t/i_params["tau2"])

                return mtlm_factor*current_function

            elif self.current_type=="real":
                return mtlm_factor*self._current_interp(t)
        return wrapper

    def sources(self, wall_src=False):
    """
    This function builds the source array that the MEEP simulation can readily
   use in the Sim(...) instanciation
    and can be called as RS_Model.sources()
    This function does also all the work to include the tower model source
   currents
        wall_src is to activate all sources at the same time (experimental!)
    """
    sc = self.src_configs
    sources = []
    v_rs = sc["v_rs"]

    sources_per_unit = sc["sources_per_unit"]
    source_extent = 1/sources_per_unit
    src_offset=source_extent/2 # to not reach into the ground. Makes no difference though
    stroke_length = sc["stroke_length"]
    source_component = sc["source_component"]

    src_x = sc["src_x"]
    surface_z = sc["surface_z"]+src_offset

    if self.tower != None:
      h_tower = self.tower["H"]
      rho_t = self.tower["rho_t"]
      rho_g = self.tower["rho_g"]
      src_z = surface_z + h_tower
    else:
      h_tower = 0
      src_z = surface_z

    total_nr_src = int(stroke_length*sources_per_unit)
    total_nr_src_tower = int(h_tower*sources_per_unit)

   # *****************************************************************
   # ************ RS ENGINEERING MODEL, no tower *********************
   # *****************************************************************

    for i in range(total_nr_src):
      h = i*source_extent
      start_time = h/(v_rs/self.mp_units.c0)
      if wall_src:
        start_time = 0
      factor = 1
      sources.append(mp.Source(mp.CustomSource(self._current_fct(height=h,
                                  t_shift=start_time,
                                  channel=True,
                                  wall_src=wall_src),
                              start_time = start_time), # before it is 0!
                              amplitude = factor, # amplitude scaling
                              component=source_component,
                              center=mp.Vector3(src_x,0,src_z+h),
                          size=mp.Vector3(0,0,source_extent)))
   # *****************************************************************
   # ************ EXTENDED RS ENGINEERING MODEL - TOWER ********
   # *****************************************************************
   # The corresponding sources will just be added to the already existing ones with the corresponding
   # time retardations (depending on the n-th iteration)
    if self.tower != None:
     # first, the immediate channel reflection stemming from the very beginning of the RS
      if rho_t != 0:
        for i in range(total_nr_src):
          h = i*source_extent
          start_time = h
          t_shift_heaviside = h/(v_rs/self.mp_units.c0)
          factor = -rho_t

          sources.append(mp.Source(mp.CustomSource(self._current_fct(height=h,
                                                  t_shift=start_time,
                                                  t_shift_heaviside=0),
                                              start_time = start_time if not
                                              self.discontinuity else t_shift_heaviside),
                                  amplitude = factor,
                                  component=source_component,
                                  center=mp.Vector3(src_x,0,src_z+h),
                              size=mp.Vector3(0,0,source_extent)))

    # *****************************************************************
    # **** n reflection components in tall object + channel injection *****
    # *****************************************************************
    for n in range(self.tower["n"]): #n is the number of iterations that are executed
      if rho_g != 0 and rho_t != 1:
        for i in range(total_nr_src):
          h = i*source_extent
          start_time = h + 2*h_tower*(n+1)
          t_shift_heaviside = h/(v_rs/self.mp_units.c0)
          factor = (1-rho_t) * (1+rho_t) * rho_g**(n+1) * rho_t**n
          sources.append(mp.Source(mp.CustomSource(self._current_fct(height=h,
                   t_shift=start_time,
                   t_shift_heaviside=0),
                   start_time = start_time if not self.discontinuity \
                   else t_shift_heaviside),
                   amplitude = factor,
                   component=source_component,
                   center=mp.Vector3(src_x,0,src_z+h),
                   size=mp.Vector3(0,0,source_extent)))

        for i in range(total_nr_src_tower):
          h = i*source_extent

          # *************** DOWNWARDS GOING WAVE **********************
          if rho_t != 1:
            start_time = h_tower*(2*n+1)-h
            factor = (1-rho_t) * rho_t**n * rho_g**n
            sources.append(mp.Source(
                              mp.CustomSource(self._current_fct(height=h,
                                          t_shift=start_time),
                                    start_time = start_time),
                                  amplitude = factor,
                                  component=source_component,
                                  center=mp.Vector3(src_x,0,surface_z+h),
                                  size=mp.Vector3(0,0,source_extent)))

          # **************** UPWARDS GOING WAVE ***********************
          if rho_g != 0 and rho_t != 1:
            start_time = h_tower+h + 2*n*h_tower
            factor = (1-rho_t) * rho_t**n * rho_g**(n+1)
            sources.append(mp.Source(
                                mp.CustomSource(self._current_fct(height=h,
                                          t_shift=start_time,
                                      channel=False),
                                    start_time = start_time),
                                  amplitude = factor,
                                  component=source_component,
                                  center=mp.Vector3(src_x,0,surface_z+h),
                                  size=mp.Vector3(0,0,source_extent)))
    return sources
```

```
# ***************************************************************************
# ********************** TerrainModel class *******************************
# ***************************************************************************
 class TerrainModel():
   def __init__(self, kind, cell_x, mp_units, params=None, filename=""):
     """
     kind: 'file' (terrain model from file), 'cos_top' (analytical harmonic,
           with the strikepoint at the local maximum of the function)

     cell_x: extent of the cell in meep units!
     params is a dictionary for example if kind is 'cos_top',
     then params contains keys 'peak_to_peak' and 'periodicity' (in MEEP units)
     """
     self.mp_units = MeepUnits(a=1000, resolution=resolution)
     self.cell_x = cell_x*self.mp_units.a # cell extent in radial direction in meters!
     self.distance = []
     self.height = []
     self.min = float(0)
     self.max = float(0)
     self._terrain_fct = None
     self.filename = filename
     self.params = params

     if kind=="file":
       self.readFile() # read data from file and generate cubic interpolation function
     elif kind == "cos_top":
       self._terrain_fct = self.harmonic(params["peak_to_peak"],
         params["periodicity"],
         angle=0)
     self.minmax() # determine minimum and maximum heights of terrain

   def harmonic(self,peak_to_peak,periodicity,angle):
     """
     peak_to_peak in mp_units a
     periodicity in mp_units a
     angle in radians!
     """
     return (lambda x: peak_to_peak/2*(1+m.cos(2*m.pi*x/periodicity+angle)))

   def readFile(self):
   """ This function reads the terrain file and generates
           the terrain interpolation function '_terrain_fct' """
     distance = []
     height = []
     h_extent = []
     with open(self.filename, "r") as f:
       data = csv.reader(f, delimiter=',')
       for row in data:
         distance.append(float(row[0]))
         height.append(float(row[1]))
     f.close()
‘    #EXTEND DISTANCE TO 130km
     distance.pop()
     distance = np.array(distance)
     if distance[-1]<130000: # distance[-1] is the last element in the list
       dx = distance[-1]-distance[-2] # determine dx, which can vary from file to file
       x_extend = np.r_[distance[-1]+dx:130000:1000j] # linspace with 1000 elements
               # use np.hstack to append the x_extend to distance:
       self.distance = np.hstack([distance, x_extend])/self.mp_units.a
       # Extend height
```

```
        height.pop()
        last_h = height[-1]
        h_extend = np.array([last_h]*len(x_extend))

      self.height = np.hstack([height, h_extend])/self.mp_units.a
      self._terrain_fct = interpolate.interp1d(self.distance,
                                               self.height, kind='cubic')

    def minmax(self):
      """
      Get minimum and maximum heights of terrain
      """
      x = np.array([i for i in range(int(self.cell_x))])/self.mp_units.a
            # map the _terrain_fct return values to all elements in x (see lambda via google)
      y = list(map(lambda x_: self._terrain_fct(x_), x))
      self.min = min(y)
      self.max = max(y)

    def terrain_fct(self, x):
      """
      makes the terrain_fct more obvious for the outside
      """
      return self._terrain_fct(x)

    def prism_func(self, geom_ground_point, ground_params):
      surface_offset = geom_ground_point
      gb_nd = np.genfromtxt(self.filename, delimiter=",")/1000
      epsr = ground_params[0]
      sigma = ground_params[1]
      geometry = []
      extent = 118
      # many trapezoids
      for idx, d in enumerate(gb_nd[:-2,0]):
          vertices = []
          if d < extent:
            vertices.append(mp.Vector3(d,0,surface_offset))
            vertices.append(mp.Vector3(d,0,surface_offset+gb_nd[idx,1]))
            vertices.append(mp.Vector3(gb_nd[idx+1,0], 0,
                                       surface_offset+gb_nd[idx+1,1]))
            vertices.append(mp.Vector3(gb_nd[idx+1,0], 0, surface_offset))
            geometry.append(mp.Prism(vertices, axis=mp.Vector3(0,1,0), height=0,
                                     material=mp.Medium(epsilon=epsr,
                                                        D_conductivity=sigma))
                            )
      return geometry

```

Listing A.2: "utility_functions.py": RS model, terrain model, etc.

A.3 Plotting fields

The fields are output as name_field_component.csv files, that are generated from the `dicts` that contain the field values with the `pandas` library for the sake of simplicity. The keys are written into the first rows. The first column is an index column, automatically added by `pandas`. The second and third column are the `Time` (in μs) and `MEEP_time` (in multiples of Δt_{MP}) respectively. The field results begin at fourth column. An example code to plot the fields, whereby the files and field types, which shall be plotted, can be selected, looks as follows (Listing A.3):

```python
import numpy as np
import sys, os
import matplotlib.pyplot as plt
import re

def plot(fields, field_dist, duration, directories, files):
  """ Plot fields of simulation results """
  plot_nr_fields = 0
  sign = -1
  delimiters = ["(", ")", "_", ","]
  regexPattern = "|".join(map(re.escape, delimiters))

  for directory in directories:
    for file in files:
      if directory == "":
        continue
      sim_output_dir = os.path.realpath(directory)

      for fld_idx, f in enumerate(fields):
        df = pd.read_csv(sim_output_dir+"/"+file)
        for distance_idx, distance in enumerate(field_dist):
          key_index = -1
          keys = list(df.columns.values)
          for idx, k in enumerate(keys[3:]):
            split = re.split(regexPattern, k)
            if (f in split and distance in split):
              key_index = idx
              break
          if key_index == -1:
            continue #continue in distance loop
          time = df['Time'].values - float(f[1])*1/2.998e8*1000*1e6
                             # shifts onset to t=0
          key = keys[3:][key_index]
          field_over_time = df[key].values
          if key.find("E") !=-1:
            field_over_time *= 1000
          fig.gca().plot(time, field_over_time)

  ax = fig.gca()
  ax.set_xlabel("Time [us]")
  ax.set_ylabel("Ez [V/m]")
  starting_point = -5
  ax.set_xlim([starting_point, duration])
  plt.grid()
#  plt.legend()
  plt.show()

if __name__ == "__main__":
  directory = os.path.realpath(__file__)
  fields = ["Ez", "Hy"]
  field_dist = ["100.0", "108.8"]
  files = ["GB-ND-lossy-subsRS_Ez.csv",
          "GB-ND-avg-lossy-subsRS_Ez.csv",
          "GB\_FLAT-lossy-0.001-subsRS_Ez.csv"]
  duration = 70 # in us
  directories = ["."]
  plot(fields, field_dist, duration, directories, files)
```

Listing A.3: Plotting fields of simulation results

Appendix B

Evaluation

B.1 Permittivity ε and field averaging

By means of the stepping functions (such as `output_eps`, `output_E_z`,...), values of interest (ε, E_z, ...) can be sampled at arbitrary instances of time or space. Since there is only a discrete grid holding values, MEEP implements an interpolation method, where any value will be averaged and weighted. Depending on the dimension of the simulation, a linear, bilinear or trilinear interpolation technique is used, where the result at a certain `Vector3(...)` position is calculated by the closest neighboring grid points. The value will be a multi-linear average of the neighboring grid points' values. This serves for the "illusion of continuity" (see [22] and [29]), making the spatial field / material profile appear smooth, as if it was continuous instead of discrete. A remark in the Introduction of the MEEP documentation ([22]) concerning this aspect shall be cited: "However, because it is a simple linear interpolation, while E and D may be discontinuous across dielectric boundaries, it means that the interpolated E and D fields may be less accurate than you might expect right around dielectric interfaces."

Besides that, the relative permittivity at object boundaries does not jump, instead it undergoes a transition depicted in Fig. B.1. Within one spatial resolution (4 m) above ground the value reaches the desired permittivity $\varepsilon_r = 1$. This can also be seen in Fig. B.10, where the ground material (green grid) has no sharp but smooth edges instead.

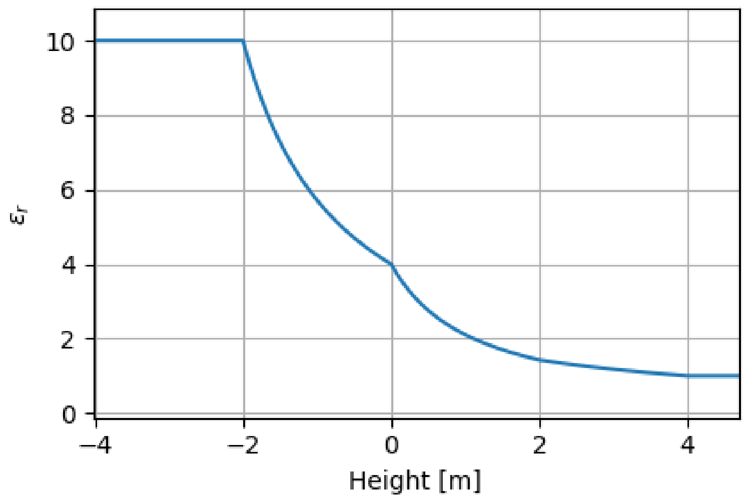
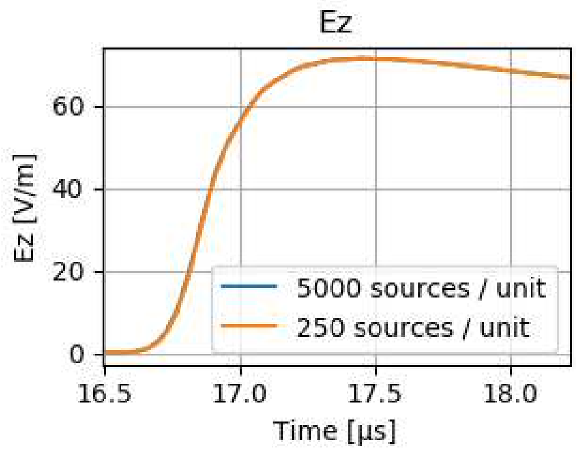


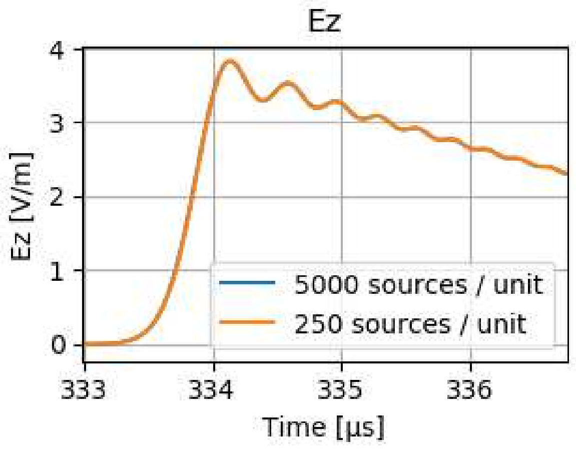
Figure B.1: Transition of $\varepsilon_r = 10$ to $\varepsilon_r = 1$ (air) happens gradually. Height 0 denotes the object surface, positive height is above the object, negative inside the object.

B.2 Supplementary results

B.2.1 Sources



(a) 5 km distance



(b) 100 km distance

Figure B.2: Comparison of sources per MEEP unit.

The E-fields for 5000 sources and 250 sources per a_{MP} are shown in Fig. B.2. There is practically no difference between the two fields, hence it is sufficient to use 250 sources per unit to reduce memory allocation at the simulation initialization. Also, the total simulation time will increase, since every source has to be evaluated once it is active. This is especially important in the case when a tower is implemented by means of distributed current sources. If more sources per unit are necessary, a maximum of 500-1000 sources per unit is recommended.

B.2.2 Terrain model

Fig. B.3 shows the accuracy of the interpolations function, which reproduces the heights perfectly with the cubic (third order) interpolation function of the `scipy.interpolate` class. It can be seen that the cubic interpolating behavior reproduces the profile very precisely. The accuracy of the height values stemming from the interpolation is much better than the resolution of MEEP (4 m). MEEP will introduce a stair stepping of 4 m height, shown in the graph plotted in green.

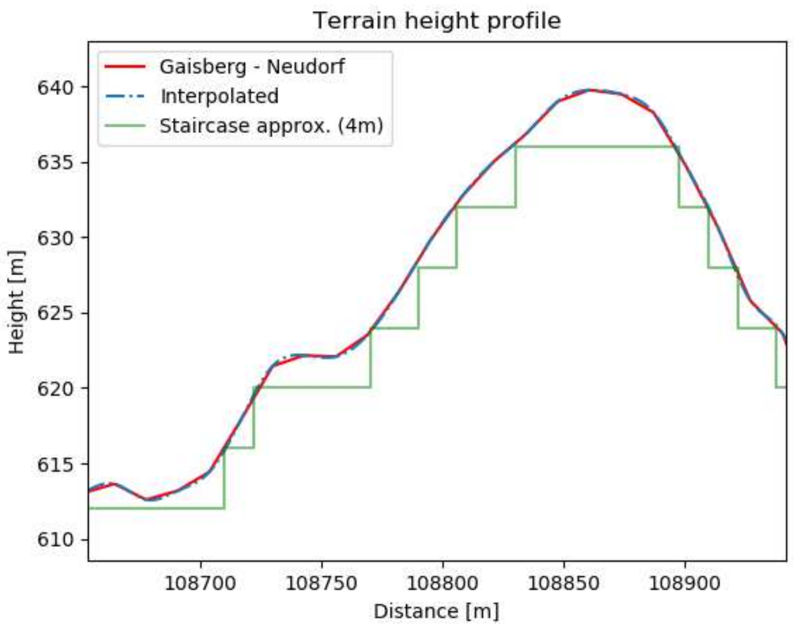
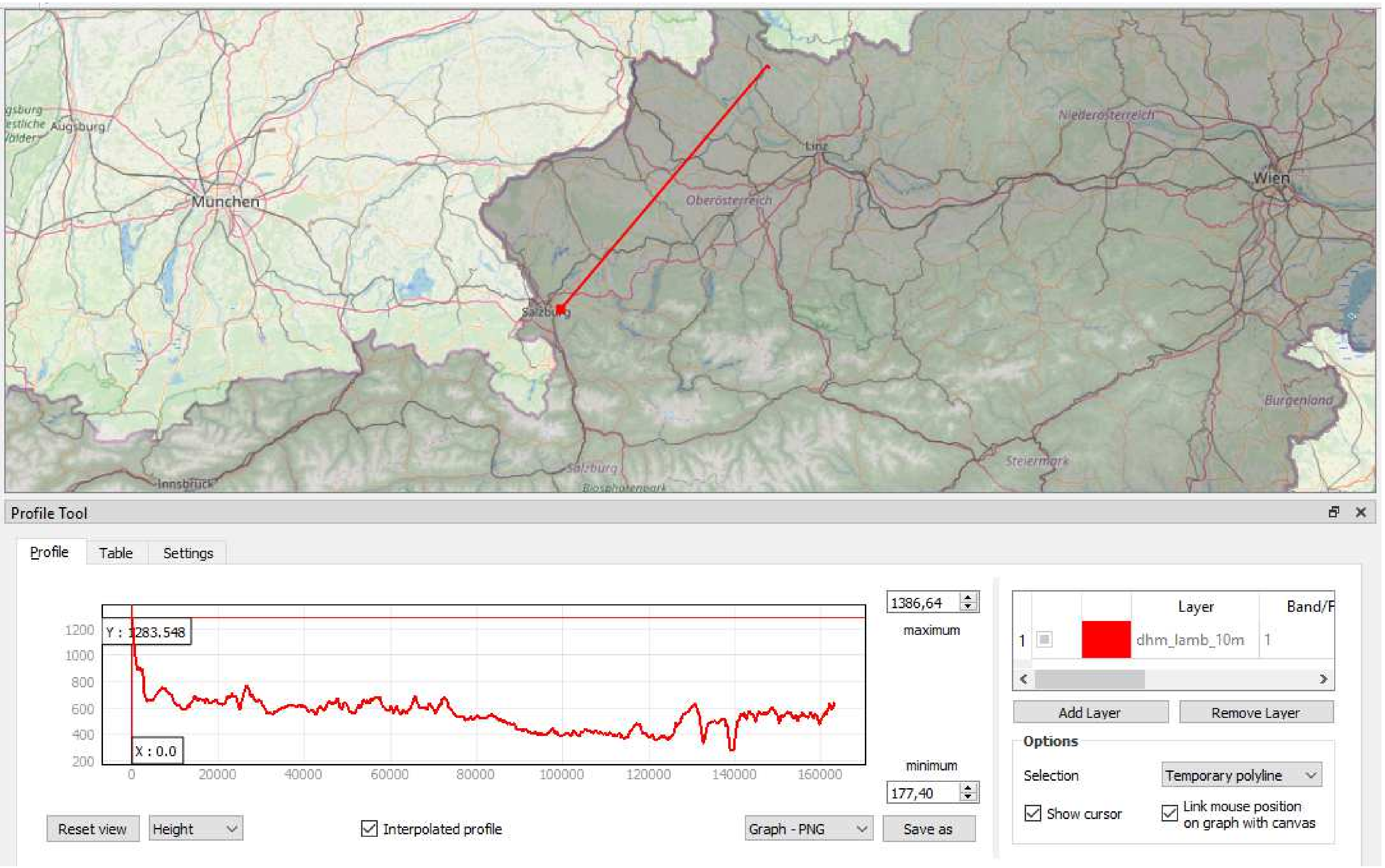


Figure B.3: The interpolated terrain model for some segment of the connection line between Gaisberg and Neudorf is shown next to the original data, as it is fed to the interpolator.

Fig. B.4 shows the extraction of the terrain height profile with QGIS.



maximum 1386,64

minimum 177,40

☑ Interpolated profile

			Layer	Band/F
1	▢	🟥	dhm_lamb_10m	1

Options

Selection: Temporary polyline

☑ Show cursor

☑ Link mouse position on graph with canvas

Figure B.4: Extracting the terrain profile from Gaisberg (Salzburg, Austria) to Neudorf (Upper Austria) along the connecting line of the two locations with QGIS

B.2.3 Terrain model results

B.2.3.1 Local sensitivity without epsilon averaging

The results of the terrain model have to be inspected and interpreted carefully. Because the height profile contains irregularities like hills, along with the stair-stepping phenomenon, the scaling (but not the shape) of the waveform is locally very sensitive. For later comparison, the simulation results of the GB-FLAT model are shown in Fig. B.5, where the field peak values

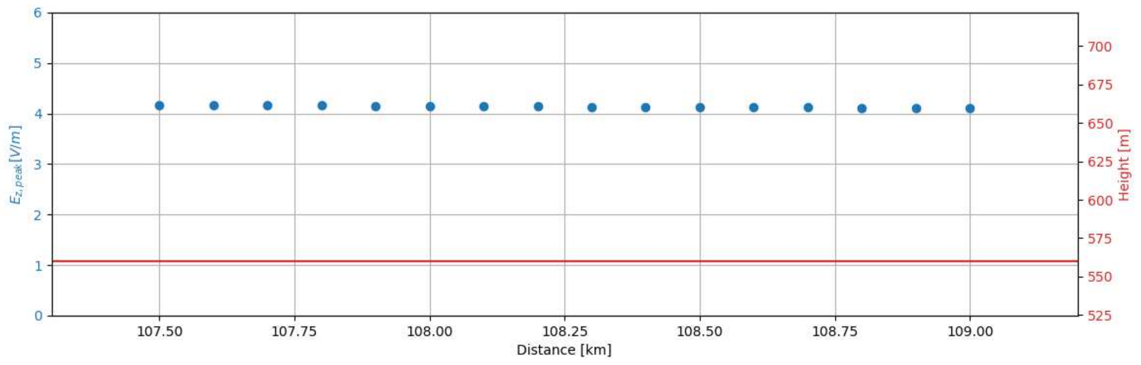
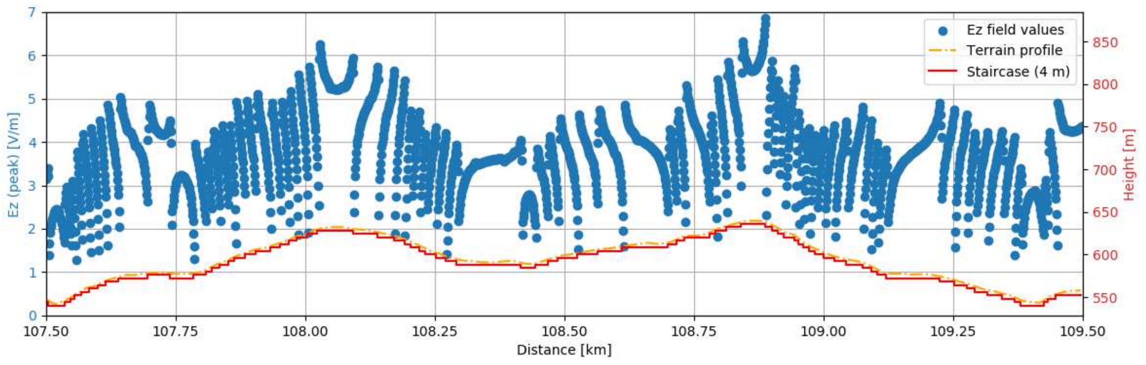
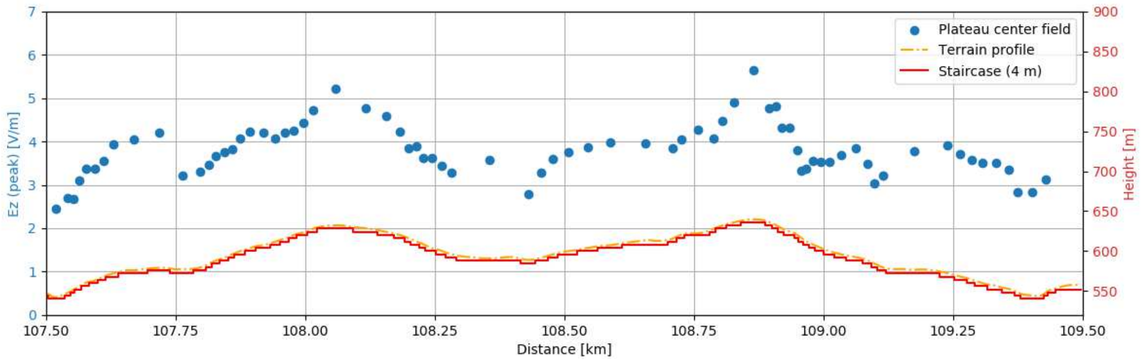


Figure B.5: Peak values of the GB-FLAT terrain model. Values are quasi equal every 100 m.

are sampled over flat ground. The effect of the local terrain curvature can be seen in Fig. B.6. The graphs show the field peaks of the waveforms (subsequent RS, Table 3.1) sampled at every 1 m and 7 m above the (stairstepped) ground. The double y-axis plot shows the peak values of the Ez-fields on the left (blue) and the height profile on the right (red). The field values are plotted together with the smooth terrain profile (orange dashed) and its 4 m stairstepped



(a) Field sampled every 1 m.

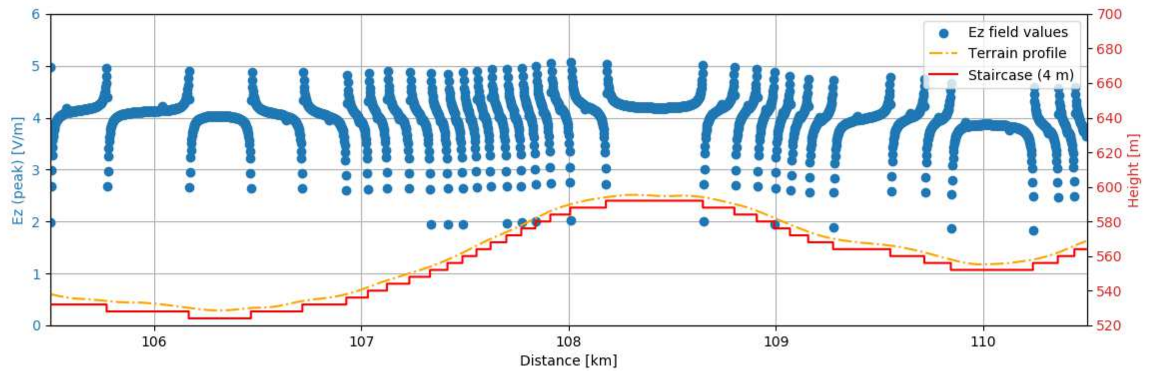


(b) Fields above plateau centers.

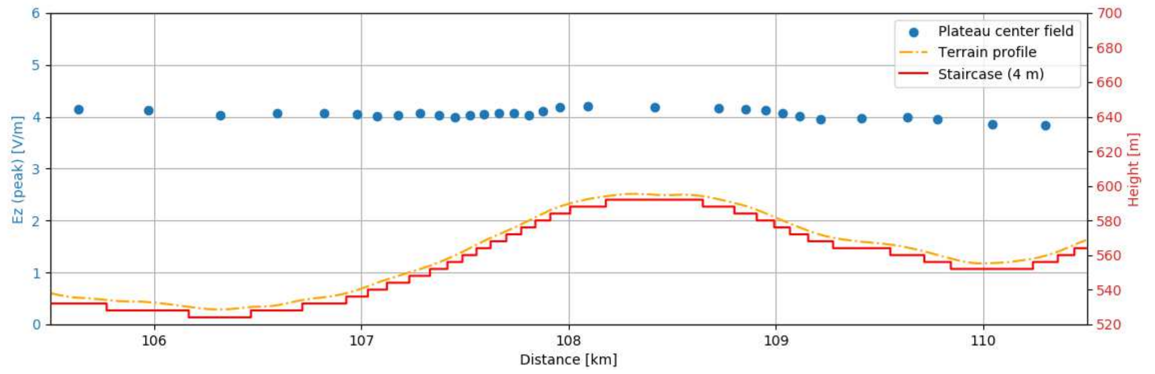
Figure B.6: Field readouts 7 m above stair-stepped ground for the true (un-averaged) terrain profile.

approximation (red solid). Readouts of the vertical E-field in Fig. B.6a spread heavily due the corner effect of the stairs. Readouts at the plateau centers of each stair are shown in B.6b and represent most plausibly the variation of the vertical E-field due to local terrain structure in comparison to Fig. B.5, where the fields just decrease[1] with $1/r$.

The same sensitivity analysis was performed for the averaged (2 km) profile. The results are shown in Fig. B.7a. Again, the fields show a strong corner effect near the discontinuities of the terrain profile. In Fig. B.7b the readouts at plateau centers of the stair steps again return plausible values when compared to Fig. B.5, but with much less deviation compared to the un-averaged terrain in Fig. B.6b.



(a) Field sampled every 2 m.



(b) Fields above plateau centers.

Figure B.7: Field readouts 7 m above stair-stepped ground a) every 2 m b) in the centers of the plateaus.

The differences between the field values sampled from the unaveraged versus the averaged terrain profile are:

- At broader plateaus, the staircase effect happens over a longer distance range, hence field samples at the plateau center are quite reliable. For the unaveraged profile, this effect is more severe, since the corner effect occurs more regularly (shorter plateaus). The stair steps approach the order of MEEP resolution, MEEP interpolation of fields between grid points will also have a bigger influence (although it normally rather improves the results). Yet, when working in the order of the grid resolution, deviations of up to 25 or 30 % in the region of high curvature (many steps along the distance) are not avoidable.
- Elevated objects cause field enhancements when the field is sampled on top of the object. For averaged terrain, smaller objects like hills will smooth out and the local effect of the

[1]Barely visible since the $1/r$ dependence appears as quasi constant values over a short range of far away distances.

terrain curvature will also be smaller. This can be clearly seen when comparing Fig. B.7b to Fig. B.6b, where in the latter graph the field peak development over distance tends to take the shape of the local terrain curvature.

When for comparison the H_φ field is sampled, as shown in Fig. B.8, one can see much more continuous field peak values over the distance. The fields are not influenced by any corner effects, hence no discontinuities are observed. This is the reason, why H fields are normally used in lightning location systems to estimate the current peak values of lightning discharges. In free space, the E-field values can be divided by the free space wave impedance $Z_W = 377\,\Omega$ to get the perpendicular magnetic fields ($H = E\ /\ Z_W$). Doing this with the E-field values of 4.2 V/m from the GB-FLAT results in Fig. B.5, one will get approximately 11 mA/m, which agrees with the range of values in Fig. B.8. Contrary to the E-field enhancement at the top of elevated objects, the H-field shows enhanced values in the valleys and lower values at elevated positions in the range of +/- 5 % of the 11 mA/m.

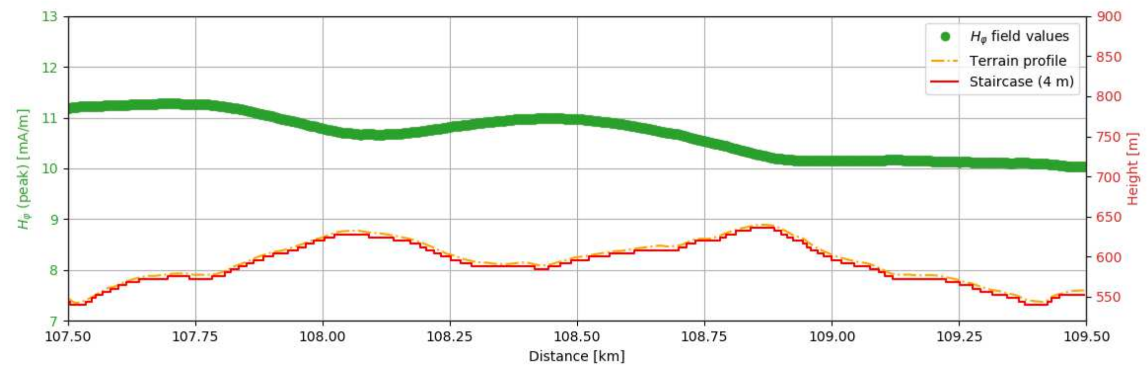


Figure B.8: H_φ peak values every 1 m along unaveraged terrain model.

B.2.3.2 Local sensitivity with epsilon averaging

In comparison to the previous subsection, where the local sensitivity of a material function based model was analyzed, the positive effect of using the prism model with fast epsilon averaging can be seen in Fig. B.9. The discontinuities at the corners of the stair steps are much less severe than in Fig. B.6a. The readouts at the plateau centers (blue) remain equally accurate compared to Fig. B.6b, which shows that the field readouts at the center of the plateau are the most reliable. Additional spatial filtering of local field peaks with a moving average filter of appropriate size can decrease the impact of the stair-stepping effect on the E-field further.

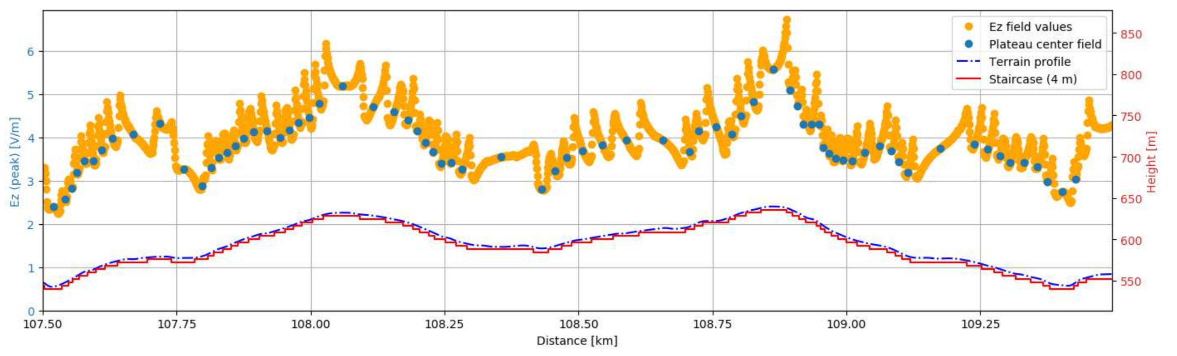


Figure B.9: E_z peak values every 1 m along unaveraged terrain model implemented as a prism model with epsilon averaging.

B.2.3.3 Material function vs. prisms

The way how the ground material is initialized in MEEP by means of a material function (herein called 'Terrain profile') is shown in Fig. B.10. At the beginning, MEEP will step through the grid and compare the location, coordinates `p.x` and `p.z`, to certain criteria (see Listing 2.2) and set the material discontinuously in the Yee grid (red dots, 4 m resolution). This results in stair stepping behaviour which will introduce corner effects, where the E_z-field (blue dots) will be first enhanced and the E-field is no more (strictly) vertical (E_x-component, green dots) directly around the corner. The epsilon grid has a better resolution of 2 m, but appears to just have the major purpose to convert a local D-field to an E-field by $E = D\ /\ \varepsilon$.[2]

Fig. B.11 is a more detailed plot of Fig. B.9. It shows E-fields calculated with the unaveraged terrain model (GB-ND) implemented as prisms with efficient epsilon averaging. The effect can be seen clearly, when comparing the peak fields values with those of the material function results in Fig. B.6a for instance at the distances 108.9 km. For unaveraged epsilon the peak values at the corner vary from 2 V/m - 6 V/m, whereas in the averaged case they only vary from about 5 V/m - 5.5 V/m. Still, care has to be taken when vertical E-fields are sampled from the FDTD grid.

[2] Although at the time of writing the 2 m stepping behavior is not fully clear since it is a very subtle implementation design question, it is apparent that the step at 108.860 km has no corner effect. Just a small bump in the field peaks is visible, which looks like an interpolation effect.

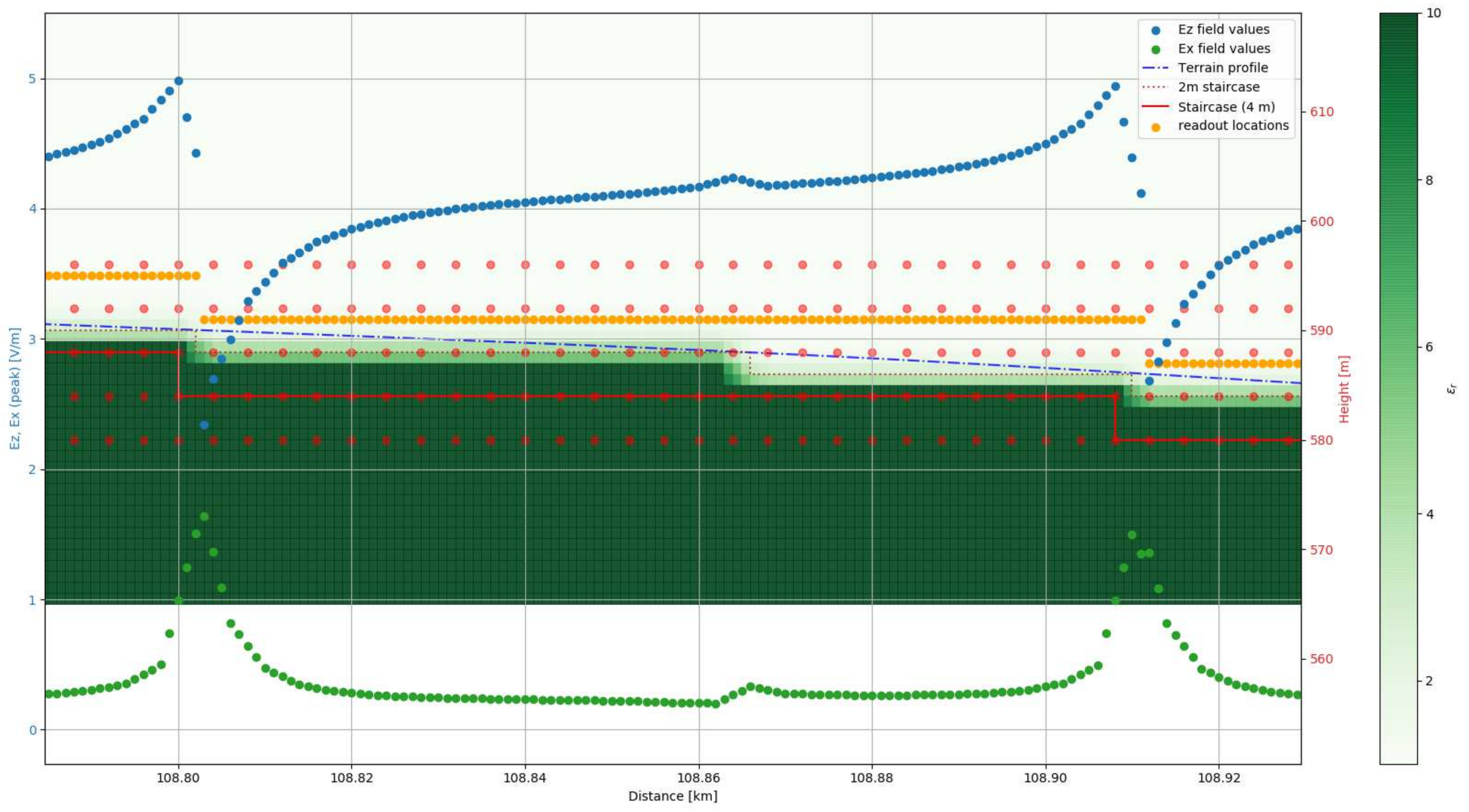


Figure B.10: MEEP stair stepping with corner effects. The grid of red dots is the Yee grid of 4 m resolution.

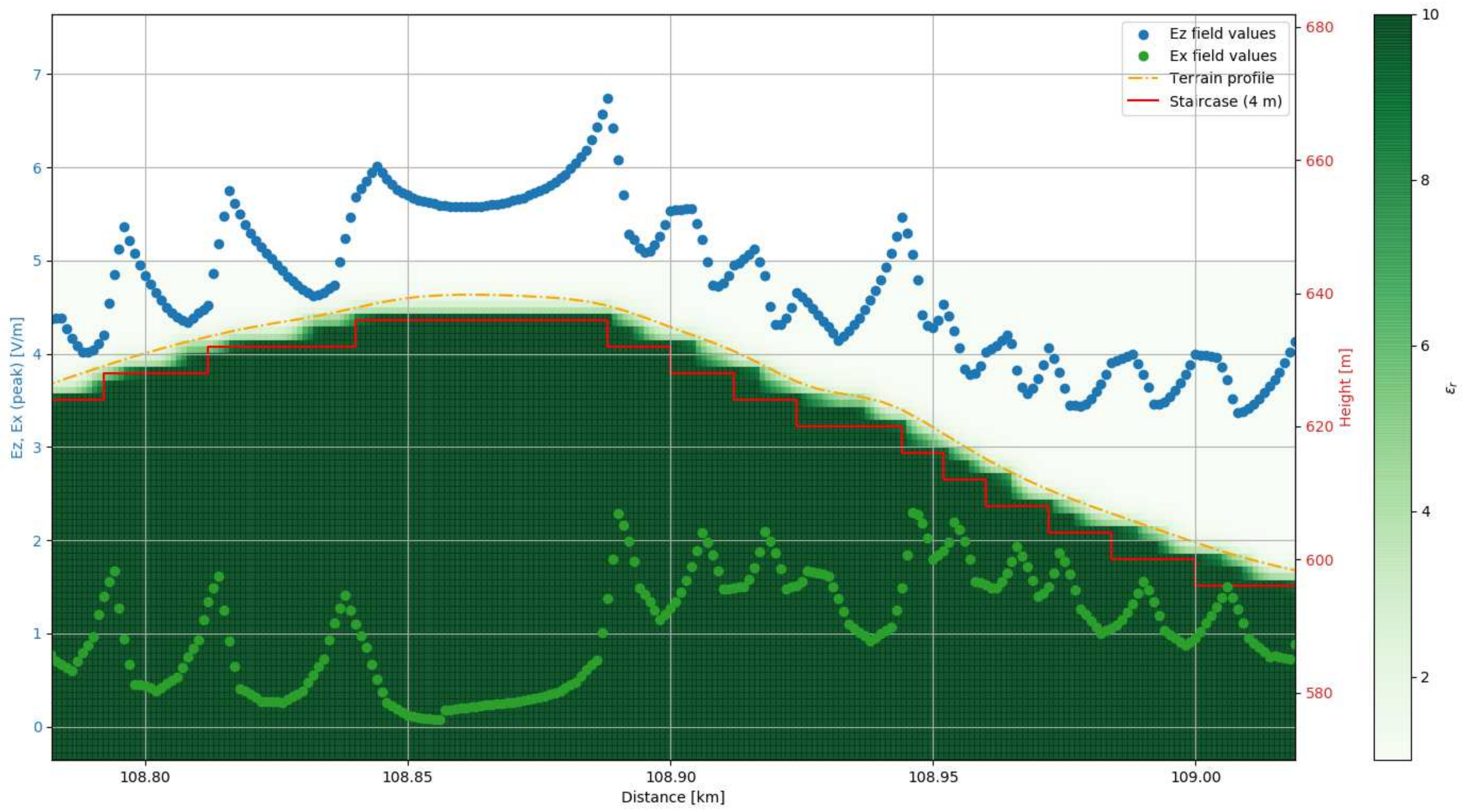
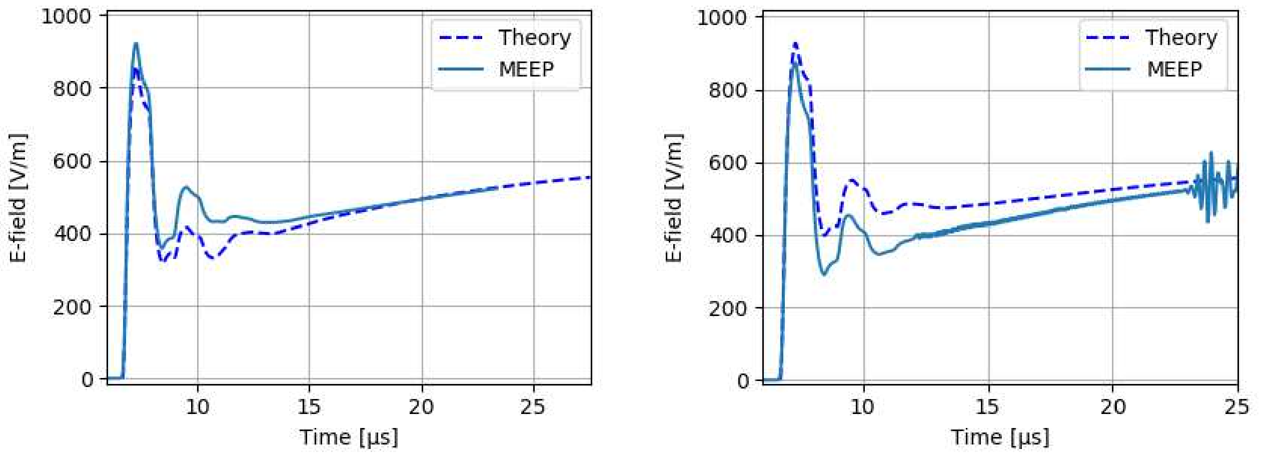


Figure B.11: Original (unaveraged) terrain model implemented as a prism model with epsilon averaging. Stair stepping is smoother with corner effects for E_z and E_x are much smaller than in Fig. B.10. Ex components are higher at the descending part of the hill than at the plateau.

B.2.4 Tower model verification

The tower model, implemented according to the `RS_Model.sources()` method in Appendix A.2, was tested for validity by means of a tool designed according to [43]. A MEEP simulation of a tower of 168 m height with reflection coefficients at the tower top and bottom being $\rho_t = -0.53$ and $\rho_b = 0.7$ (parameters taken from [42]) is performed for a subsequent RS from 3.1. The results compared to the waveforms obtained by the testing tool (called 'Theory' in the plots) and are depicted in B.12. The testing tool only implements the tower model for transmission line (TL) current source RS model only. Hence, the TL model was used instead of MTLE in MEEP.

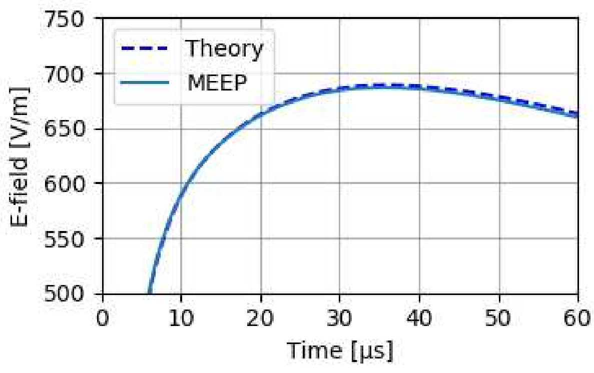


(a) Discontinuity neglected (b) Discontinuity considered (evokes instability)

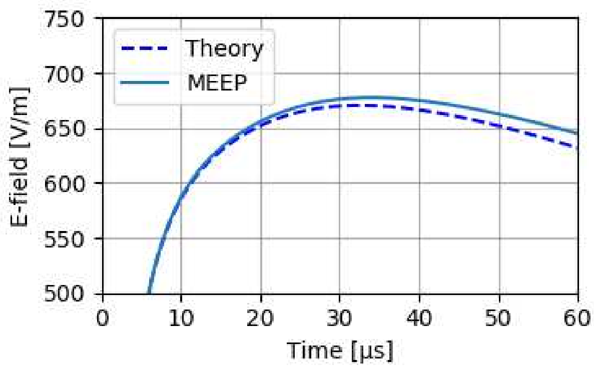
Figure B.12: Tower model in distance of 2 km. Theory (testing tool) versus MEEP.

The discontinuity in Fig. B.12b refers to a cutoff of the reflected wave fronts traveling with c_0 (for details see [43] or [42]) and eventually overtake the primary wave front traveling with $c_0/2$. The cutoff is realized by means of a heaviside function which generates a jump from 0 to a non zero value (discontinuity) at the upwards traveling reflected wave front(s), which results in instability in MEEP, beginning at about 20 μs. The slower the primary RS front is, the faster and more often such a discontinuity will take place. Therefore this model is not feasible for long distance FDTD simulations. In Fig. B.12a that discontinuity is neglected, which means that the reflected waves (c_0) overtake the primary upwards traveling wave front ($c_0/2$). All current sources are therefore smooth and no instabilities occur and the model is feasible for FDTD simulations. Care has to be taken with the reflected waves that will reach the boundary (PML) of the FDTD cell faster, therefore generating radiation artifacts once they enter the PML at the time H_{ch}/c_0, where H_{ch} is the channel length.

Further, although the timing of the changes in orientation of the slopes due to the reflections of the tower is in perfect accordance, a slight deviation of the amplitude and shape can be observed for MEEP versus Theory. Therefore the sanity of the testing tool was checked by means of a very small tower (10 m) with both reflection coefficients being zero. Then the E-field is supposed to be very close to the results without any tower. This was performed for both MEEP and the testing tool and is shown in Fig. B.13. The expected behavior was reproduced better by the MEEP tower implementation than by the testing tool, whose curves decay faster towards 60 μs. Yet, the MEEP simulation reproduces the waveform with respect to the amplitude and shape sufficiently well, meaning that the tower model implementation in MEEP can be assumed to be correct for the scope of the simulations performed in that thesis.



(a) Without tower



(b) Small tower (10m)

Figure B.13: Tower model test for the case of a very small tower with top and bottom reflection coefficients ρ_t and ρ_b both assumed 0. Theory (testing tool) versus MEEP. While a) shows the case without tower

Bibliography

[1] B.F.J. Schonland. “The work of Benjamin Franklin on thunderstorms and the development of the lightning rod”. In: *Journal of the Franklin Institute* 253.5 (May 1952), pp. 375–392. DOI: 10.1016/0016-0032(52)90717-5. URL: https://doi.org/10.1016/0016-0032(52)90717-5.

[2] Martin Uman. *The Lightning Discharge*. Amsterdam, Boston: Academic Press, 1987. ISBN: 978-0-080-95981-8.

[3] Vladimir A. Rakov and Martin A. Uman. *Lightning*. Cambridge University Press, 2003. DOI: 10.1017/cbo9781107340886. URL: https://doi.org/10.1017/cbo9781107340886.

[4] Vladimir A. Rakov. *Fundamentals of Lightning*. Cambridge University Press, 2016. DOI: 10.1017/cbo9781139680370. URL: https://doi.org/10.1017/cbo9781139680370.

[5] Vernon Cooray. *The Lightning Flash*. UK. IET Power and Energy Series 69. Institution of Engineering and Technology, 2014. ISBN: 978-1-849-19691-8.

[6] Vernon Cooray. *Lightning Electromagnetics*. IET Power and Energy Series 62. The Institution of Engineering and Technology, 2012. ISBN: 1849192154, 978-1-849-19215-6.

[7] Vernon Cooray. *An Introduction to Lightning*. Springer Netherlands, 2015. DOI: 10.1007/978-94-017-8938-7. URL: https://doi.org/10.1007/978-94-017-8938-7.

[8] B. Baker, M. B. Baker, E. R. Jayaratne, et al. “The Influence of Diffusional Growth Rates On the Charge Transfer Accompanying Rebounding Collisions Between Ice Crystals and Soft Hailstones”. In: *Quarterly Journal of the Royal Meteorological Society* 113.478 (July 2007), pp. 1193–1215. DOI: 10.1002/qj.49711347807. URL: https://doi.org/10.1002/qj.49711347807.

[9] S. E. Reynolds, M. Brook, and Mary Foulks Gourley. “THUNDERSTORM CHARGE SEPARATION”. In: *Journal of Meteorology* 14.5 (Oct. 1957), pp. 426–436. DOI: 10.1175/1520-0469(1957)014<0426:tcs>2.0.co;2. URL: https://doi.org/10.1175/1520-0469(1957)014%3C0426:tcs%3E2.0.co;2.

[10] Tsutomu Takahashi. “Riming Electrification as a Charge Generation Mechanism in Thunderstorms”. In: *Journal of the Atmospheric Sciences* 35.8 (Aug. 1978), pp. 1536–1548. DOI: 10.1175/1520-0469(1978)035<1536:reaacg>2.0.co;2. URL: https://doi.org/10.1175/1520-0469(1978)035%3C1536:reaacg%3E2.0.co;2.

[11] YouTube. *High-speed video of lightning striking buildings*. 2019. URL: https://www.youtube.com/watch?v=_8cDysr7iPM (visited on 03/19/2019).

[12] V.A. Rakov and M.A. Uman. “Review and evaluation of lightning return stroke models including some aspects of their application”. In: *IEEE Transactions on Electromagnetic Compatibility* 40.4 (1998), pp. 403–426. DOI: 10.1109/15.736202. URL: https://doi.org/10.1109/15.736202.

[13] F. Heidler. “Traveling current source model for LEMP calculation”. In: *Proc. 6th Int. Symp. Electromagnetic Compatibility, Zurich, 1985* (1985).

[14] F. Rachidi, W. Janischewskyj, A.M. Hussein, et al. “Current and electromagnetic field associated with lightning-return strokes to tall towers”. In: *IEEE Transactions on Electromagnetic Compatibility* 43.3 (2001), pp. 356–367. DOI: 10.1109/15.942607. URL: https://doi.org/10.1109/15.942607.

[15] John David Jackson. *Classical electrodynamics.* 3rd ed. Wiley, 1999. ISBN: 9780471309321,047130932X

[16] David J. Griffiths. *Introduction to Electrodynamics* -. 4. Aufl. Cambridge: Cambridge University Press, 2017. ISBN: 978-1-108-42041-9.

[17] Allen Taflove and Susan C. Hagness. *Computational electrodynamics: the finite-difference time-domain method.* 3rd. Norwood: Artech House, 2005.

[18] Lorenz gauge condition. *Lorenz gauge condition — Wikipedia, The Free Encyclopedia.* URL: https://en.wikipedia.org/wiki/Lorenz_gauge_condition (visited on 06/09/2019).

[19] Boundary Element Method. *Boundary Element Method — Wikipedia, The Free Encyclopedia.* URL: https://en.wikipedia.org/wiki/Boundary_element_method (visited on 07/24/2019).

[20] Hongcai Chen, Yang Zhang, Yaping Du, et al. “Lightning Propagation Analysis on Telecommunication Towers Above the Perfect Ground Using Full-Wave Time Domain PEEC Method”. In: *IEEE Transactions on Electromagnetic Compatibility* 61.3 (June 2019), pp. 697–704. DOI: 10.1109/temc.2019.2898036. URL: https://doi.org/10.1109/temc.2019.2898036.

[21] Yoshihiro Baba and Vladimir A. Rakov. *Electromagnetic Computation Methods for Lightning Surge Protection Studies.* John Wiley & Sons Singapore Pte. Ltd, Apr. 2016. DOI: 10.1002/9781118275658. URL: https://doi.org/10.1002/9781118275658.

[22] MEEP. *MEEP Documentation.* 2019. URL: https://meep.readthedocs.io (visited on 03/19/2019).

[23] Allen Taflove, Ardavan Oskooi, and Steven G. Johnson. *Advances in FDTD computational electrodynamics: photonics and nanotechnology.* Artech House, 2013.

[24] John D. Joannopoulos, Steven G. Johnson, and Joshua N. Winn. *Photonic crystals: molding the flow of light.* Princeton University Press, 2008.

[25] Kane Yee. “Numerical solution of initial boundary value problems involving maxwell’s equations in isotropic media”. In: *IEEE Transactions on Antennas and Propagation* 14.3 (May 1966), pp. 302–307. DOI: 10.1109/tap.1966.1138693. URL: https://doi.org/10.1109/tap.1966.1138693.

[26] R. Courant, K. Friedrichs, and H. Lewy. “On the Partial Difference Equations of Mathematical Physics”. In: *IBM Journal of Research and Development* 11.2 (Mar. 1967), pp. 215–234. DOI: 10.1147/rd.112.0215. URL: https://doi.org/10.1147/rd.112.0215.

[27] MIT NanoComp. *Github Repository.* 2019. URL: https://github.com/NanoComp/meep/blob/master/doc/docs/index.md (visited on 03/19/2019).

[28] HDFGroup. *HDFView Download.* 2019. URL: https://www.hdfgroup.org/downloads/hdfview/ (visited on 07/28/2019).

[29] Ardavan F. Oskooi, David Roundy, Mihai Ibanescu, et al. “Meep: A flexible free-software package for electromagnetic simulations by the FDTD method”. In: *Computer Physics Communications* 181.3 (2010), pp. 687–702. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2009.11.008.

[30] Dongshuai Li, Mohammad Azadifar, Farhad Rachidi, et al. “Analysis of lightning electromagnetic field propagation in mountainous terrain and its effects on ToA-based lightning location systems”. In: *Journal of Geophysical Research: Atmospheres* 121.2 (Jan. 2016), pp. 895–911. DOI: 10.1002/2015jd024234. URL: https://doi.org/10.1002/2015jd024234.

[31] SI derived unit. *SI derived unit — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/SI_derived_unit (visited on 05/05/2019).

[32] Simpetus. *Launching MEEP Simulations using 1-Click via AWS MarketPlace*. 2016. URL: https://www.youtube.com/watch?v=40SIIdddVyI (visited on 09/22/2019).

[33] Simpetus. *Launching MEEP Simulations using Manual Launch and EC2 Console*. 2016. URL: https://www.youtube.com/watch?v=T9ZcLGxt8Kg (visited on 09/22/2019).

[34] Simpetus. *Launching MEEP simulations in AWS EC2 using StarCluster*. 2016. URL: https://www.youtube.com/watch?v=-3kee67l-As (visited on 09/22/2019).

[35] K.A. Norton. “The Propagation of Radio Waves over the Surface of the Earth and in the Upper Atmosphere”. In: *Proceedings of the IRE* 25.9 (Sept. 1937), pp. 1203–1236. DOI: 10.1109/jrproc.1937.228544. URL: https://doi.org/10.1109/jrproc.1937.228544.

[36] Gerhard Diendorfer, Hannes Pichler, and Martin Mair. “Some Parameters of Negative Upward-Initiated Lightning to the Gaisberg Tower (2000–2007)”. In: *IEEE Transactions on Electromagnetic Compatibility* 51.3 (Aug. 2009), pp. 443–452. DOI: 10.1109/temc.2009.2021616. URL: https://doi.org/10.1109/temc.2009.2021616.

[37] Wolfgang Schulz, Gerhard Diendorfer, Stéphane Pedeboy, et al. “The European lightning location system EUCLID – Part 1: Performance analysis and validation”. In: *Natural Hazards and Earth System Sciences* 16.2 (Mar. 2016), pp. 595–605. DOI: 10.5194/nhess-16-595-2016. URL: https://doi.org/10.5194/nhess-16-595-2016.

[38] CC-BY-4.0: Land Kärnten - data.ktn.gv.at. *Geländemodell Österreich*. 2019. URL: https://data.gv.at/katalog/dataset/b5de6975-417b-4320-afdb-eb2a9e2a1dbf (visited on 06/06/2019).

[39] QGIS. *A Free and Open Source Geographic Information System*. 2019. URL: https://qgis.org/ (visited on 06/06/2019).

[40] QGIS. *Plugin: Profile tool*. 2019. URL: https://plugins.qgis.org/plugins/profiletool/ (visited on 06/06/2019).

[41] Borys Jurgiel et al. *Profile tool*. 2019. URL: https://plugins.qgis.org/plugins/profiletool (visited on 06/06/2019).

[42] D. Pavanello, F. Rachidi, V.A. Rakov, et al. “Return stroke current profiles and electromagnetic fields associated with lightning strikes to tall towers: Comparison of engineering models”. In: *Journal of Electrostatics* 65.5-6 (May 2007), pp. 316–321. DOI: 10.1016/j.elstat.2006.09.014. URL: https://doi.org/10.1016/j.elstat.2006.09.014.

[43] Andreas F. Dvořak. “SciLab Program for the Calculation of Lightning EM-Fields using different Return Stroke Models”. MA thesis. Technical University of Vienna, June 2014.