

Development of a solver of the Maxwell-Bloch equations with GPGPUs

João Paulo Couto Costa

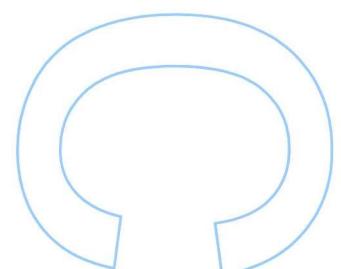
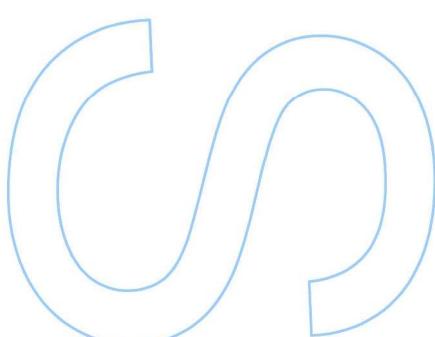
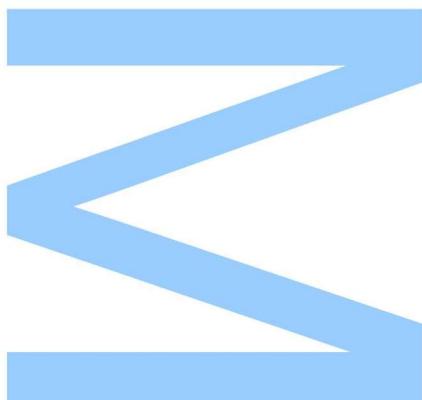
Mestrado Integrado em Engenharia Física

Departamento de Física e Astronomia

2017

Orientador

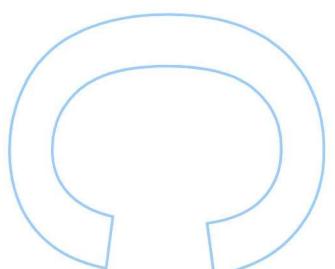
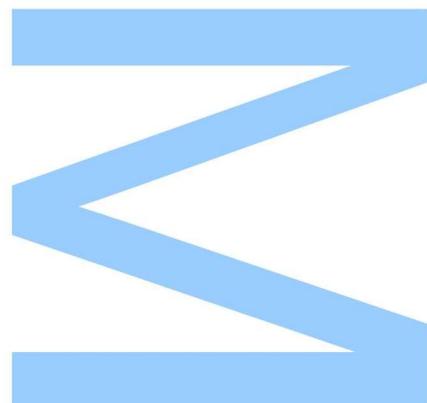
Ariel Ricardo Negrão da Silva Guerreiro, Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto, Investigador Sénior
INESC-TEC





All modifications determined by the
Jury, and only those, were made.
The chairman of the Jury,

Porto, ____ / ____ / ____



“The sea’s only gifts are harsh blows and, occasionally, the chance to feel strong. Now, I don’t know much about the sea, but I do know that that’s the way it is here. And I also know how important it is in life not necessarily to be strong but to feel strong, to measure yourself at least once, to find yourself at least once in the most ancient of human conditions, facing blind, deaf stone alone, with nothing to help you but your own hands and your own head.”

Christopher McCandless (*Into the Wild*)

Acknowledgements

I would like to express my gratitude towards my supervisor, Prof. Dr. Ariel Guerreiro, for guiding the development of this thesis and for pushing me to overcome what I thought to be my limits.

I would like to thank Nuno Azevedo Silva for his help and insights, which were crucial to many decisions during the development of this thesis.

I would also like to thank my friend and colleague Miguel Gomes who worked with me on the design and implementation of the solver and provided constructive criticism to many solutions for the problems we encountered during the development.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto

Integrated Master's in Engineering Physics

Development of a solver of the Maxwell-Bloch equations with GPGPUs

by João Costa

This thesis describes the development of a solver of the three-dimensional Maxwell-Bloch equations implemented using C++ and GPGPU computing technologies. The solver is also a modular software that provides a broad set of tools to study physical phenomena regarding the interaction between electromagnetic fields and atomic gases. Furthermore, this work reviews the physical models that govern the dynamics of the atomic states in the presence of electromagnetic fields using the Maxwell-Bloch equations and accesses some of the available computational methods capable of simulating them. The key concepts of GPGPU computing are presented alongside the design and implementation details of the solver. We compare the performance of the solver using multiple GPU and CPU back-ends and test the numerical stability of the implementation. Finally, we apply our solver to physical scenarios including two and three-level atomic gases and combine our software with a PIC code to demonstrate its modularity.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto

Mestrado Integrado em Engenharia Física

Desenvolvimento de um simulador das equações de Maxwell-Bloch com GPGPUs

por João Costa

Esta tese descreve o desenvolvimento de um *solver* das equações de Maxwell-Bloch tridimensionais utilizando C++ e tecnologia GPGPU. Este *solver* é também um *software* modular que fornece um amplo conjunto de ferramentas para estudar fenómenos físicos relacionados com a interação entre campos eletromagnéticos e gases atómicos. Além disso, este trabalho também analisa os modelos físicos que regem a dinâmica dos estados atómicos na presença de campos eletromagnéticos usando as equações de Maxwell-Bloch e avalia alguns dos métodos computacionais disponíveis capazes de simulá-las. Os conceitos-chave da tecnologia GPGPU são apresentados juntamente com a implementação e o design do *solver*. Comparamos ainda o desempenho do *solver* usando múltiplos GPUs e CPUs e testamos a estabilidade numérica do software. Finalmente, simulamos alguns casos de teste, incluindo gases atómicos a dois e a três níveis e combinamos nosso *solver* com um código PIC para demonstrar sua modularidade.

Contents

Acknowledgements	v
Abstract	vii
Resumo	ix
Contents	xi
List of Figures	xiii
Abbreviations	xv
1 Introduction	1
2 Physical Model	7
2.1 Electromagnetic field	7
2.2 Quantum atomic systems	10
2.2.1 A simple model of a single atom	10
2.2.2 The density operator and the quantum description of a collection of atoms	11
2.3 Electromagnetic fields interacting with atomic gases	12
2.3.1 The Hamiltonian of the atomic system	12
2.3.2 Two-level atomic systems	14
2.3.3 Three dimensional N-level atomic systems	17
2.3.4 Optical phenomena on atomic gases	20
2.3.5 Beyond the Maxwell-Bloch equations	22
2.4 Conclusions	23
3 Numerical model and algorithms	25
3.1 Numerical treatment of the Maxwell equations	26
3.1.1 Solving the Maxwell equations	26
3.1.2 Boundary conditions	29
3.1.3 Electromagnetic field sources	33
3.2 Solving the optical Bloch equations	35
3.3 Coupling the solvers	38
3.4 Conclusions	40

4 Implementation of the solver	41
4.1 GPU computing	41
4.2 Solver design and implementation scheme	44
4.2.1 Scaling of physical equations and natural units	47
4.2.2 The Yee mesh	48
4.2.3 The Field class	49
4.2.4 Tensor Fields	50
4.2.5 Constant Matrix	51
4.2.6 General auxiliary functions	51
4.2.7 Implementation of boundary conditions	53
4.2.8 Implementation of electromagnetic field sources	53
4.2.9 The solver of the Maxwell equations	54
4.2.10 The solver of the Bloch equations	56
4.2.11 Launching a simulation	58
4.3 Performance analysis	59
4.4 Numerical error analysis	61
4.5 Boundary conditions performance	62
4.6 Conclusions	65
5 Physical test cases	67
5.1 Three-level atomic gas in a one-dimensional mesh	68
5.2 Two-level atomic gas in a thin slab	69
5.3 Gaussian pulse propagation in a three-dimensional mesh	70
5.4 Coupling of the solver with a PIC code	72
5.5 Conclusions	74
6 Concluding remarks and future work	77
A Simulation launch example	81
B Thesis outputs	91
Bibliography	95

List of Figures

1.1	Schematic representation of linear and nonlinear optical properties in atomic gases	4
2.1	Scheme of a two-level atomic system	17
2.2	Schematical representation of the dynamics of the transition dipole moment orientation	19
2.3	Scheme of a three-level λ -type atomic system	22
3.1	Yee cell representation	27
3.2	Example of the MABC method layers	33
3.3	Schematic representation of the error per integration step as function of h	35
4.1	GPU vs. CPU FLOPS comparison	43
4.2	GPU computing basic architecture	44
4.3	Simulation domain components	45
4.4	Overview of the simulation execution model	46
4.5	Code classes and libraries hierarchy	47
4.6	General solver scheme	59
4.7	Iteration performance of the solver using different CPUs and GPUs	60
4.8	Speedup comparison on different hardware configurations	61
4.9	Analysis of the numerical stability of the trace of the density matrix during a simulation	62
4.10	MABCs parameter study	63
4.11	Illustration of the angle of incidence used for the boundaries characterization	64
4.12	UPML and MABC attenuation comparison	64
4.13	Iteration performance using MABCs and UPMLs	65
5.1	Preparation of a dark state in a one-dimensional simulation	68
5.2	Transient response of a three-level atomic gas	69
5.3	Evolution of the atomic gas state and polarization upon the interaction with an external field	71
5.4	Propagation of a Gaussian pulse in a cubic mesh filled with a two-level atomic gas	72
5.5	Evolution of the polarization and local density of an atomic gas interacting with a negatively detuned stationary wave	74
5.6	Evolution of the polarization and local density of an atomic gas interacting with a positively detuned stationary wave	75

Abbreviations

GPGPU	General Purpose Graphical Processing Unit
CPU	Central Processing Unit
GPU	Graphical Processing Unit
API	Application Programming Interface
RWA	Rotating Wave Approximation
EIT	Electromagnetically Induced Transparency
PDE	Partial Differential Equation
ODE	Ordinary Differential Equation
FDTD	Finite Differences in the Time Domain
CFL	Courant-Friedrichs-Lowy condition
RK2	Second-Order Runge-Kutta Method
RK4	Fourth-Order Runge-Kutta Method
PML	Perfectly Matched Layer
UPML	Uniaxial Perfectly Matched Layer
MABC	Multiplicative Absorbing Boundary Condition
FWHM	Full Width Half Maximum
PIC	Particle In Cell

Dedicated to my parents Paulo and Rosa and to my brother Rui Pedro, who shared my best moments and supported me through the worst ones, to my girlfriend Inês for always being present, to my friends and family that were always available to lend a hand and to my dogs Yaki and Ralph who accompanied me through my student years.

Chapter 1

Introduction

For many years the field of Optics has studied the propagation of electromagnetic fields through optical media and how the properties of light and matter influence each other. Although at a microscopic level the interaction between light and the atoms or molecules that constitute the optical medium can be quite complex, at a macroscopic level their interplay can be reduced to the refractive index, a parameter that describes the average velocity and absorption of light in the medium. Despite the refractive index constituting a simplified description of the response of the medium, it expresses the microscopic nature of the interaction between light and the medium. For example, when the response of the microscopic components of the medium depends on the state of polarization of light, the medium usually exhibits anisotropic refraction. As a result, this complex interaction between light and matter can frequently be simplified into the macroscopic formulations of the Maxwell equations, which is sufficient for many current day technological applications.

For most materials, the interaction with light can be characterized in terms of the dielectric polarization response of the material, which for low intensities can be assumed to be linear. With the development of high intensity and high coherence optical sources, it became clear that media capable of supporting exotic states of matter, such as graphene and atomic gases, could also exhibit nonlinear responses, which triggered a new field of study called *Nonlinear Optics* or sometimes *Coherent Nonlinear Optics*. In the case of atomic gases, the electromagnetic field can drive the electrons in the atoms out of equilibrium into excited states or exhibit strong forms of quantum coherence. In these cases, it becomes necessary to re-evaluate the notion of refractive index and to delve into physical models that take into consideration the quantum properties of optical media such as the

absorption and emission of radiation by electrons. Atomic gases also combine phenomena that are adequately described by classical models, such as the motion of the atoms or the evolution of the atom density of neutral gases and plasmas, with others that are clearly quantum in nature, such as the atomic quantum state in neutral gases and the macroscopic wave-function for Bose-Einstein condensates. This is an enticing diversity that renders these systems compelling to the study of many physical phenomena.

In these nonlinear regimes, the polarization of the materials typically can be given in terms of the electric field as

$$\mathbf{P}(t) = \epsilon_0(\chi^{(1)}\mathbf{E}(t) + \chi^{(2)}\mathbf{E}^2(t) + \dots + \chi^{(n)}\mathbf{E}^n(t)),$$

where $\mathbf{P}(t)$ and $\mathbf{E}(t)$ represent the instantaneous polarization and electric field respectively, and $\chi^{(n)}$ are the n-th-order susceptibilities of the medium [1]. The linear response of an atom to a resonant light field is described by the first-order susceptibility $\chi^{(1)}$, whose imaginary part is associated with the absorption of the field by the medium, while the real part determines the effective refractive index of the media. The other susceptibilities are associated with the different orders of nonlinearity of the medium. Typically, the last terms are only relevant at very high field intensities, namely when the amplitudes of the external fields are comparable to the interatomic electric fields, which can be achieved using intense and ultrashort laser beams. Even though the values of the different orders of the susceptibility are determined by the instantaneous state of the individual atoms, which in general are continuously evolving, they can be considered as constants if the statistical average over all the local population remains stationary. Since the atoms are driven by the field, the conditions the statistical average state of the atoms to be stationary typically occur when the field is a continuous wave or a pulse with a long duration. For ultrashort pulses, the transient aspects of the atom dynamics result in time dependent susceptibilities that produce exotic effects, well distinct from the typical properties of optical media. The dependence of the susceptibility on the nature of the field can be expressed in another way. Coherent light tuned to specific electronic transitions can prepare the atoms or molecules in exotic states very different from the thermal electronic distribution over the energy levels, for example favouring the population of specific excited levels.

Therefore, to explain the optical properties of atomic or molecular gases it is necessary to describe the quantum state of the atoms or molecules that compose them. The idea

is to decompose a volume of the gas into cells that are small enough to characterize the optical properties of the material as a continuous medium, but large enough to contain a numerous population of atoms or molecules such that the material can be characterized by their average properties. Typically, such cells have sizes below a tenth of the optical wavelengths. Then, the quantum state of the population in each cell can be characterized by a density operator $\hat{\rho}$ and the evolution of the atom-field system is determined by a master equation. Using this description, it is necessary to retain the phase information associated with the evolution of the amplitude of each component of the atomic state, and it is in this sense that one refers to atomic coherence and coherent preparation.

The research of coherent states in optical media is focused on dilute atomic and molecular gases because the mechanisms of decoherence are sufficiently weak to allow the atomic systems to preserve their coherence over several optical cycles (unlike what occurs in solid-state media where decoherence mechanisms quickly kill quantum coherence). Other important advantages of these media are the capacity to be cooled down to ultracold temperatures and their tolerance for high optical intensities without breaking down, enabling the possibility of performing much experimental work over a wide range of conditions.

As mentioned, the optical properties of atomic and molecular gases are essentially defined by the quantum state corresponding to the electron distribution over the different energy levels. Figure 1.1 (a) shows a typical representation of the real and imaginary parts of the linear susceptibility of an atomic medium for two distinct electronic states. The dashed line describes the typical optical properties of a medium when the atoms are in thermal equilibrium, while the bold line describes the same quantities under specific conditions produced by a control external coherent field that creates a destructive interference of the quantum excitation pathways for $\chi^{(1)}$. As a results, it generates a narrow transparency window at the center of the frequency corresponding to a specific electronic transition. This process produces a subsidiary effect shown in figure 1.1 (b), which illustrates a constructive interference of the quantum excitation pathways for $\chi^{(3)}$, resulting in an enhancement of Kerr nonlinearities. This phenomenon was studied by Harris *et al.* [2] and is known as Electromagnetically Induced Transparency (EIT).

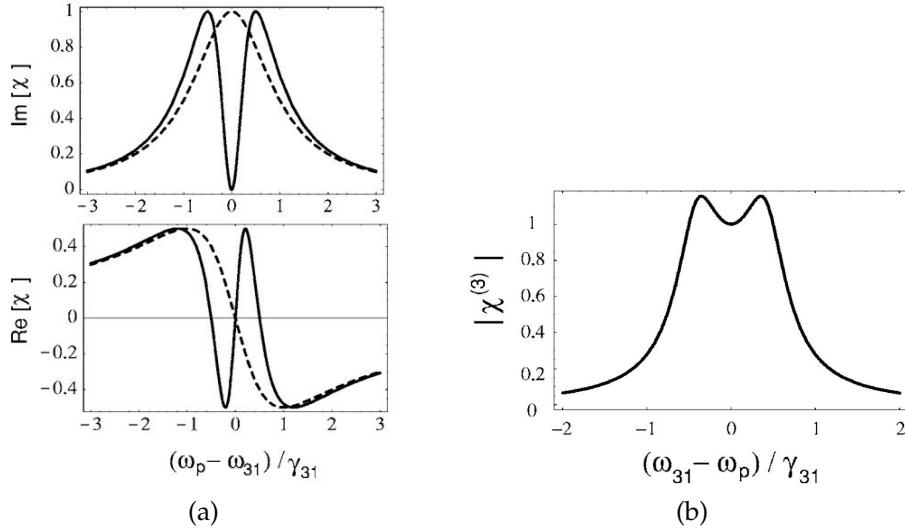


FIGURE 1.1: Schematic representation of linear and nonlinear optical properties in atomic gases discovered by Harris *et. al.* (a) corresponds to the linear susceptibility $\chi^{(1)}$. (b) represents Kerr (third-order) susceptibility $\chi^{(3)}$ (source: [2])

Although the study of nonlinear quantum optical properties is in most cases non-trivial, it permits the development of new technologies and provide insights into the nature of quantum systems. Lately, researchers in this field have been focused on studying both the dynamic and stationary phenomena inherent to light-matter interactions at quantum scales. However, finding exact analytic solutions of quantum-optical problems in both equilibrium and out of equilibrium situations is an arduous task and in a wide range of cases even impossible. Thus, it becomes necessary to introduce simplifications or to adopt numerical methods that can benefit from the enhanced calculation possibilities of modern computers.

Using three-level systems, several authors have identified steady state solutions for the atomic populations using incoherent pumping sources [3] and studied the undisturbed spatial propagation of light solitons in atomic media using both Laguerre-Gaussian beams [4] and standing waves [5] as control fields. These results provide a good understanding of the necessary conditions for stability in such systems, as well as, the main sources of instability that one might encounter when designing experiments. These phenomena also premiss the development of physical systems capable of supporting ultra-slow group velocity as studied by Huang *et al.* [6] and the creation of quantum memories for quantum information processing as proposed by Beausoleil *et al.* [7].

In four-level atomic gases, Sarma *et al.* showed that the electronic population of gas can acquire coherence [8] using trains of femtosecond pulses and analysed the conditions

for maximization and uniformization of such coherence. Also, Hong *et al.* have shown the enhancement of Kerr nonlinearity [9] via creation of atomic coherence in four-level systems and the possibility of controlling both the real and imaginary parts of χ .

A problem that typically arises when studying the interaction of light with atomic gases consists in the difficulty of computing the evolution of the combined systems, especially when the state of the atoms is non-stationary. Calculating the evolution of the electromagnetic field and of the atomic states sometimes exceed the resources of current day computer systems. Hence, the study of the transient phenomena associated with atomic gases is often disregarded in the literature.

Thierry *et al.* [10] simulated ultrashort pulses interacting with optical media using *Finite Differences-Time Domain* (FDTD) scheme with different temporal scales for the multiple phenomena. Alternatively, Xiong *et al.* [11] used methods based on *Fast Fourier Transforms* (FFTs) to compute the evolution of the Maxwell equations, and thus the propagation of the electromagnetic field, while simulating the atomic states dynamics described by Bloch equations using a FDTD approach. Finally, a more general discussion about the simulation of the Maxwell-Bloch equations in N-level media is presented in [12].

The common problem to all the previous schemes is that they require considerable computational time in order to obtain results. On the other hand, these physical systems, from a numerical standpoint, are susceptible of being parallelized using current computer hardware and state of the art computation paradigms. Using today's affordable *General Purpose Graphics Processing Units* (GPGPUs) that have a large amount of cores it is possible to greatly decrease the required computational time to simulate Maxwell-Bloch equations. These technologies are not yet vastly explored in this scientific field but some groundbreaking developments have already shown speedups of approximately twenty [13] and forty [14] when compared to the respective *single-thread* implementations, setting up great expectations for the improvement of light-matter interaction computational solvers.

Another common issue to many simulation schemes is that they are only viable for a specific range of physical phenomena and are most of the times implemented without any well-thought software architecture, which becomes an impediment for future upgrades and the study of other physical effects without having to reimplement a whole new solver.

This work focuses both on identifying the key factors related to simulating semi-classical models of light-atom interaction and on the development of a numerical solver

using GPGPU computing that can simulate the dynamics of electromagnetic fields interacting with N-level atomic gases. The solver should also provide a modular architecture that serves as a foundation for the inclusion of additional solvers that handle other physical phenomena in the future, such as the study of the motion of the atoms in the system.

This master's thesis is organized as follows: The second chapter introduces the physical models of the dynamics of the interaction between electromagnetic fields and quantum atomic systems that constitute the foundation of our solver; chapter three reviews the algorithms and numerical methods to approach the simulation of said interaction; chapter four presents a discussion of GPGPU computing technologies alongside the design and implementation scheme of our solver. Due to the technical nature of this chapter, we include an overview of the important aspects of the software architecture alongside an in-depth analysis of our code; in chapter five we apply our software to the simulation of multiple physical scenarios in order to demonstrate the features of our solver. Finally, chapter six presents the conclusions and discusses future work.

Chapter 2

Physical Model

In order to implement a solver to study the interaction of electromagnetic fields with atomic gases, it is first necessary to review and understand the underlying physical phenomena. However, there are many models that describe this interaction with distinct levels of detail. Therefore, in this chapter, we review some of the available approaches that are more relevant for our solver. Based on these models, we derive a semi-classical model for light-matter interactions that described the field classically, using the Maxwell equations, and the state of atoms from a quantum statistical perspective, using the density operator. This models constitutes an adaptation of the three-dimensional Bloch equations that disregards the Rotating Wave Approximation (RWA) to study the local quantum state of the atomic ensembles.

In the next section, we start with a review of the propagation of electromagnetic fields through optical media, followed by a study of the quantum behaviour of unperturbed atomic systems. Then, we present the relevant models concerning atomic gases and their interaction with electromagnetic fields, starting with the model of two-level atomic systems and then generalising to N-level atoms in three-dimensional systems. Finally, we present the phenomenon of *dark states* in three-level atomic gases for reference in the latter chapters of this thesis and also a brief discussion of the impact of the quantum state of the atoms in their kinematic properties.

2.1 Electromagnetic field

The propagation of electromagnetic fields in a medium can be described with many and distinct levels of detail. At the most fundamental level, one needs to adopt a quantum

electrodynamical approach that describes the field in terms of quanta of light: the photons. However, this model is excessively complex to describe the electromagnetic fields used in experimental atomic-gas systems since the laser sources are usually intense and thus far from the few photon regime where the quantum properties of light are more relevant [15].

Hence and from a classical perspective, the electromagnetic field is described by the *Maxwell equations* [16]

$$\nabla \cdot \mathbf{D} = \rho \quad (2.1)$$

$$\nabla \cdot \mathbf{B} = 0 \quad (2.2)$$

$$\dot{\mathbf{D}} = \nabla \times \mathbf{H} - \mathbf{J} \quad (2.3)$$

$$\dot{\mathbf{B}} = -\nabla \times \mathbf{E}, \quad (2.4)$$

combined with the constitutive relations

$$\mathbf{E} = \frac{1}{\epsilon_0} (\mathbf{D} - \mathbf{P}) \quad (2.5)$$

$$\mathbf{H} = \frac{\mathbf{B}}{\mu_0} - \mathbf{M}, \quad (2.6)$$

where \mathbf{E} , \mathbf{H} , \mathbf{D} , \mathbf{B} , \mathbf{P} and \mathbf{M} are the local electric, magnetic, dielectric displacement, magnetic induction, polarization and magnetization fields and ϵ_0 , μ_0 correspond to the electric permittivity and magnetic permeability in vacuum, respectively. Equations (2.3 - 2.6) are therefore responsible for describing the evolution of the field.

In even simpler models, the electromagnetic field can be described in terms of scalar fields. For example, considering linearly polarized light, the field can be reduced to a scalar field described by the *wave equation* [17]

$$\left(\nabla^2 - \frac{1}{c^2} \frac{\partial^2}{\partial t^2} \right) E(\mathbf{r}, t) = \frac{1}{\epsilon_0 c^2} \frac{\partial^2}{\partial t^2} P(\mathbf{r}, t), \quad (2.7)$$

where c and ϵ_0 are the speed of light and dielectric permittivity of vacuum, respectively. This model can be further simplified when considering, for example, a laser pulse where the field can be described by the corresponding amplitude or envelope profile governed by the *slowly varying envelope approximation* [1]

$$\mathbf{k}_0 \cdot \nabla E_0(\mathbf{r}, t) + \omega_0 \mu_0 \epsilon_0 \frac{\partial E_0(\mathbf{r}, t)}{\partial t} = 0. \quad (2.8)$$

where $E_0(\mathbf{r}, t)$ is the local magnitude of the electric field and \mathbf{k}_0 , ω_0 represent the wave vector and the angular frequency of the laser pulse, respectively. Another common simplification of the wave equation is its time-independent form, known as the *Helmholtz equation* [18], which under the paraxial approximation can be expressed as

$$\nabla_{\perp}^2 u + 2ik \frac{\partial u}{\partial z} = 0, \quad (2.9)$$

where z represents the propagation axis of the electromagnetic field, $u(\mathbf{r})$ is the amplitude of the electric field and ∇_{\perp}^2 corresponds to the transverse Laplacian.

To build an adequate numerical model to study the interaction between light and atomic gases it is necessary to employ a description of the propagation of electromagnetic fields that balances, not only the complexity of the system of equations and the compatibility with the relevant optical phenomena, but also the amount of computer memory necessary to perform the simulation.

As previously stated, the typical optical fields used in experiments with atomic gases are far from the few photon regime and therefore the quantum description of the electromagnetic field is not an appropriate model for this work since the equations are excessively complex and it also has large memory requirements to fully describe the state of light.

The choice relies then between the wave equation model and the classical Maxwell equations. Considering the wave and paraxial equation models, they are very simple to solve numerically and present a low set of requirements in terms of memory usage and therefore have been used already in computer simulations. However, these models are only well-suited when the vector nature of the electromagnetic field can be neglected, typically cases where the mean direction of the propagation vector can be assumed constant. However, the vector and spectral profiles of the electromagnetic field are crucial for our model of light-matter interaction, as presented further in this chapter and therefore, we have adopted the Maxwell equations approach that preserves all the key aspects required by our simulation model and which can be implemented using numerical solvers that have been studied in-depth in the literature.

2.2 Quantum atomic systems

2.2.1 A simple model of a single atom

In order to understand the dynamics of the interaction between atomic gases and electromagnetic fields, it is first necessary to perceive the nature of isolated atomic systems. In modern physics the standard approach to study this problem is to consider an electron that orbits the atomic nucleus and is described by the *Hamiltonian*

$$\hat{H} = \frac{\hat{\mathbf{p}}^2}{2m_e} + V_{eff}(\hat{\mathbf{r}}), \quad (2.10)$$

where $\hat{\mathbf{r}}$ and $\hat{\mathbf{p}}$ correspond to the position and momentum self-adjoint operators, respectively defined as

$$\hat{\mathbf{r}} = \mathbf{r} \quad \text{and} \quad \hat{\mathbf{p}} = -i\hbar\nabla, \quad (2.11)$$

satisfying the canonical commutation relation $[\hat{\mathbf{r}}, \hat{\mathbf{p}}] = i\hbar$ and where m_e corresponds to the mass of the electron. Also, V_{eff} contains not only the electrostatic potential of the positive charges that compose the nucleus but also the repulsive interaction with the remaining electrons, as well as the magnetic interactions between the spins of the electrons and the nucleus that give rise to the fine and hyperfine structure of the electronic energy levels.

The stationary states of the *time-dependent Schrödinger equation* [19]

$$i\hbar \frac{\partial}{\partial t} \Psi(\mathbf{r}, t) = \hat{H}\Psi(\mathbf{r}, t), \quad (2.12)$$

are the eigenstates of the Hamiltonian (2.10). Typically, each state is characterized by a set of distinct quantum numbers which determine many of the properties of the electrons and in particular, their energy.

These quantum numbers combined with the adequate selection rules determine the allowed electronic transitions between the eigenstates under the influence of an electromagnetic field [20]. These features have been extensively studied in atomic physics and spectroscopy [21] and are determinant in the development of the N-level models for the atoms in the following sections. However, not to lose focus in this thesis, we assume that such eigenstates have been indexed by an integer proportional to the energy of the state.

As a results, the state of the electron in the atom can be described as a superposition of states as

$$|\Psi\rangle = \sum_{i=0}^{\infty} C_i |\phi_i\rangle, \quad (2.13)$$

where $|C_j|^2$ corresponds to the probability of the quantum state $|\Psi\rangle$ collapsing in the state $|\phi_j\rangle$ upon observation.

The possibility of describing quantum atomic systems as a superposition of energy levels will be further explored in the following sections.

2.2.2 The density operator and the quantum description of a collection of atoms

In order to study the properties of a quantum physical system, namely one constituted by gas phase atoms and molecules, it is common to perform an analysis over the *Density Matrix* of the system, which corresponds to the matrix representation of the *Density Operator* $\hat{\rho}$ [22]. This operator is related with the impossibility of directly accessing the quantum state of a system and to only being possible to discuss the classical probability of finding that system in a given quantum state. This probability may be interpreted as the frequency of finding a given state in a collection of identical systems (a statistical ensemble). When such classical probabilistic distribution describes a physical system, it is said that the system is in a mixed state, meaning that the system is described by a classical mixture of possible pure quantum states. A pure state can be interpreted as the case when all the atoms of the ensemble occupy the same quantum of the system. The density operator is the quantum-mechanical analogue to a phase-space probability measurement in classical statistical mechanics.

Given this interpretation of the Density Operator and the description of the states of the system in terms of a set of $|\phi_j\rangle$ eigenstates, $\hat{\rho}$ is mathematically defined as

$$\hat{\rho} = \sum_{i=0}^k P_i |\psi_i\rangle \langle \psi_i|, \quad (2.14)$$

where P_i represents the classical probability described above. This description will, therefore, span a $k \times k$ matrix represented in the $|\phi_j\rangle$ base with both diagonal and off-diagonal elements. The diagonal elements portray the information regarding the occupation of the energy levels or populations, while the off-diagonal elements are often referred as the coherence terms.

The most important mathematical properties of a density operator are

$$\text{Tr}(\hat{\rho}) = 1 \quad \text{and} \quad \hat{\rho} = \hat{\rho}^\dagger, \quad (2.15)$$

meaning that the sum of all the classical probabilities is 1 and that the diagonal elements are real numbers between 0 and 1, meaning that they can be interpreted as probabilities.

Knowing the density matrix or density operator it becomes possible to determine the classical mean value of an operator using the result

$$\langle \hat{A} \rangle_{\text{classical}} = \text{Tr}(\hat{\rho}\hat{A}), \quad (2.16)$$

which will be useful in the following section to calculate the classical polarization of an atomic gas.

2.3 Electromagnetic fields interacting with atomic gases

2.3.1 The Hamiltonian of the atomic system

To study the interaction of electromagnetic fields with the particles of an atomic gas the first step consists in determining the Hamiltonian of the system. Here we present an expansion of the concepts of an unperturbed quantum atomic system, introduced in the preceding section, by considering the interaction of the electrons in the atoms with electromagnetic fields. It is important to note that in the scope of this work, we adopt a semi-classical approach where the field is studied classically while the atomic system is described by a quantum model.

We begin by considering the motion of an electron in the presence of an electromagnetic field, for which the classical Hamiltonian is [20]

$$H = \frac{1}{2m_e} [\mathbf{p} - e\mathbf{A}(\mathbf{r}, t)]^2 + V(\mathbf{r}) \quad (2.17)$$

$$= \frac{p^2}{2m_e} - \frac{e}{m_e} \mathbf{p} \cdot \mathbf{A}(\mathbf{r}, t) + V(\mathbf{r}) + \frac{e^2}{2m_e} A^2(\mathbf{r}, t), \quad (2.18)$$

where $\mathbf{A}(\mathbf{r}, t)$ is the *vector potential* associated with the electromagnetic field. Examining the case of electrons that present transitions between levels in the optical frequency range, the typical wavelength of the electromagnetic field is much larger than the size of the atom. Then, it is sufficient to evaluate the vector potential at the centre of mass of the

atom $\mathbf{A}(\mathbf{r}) \approx \mathbf{A}(\mathbf{r}_{cm})$, commonly known as the *dipole approximation* [23], and thus the Hamiltonian of the atomic system can be approximated by

$$H = \frac{p^2}{2m_e} - \frac{e}{m_e} \mathbf{p} \cdot \mathbf{A}(\mathbf{r}_{cm}, t) + V(\mathbf{r}) + \frac{e^2}{2m_e} A^2(\mathbf{r}_{cm}, t). \quad (2.19)$$

Using the previous result and applying the first quantization to the state of the electron while keeping the electromagnetic field classic, it is possible to obtain the Hamiltonian operator

$$\hat{H} = \frac{\hat{p}^2}{2m_e} - \frac{e}{m_e} \hat{\mathbf{p}} \cdot \mathbf{A}(\mathbf{r}_{cm}, t) + V(\hat{\mathbf{r}}) + \frac{e^2}{2m_e} A^2(\mathbf{r}_{cm}, t). \quad (2.20)$$

Now, defining the unitary transformation operator

$$\hat{U} = \exp\left(\frac{ie\hat{\mathbf{r}} \cdot \mathbf{A}(\mathbf{r}_{cm}, t)}{\hbar}\right), \quad (2.21)$$

and using it to perform a change of basis of the Hamiltonian operator, we obtain

$$\hat{H}' = \hat{U}^\dagger \hat{H} \hat{U} \quad (2.22)$$

$$= \frac{1}{2m_e} \hat{U}^\dagger \hat{p}^2 \hat{U} - \frac{e}{m_e} \hat{U}^\dagger \hat{\mathbf{p}} \hat{U} \cdot \mathbf{A}(\mathbf{r}_{cm}, t) + V(\hat{\mathbf{r}}) + \frac{e^2}{2m_e} A^2(\mathbf{r}_{cm}, t), \quad (2.23)$$

Using the Zassenhaus formula [24] is possible to calculate

$$\hat{U}^\dagger \hat{\mathbf{p}} \hat{U} = \hat{\mathbf{p}} + e\mathbf{A}(\mathbf{r}_{cm}, t). \quad (2.24)$$

Thus, using the previous results to multiply the Schrödinger equation (2.12) (in the Dirac picture) by \hat{U} and by replacing $|\Psi\rangle$ with $\hat{U}|\phi\rangle$ one obtains

$$\hat{U}^\dagger \hat{H} \hat{U} |\phi\rangle = i\hbar \hat{U}^\dagger \frac{\partial}{\partial t} (\hat{U} |\phi\rangle) = i\hbar \frac{\partial}{\partial t} |\phi\rangle + i\hbar \hat{U}^\dagger \frac{\partial}{\partial t} \hat{U} |\phi\rangle. \quad (2.25)$$

Considering that

$$\hat{U}^\dagger \frac{\partial}{\partial t} \hat{U} = \frac{ie}{\hbar} \hat{\mathbf{r}} \cdot \frac{\partial}{\partial t} \mathbf{A}(\mathbf{r}_{cm}, t) = -\frac{ie}{\hbar} \hat{\mathbf{r}} \cdot \mathbf{E}(\mathbf{r}_{cm}, t), \quad (2.26)$$

equation (2.25) becomes

$$\left[\frac{\hat{p}^2}{2m_e} + V(\hat{\mathbf{r}}) - e\hat{\mathbf{r}} \cdot \mathbf{E}(\mathbf{r}_{cm}, t) \right] |\phi\rangle = i\hbar \frac{\partial}{\partial t} |\phi\rangle, \quad (2.27)$$

which consists in moving to an interaction picture with the Hamiltonian

$$\hat{H} = \frac{\hat{p}^2}{2m_e} + V(\hat{\mathbf{r}}) - e\hat{\mathbf{r}} \cdot \mathbf{E}(\mathbf{r}_{cm}, t). \quad (2.28)$$

The Hamiltonian \hat{H} is then composed by an unperturbed Hamiltonian \hat{H}_0 , as presented on equation (2.10), plus the interaction term $\hat{H}_I = -e\hat{\mathbf{r}} \cdot \mathbf{E}(\mathbf{r}_{cm}, t)$. While it is true that by using the dipolar approximation one disregards some physical phenomena, which are mostly related to transport effects, such as the Doppler effect, these phenomena are usually small when studying atomic gases and will not be considered for now.

2.3.2 Two-level atomic systems

Although semi-classical two level atomic systems are not the most interesting examples of atomic gases, they provide a good and simple foundation for the understanding of several quantum optical effects present in the interaction between light and matter, which are important when modelling more complex systems, such as three and four level atomic gas systems.

The importance of these systems also becomes evident when one considers that the interaction between light and optical materials is usually described in terms of the variation of the polarization or dipole moment of the atoms or molecules that constitute the medium and are usually described as active optical modes. In the quantum theory, the simplest way to describe this process is to consider that the transition between two electronic states produces a specific dipole moment, known as transition dipole moment.

In a two level atom the electron that interacts with the electromagnetic field can occupy one of two states: A ground state $|1\rangle$ and an excited state $|2\rangle$, with energies $\hbar\omega_1$ and $\hbar\omega_2$, respectively. The states are thereby separated by a transition frequency $\omega_0 = \omega_2 - \omega_1$, as presented of figure (2.1).

The state of an electron in such system can be described as

$$|\psi(r, t)\rangle = C_1(t)|1\rangle + C_2(t)|2\rangle, \quad (2.29)$$

where the evolution of the coefficients C_1 and C_2 is determined by the *Schrödinger* equation

$$i\hbar \frac{d}{dt}C_1 = \langle 1 | \hat{H} | 1 \rangle C_1 + \langle 1 | \hat{H} | 2 \rangle C_2 \quad (2.30)$$

$$i\hbar \frac{d}{dt}C_2 = \langle 2 | \hat{H} | 1 \rangle C_1 + \langle 2 | \hat{H} | 2 \rangle C_2, \quad (2.31)$$

and where \hat{H} is the Hamiltonian operator presented in equation (2.28). The previous result is equivalent to

$$i\hbar \frac{d}{dt} C_1 = \langle 1 | \hat{H}_I | 1 \rangle C_1 + \langle 1 | \hat{H}_I | 2 \rangle C_2 - \omega_0 C_2 \quad (2.32)$$

$$i\hbar \frac{d}{dt} C_2 = \langle 2 | \hat{H}_I | 1 \rangle C_1 + \langle 2 | \hat{H}_I | 2 \rangle C_2 + \omega_0 C_1, \quad (2.33)$$

since the Hamiltonian of the system is composed by an interaction term and the unperturbed Hamiltonian of the atomic system.

To simplify the notation it is common to define

$$\Omega_{ij} \equiv \frac{1}{\hbar} \langle i | \hat{H}_I(t) | j \rangle, \quad (2.34)$$

which corresponds to the *instantaneous Rabi frequency* [25] and it is responsible for driving the electronic transitions between the atomic energy levels. Considering the definition of \hat{H}_I presented of equation (2.28), equation (2.34) becomes

$$\Omega_{ij} = \frac{1}{\hbar} \int_{-\infty}^{+\infty} \phi_i^* (-e\mathbf{r} \cdot \mathbf{E}(\mathbf{r}_{cm}, t)) \phi_j d\mathbf{r}. \quad (2.35)$$

Since we are using the dipolar approximation and therefore only considering the value of the electric field in the centre of mass of the atoms, the previous result is equivalent to

$$\Omega_{ij}(\mathbf{r}_{cm}, t) = \frac{\mu_{ij} \mathbf{u} \cdot \mathbf{E}(\mathbf{r}_{cm}, t)}{\hbar}. \quad (2.36)$$

where μ_{ij} represents the magnitude of the *transition dipole moment* [25] associated with states i and j , corresponding to the *electric dipole moment* associated with the transition between the two states and \mathbf{u} is a unit vector that reflects the atomic transition dipole orientation in space. The transition dipole moment is useful in determining whether the transition between two states is allowed by the principles of quantum mechanics as

$$\mu_{ij} \mathbf{u} = -e \int_{-\infty}^{+\infty} \phi_i^* \mathbf{r} \phi_j d\mathbf{r} \quad (2.37)$$

is only non-zero when the transition respects the *parity selection rule* for electronic dipole transitions. These limitations are usually known as transition rules [20]. The transition dipole moment can also be understood with an analogy to the classical dipole. When one considers two opposing electrical charges in the presence of an electric field, the two charges experience a force in opposite directions leading to a torque on the charge system or dipole. Similarly, depending on the charge density of the electronic orbitals of an atom, in the presence of an electric field, the system will experience a torque that reorganizes the electrons and affects the underlying polarization as presented further in this chapter.

Using the previous results it is possible to re-write equations (2.32) and (2.33) on a matrix form as

$$i\hbar \frac{d}{dt} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} \Omega_{11} & \Omega_{12} \\ \Omega_{21} & \Omega_{22} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} + \begin{bmatrix} 0 & -\omega_0 \\ \omega_0 & 0 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \end{bmatrix}. \quad (2.38)$$

Equation (2.38) describes the evolution of the state of a single two-level atom. For an ensemble of two-level atoms, it is possible to study the system using the *Density Operator* description. Considering the evolution of the state coefficients deduced above and taking into account the notions in section (2.2.2), it is possible to write $\hat{\rho}$ as

$$\rho = \begin{bmatrix} \rho_{11} & \rho_{12} \\ \rho_{21} & \rho_{22} \end{bmatrix}, \quad (2.39)$$

for any mixed state.

The evolution of $\hat{\rho}$ in the Schrödinger picture can be calculated using the *Master Equation* as follows

$$\frac{d}{dt} \hat{\rho} = \frac{i}{\hbar} [\hat{\rho}, \hat{H}], \quad (2.40)$$

where

$$\hat{H} = \begin{bmatrix} \Omega_{11} & \Omega_{12} \\ \Omega_{21} & \Omega_{22} \end{bmatrix} + \begin{bmatrix} 0 & -\omega_0 \\ \omega_0 & 0 \end{bmatrix}. \quad (2.41)$$

However, the density matrix formalism also allows the inclusion of processes that have a stochastic nature and that can be accounted by an effective decay rates between the atomic energy levels. In the effective decay rate are included the decay processes caused by collisions between the atoms and by *spontaneous emission*, which is phenomenologically linked to a finite probability of an electron decaying without any external interference and emitting a photon in the process. Accounting for such phenomena, equation (2.40) becomes

$$\frac{d}{dt} \hat{\rho} = \frac{i}{\hbar} [\hat{\rho}, \hat{H}] + \hat{\Gamma} \rho, \quad (2.42)$$

and considering that the excited state $|2\rangle$ decays to state $|1\rangle$ with an effective decay rate Γ , the dynamics of the entries of the density matrix for a two-level atomic system follows the system of equations

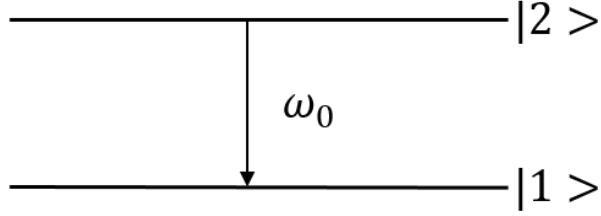


FIGURE 2.1: Graphical representation a two-level atomic system. $|1\rangle$ and $|2\rangle$ correspond to the ground and excited states respectively and ω_0 represents the natural transition frequency between the two energy levels.

$$\rho_{11} = i(\Omega_{12}\rho_{12} + \Omega_{21}\rho_{21}) + \Gamma\rho_{22} \quad (2.43)$$

$$\dot{\rho}_{22} = -i(\Omega_{12}\rho_{12} + \Omega_{21}\rho_{21}) - \Gamma\rho_{22} \quad (2.44)$$

$$\dot{\rho}_{21} = i(\Omega_{12}\rho_{11} + \Omega_{21}\rho_{22} + \omega_0\rho_{12}) - \frac{\Gamma}{2}\rho_{12} \quad (2.45)$$

$$\dot{\rho}_{21} = \rho_{12}^*, \quad (2.46)$$

which is an adaptation of the *optical Bloch Equations* [25, 26]. The main difference between equations (2.43 - 2.46) and the formulations found in the literature is that they describe the dynamics of the density matrix in the Schrödinger picture without using the *Rotating Wave Approximation* (RWA) in the calculation of Ω_{ij} . The RWA introduces simplifications in the dynamics of the atom that we do not want to include in our model. In particular, it filters out frequencies of the spectrum of the electromagnetic field that are far from the electronic resonances while averaging out the dynamics of the atom over the oscillating period of the field.

By coupling the Maxwell and Bloch equations it is possible to retain more optical phenomena intrinsic to the interaction between light and matter that are disregarded when one relies on a simple wave equation description for the electric field. However, this model imposes an increased complexity when one considers the three-dimensional case that will be discussed in the following section.

2.3.3 Three dimensional N-level atomic systems

Having covered the foundations of the interaction between electromagnetic fields and quantum atomic gases, we now present an extension of the model that is compatible with

a three-dimensional system composed of a general number of quantum states and that accounts for the effects of the atoms on the propagation of the electromagnetic field.

Considering the case where the electron can occupy a state corresponding to an overlap of energy levels, as described by equation (2.13), say

$$|\Psi\rangle = \sum_{i=0}^N C_i |\phi_i\rangle, \quad (2.47)$$

it is possible to extend the model obtained in the previous section for the description of a general atomic system composed of N atomic energy levels and deduce the dynamical equations for the overlap factors C_i as

$$\frac{d}{dt} \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix} = -i \begin{bmatrix} \Omega_{11} & \Omega_{12} & \dots & \Omega_{1N} \\ \Omega_{21} & \Omega_{22} & \dots & \Omega_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \Omega_{N1} & \Omega_{N2} & \dots & \Omega_{NN} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix} - \frac{i}{\hbar} \begin{bmatrix} 0 & \omega_{12} & \dots & \omega_{1N} \\ \omega_{21} & 0 & \dots & \omega_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{N1} & \omega_{N2} & \dots & 0 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_N \end{bmatrix} \quad (2.48)$$

Using the previous result and once again applying the properties of the density operator it is now possible to deduce a general formulation of the optical Bloch equations where the entries of the density matrix evolve according to

$$\dot{\rho}_{ij} = \sum_{kl}^N i\Omega_{kl}\rho_{kl} + i\omega_{kl}\rho_{kl} + \Gamma_{kl}\rho_{kl}, \quad (2.49)$$

where for a three-dimensional distribution of atoms, the density matrix describes the quantum properties of the local population particles, and therefore can be understood as a function of time and space, say $\rho_{ij} \equiv \rho_{ij}(\mathbf{r}, t)$. Note that in the scope of this work we consider that all the atoms of the gas are identical and therefore μ_{ij} , ω_{ij} and Γ_{ij} are the same for all atoms.

However, in our three-dimensional model, there is also another major distinction relative to the ones found in the literature [25, 26], besides the disregard of the RWA. Observing equation (2.36) it is possible to conclude that the Rabi frequency in our model is not only time dependent but also presents a spatial dependency since it depends on the local electric field, which permits the study of the interaction of the atomic system with broad-spectrum electromagnetic pulses with complex spatial profiles. This consideration also implies that it is necessary to take into account the instantaneous orientation of the

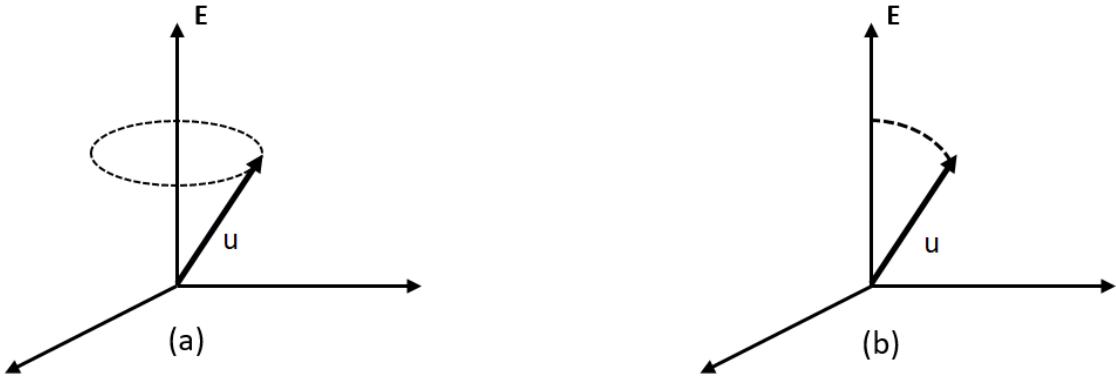


FIGURE 2.2: Graphical representation of the alignment of the unit vector u with the electric field E . (a) represents the precession of u around the direction of the electric field and (b) illustrates the alignment phenomena

atom in space by considering the dipole moment of the atom as vector quantity oriented along the unit vector u . As the local population of atoms is influenced by a local electromagnetic, the orientation of the atomic transition dipole moment tends to align with the electric field through a process that can be modelled using an adaptation of the Landau-Lifshitz equation [27]

$$\dot{\mathbf{u}} = \alpha (\mathbf{u} \times \mathbf{E}) - \beta [\mathbf{u} \times (\mathbf{u} \times \mathbf{E})], \quad (2.50)$$

where $\mathbf{u} \equiv \mathbf{u}(\mathbf{r}, t)$. The first term describes a process of precession around the direction of the electric field and the second term describes a decay corresponding to the alignment of u with the electric field as illustrated on figure (2.2). The parameters α and β that determine respectively the timescales of perception and decay, are obtained experimentally and are typically faster or of the same order of ω_{ij} .

In our model, the optical medium also provides a response to the electromagnetic field in terms of a macroscopic polarization produced by the population of atoms in each point of space. The average polarization is then calculated as

$$\mathbf{P}(x, y, z, t) = \eta \mathbf{u} \text{Tr}(\mu \hat{\rho}) = \eta \sum_{ij}^N \mu_{ij} \mathbf{u} \rho_{ij}, \quad (2.51)$$

where η is the number of atoms per unit volume or density of the gas. This polarization is the same as presented in equation (2.5) and therefore, the atomic gas also affects the electromagnetic field. As a result, the combination of equations (2.5) and (2.51) describe feedback mechanism between light and matter. Although this model does not describe

an explicit and direct interaction between neighbouring atoms, such interaction is present indirectly via the feedback mechanisms between the atoms and the electromagnetic field (which mediates the interaction between different atoms) and via the decay rate of the states Γ_{ij} .

2.3.4 Optical phenomena on atomic gases

In the literature there is a widely discussed phenomena related to the interaction of electromagnetic fields with atomic gases known as *dark states*. Although our model is capable of describing such effects, it is simpler to describe them using the RWA and other approximations as found in the literature [2]. In this section we present a brief review of this effect for reference in later chapters, mainly chapter five.

Consider a three-level atomic system as presented in figure (2.3), known as the λ -type system, composed of three distinct energy levels. If $|1\rangle$, $|2\rangle$ and $|3\rangle$ correspond to the unperturbed eigenstates of the system (ground, excited and metastable, respectively), for any given moment in time it possible to write the state of the system as

$$|\psi(r,t)\rangle = C_1(r,t)|1\rangle + C_2(r,t)|2\rangle + C_3(r,t)|3\rangle. \quad (2.52)$$

For simplicity, it is also assumed that the electric field is composed by two distinct monochromatic waves, denominated as the probe (\mathbf{E}_p) and the coupling (\mathbf{E}_c) fields, and thus the interaction Hamiltonian can be written as $\hat{H}_I = -e\hat{\mathbf{r}} \cdot (\mathbf{E}_p + \mathbf{E}_c)$. If one considers that there is no coupling between states $|1\rangle$ and $|3\rangle$ (i.e μ_{13} and μ_{31} are null) and that the probe field only stimulates transitions between states $|1\rangle$ and $|2\rangle$ while the coupling field is responsible for the transitions between states $|2\rangle$ and $|3\rangle$, it is possible to define

$$\Omega_p(\mathbf{r}, t) = \left[e^{i(\omega_p - \omega_{21})t} + e^{-i(\omega_p + \omega_{21})t} \right] \frac{\Omega_p(\mathbf{r}, 0)}{2} \quad (2.53)$$

$$\Omega_c(\mathbf{r}, t) = \left[e^{i(\omega_c - \omega_{32})t} + e^{-i(\omega_c + \omega_{32})t} \right] \frac{\Omega_c(\mathbf{r}, 0)}{2}, \quad (2.54)$$

as the approximate Rabi frequencies for the probe and coupling fields respectively, using the RWA and with

$$\Omega_p(\mathbf{r}, 0) = \frac{\mu_{12}E_p(\mathbf{r}, 0)}{\hbar} \quad (2.55)$$

$$\Omega_c(\mathbf{r}, 0) = \frac{\mu_{32}E_c(\mathbf{r}, 0)}{\hbar}. \quad (2.56)$$

The RWA also implies that the terms proportional to $e^{-i(\omega_p + \omega_{21})t}$ and $e^{-i(\omega_c + \omega_{32})t}$ may be neglected because they correspond to very fast oscillations when compared to the remaining terms. Therefore, we obtain that

$$\Omega_p(\mathbf{r}, t) \approx \frac{\Omega_p(\mathbf{r}, 0)}{2} e^{i\Delta_1 t} \quad (2.57)$$

$$\Omega_c(\mathbf{r}, t) \approx \frac{\Omega_c(\mathbf{r}, 0)}{2} e^{i\Delta_2 t}. \quad (2.58)$$

where $\Delta_1 = \omega_p - \omega_{21}$ and $\Delta_2 = \omega_c - \omega_{32}$ are the detunings of the fields from the transitions resonances.

Considering the RWA for the probe and coupling field, it is possible to obtain the following matrix representation for interaction Hamiltonian

$$\hat{H}_I = -\frac{\hbar}{2} \begin{bmatrix} 0 & |\Omega_p| & 0 \\ |\Omega_p| & -2\Delta_1 & |\Omega_c| \\ 0 & |\Omega_c| & -2(\Delta_1 - \Delta_2) \end{bmatrix}, \quad (2.59)$$

The eigenstates of the Hamiltonian (2.59), known as the *dressed states*, and their corresponding eigenvalues can be expressed as

$$|a^0\rangle = \cos(\theta)|1\rangle - \sin(\theta)|3\rangle \quad ; \quad E_0 = 0 \quad (2.60)$$

$$|a^+\rangle = \sin(\theta)\sin(\phi)|1\rangle + \cos(\phi)|2\rangle + \cos(\theta)\sin(\phi)|3\rangle \quad ; \quad E_+ = \frac{\Delta + \sqrt{\Delta^2 + \Omega_p^2 + \Omega_c^2}}{2} \quad (2.61)$$

$$|a^-\rangle = \sin(\theta)\cos(\phi)|1\rangle - \sin(\phi)|2\rangle + \cos(\theta)\cos(\phi)|3\rangle \quad ; \quad E_- = \frac{\Delta - \sqrt{\Delta^2 + \Omega_p^2 + \Omega_c^2}}{2} \quad (2.62)$$

where $\Delta = \Delta_1 = \Delta_2$ and where θ and ϕ follow the relations

$$\tan(\theta) = \frac{\Omega_p}{\Omega_c} \quad (2.63)$$

$$\tan(2\phi) = \frac{\sqrt{\Omega_p^2 + \Omega_c^2}}{\Delta_1}. \quad (2.64)$$

Both states $|a^+\rangle$ and $|a^-\rangle$ retain a contribution of all of the atomic eigenstates, but in contrast, the state $|a^0\rangle$ has no component in $|2\rangle$ and is therefore considered a *dark*

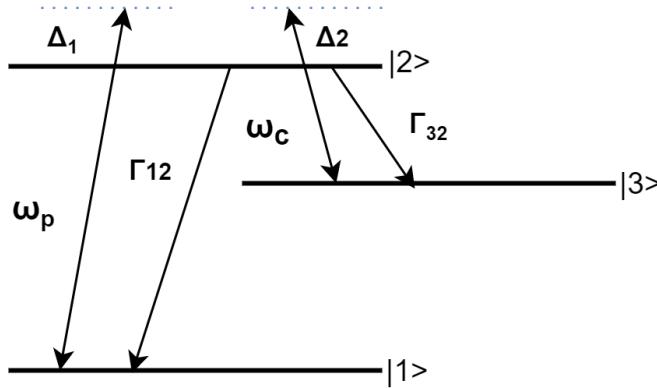


FIGURE 2.3: Graphical representation of a three-level λ -type atomic system. $|1\rangle$, $|2\rangle$ and $|3\rangle$ correspond to the ground, excited and metastable states respectively. ω_p and ω_c represent the probe and coupling field frequencies, which may be detuned from the atomic resonances by Δ_1 and Δ_2 , respectively. Γ_{12} and Γ_{32} are effective decay rates.

state, since if the atom occupies this state there is no possibility of excitation to $|2\rangle$ and subsequent spontaneous emission.

The dark state can be produced via optical pumping, a technique well known in laser spectroscopy and laser-atom manipulation [3]. Another possible method to drive the system into a dark state where spontaneous emission is blocked, is to use a weak probe and a strong coupling field ($\Omega_p \ll \Omega_c$). Using equations (2.63) and (2.64) one sees that under the previous assumption $\sin(\theta) \approx 0$, $\cos(\theta) \approx 1$ and therefore the dark state corresponds to the naked state $|1\rangle$. If a λ -type is prepared in such a way that it occupies a dark state, it will behave as transparent for a propagating electromagnetic field. This phenomena is often denominated Electrochemically Induced Transparency (EIT) for this reason [2].

2.3.5 Beyond the Maxwell-Bloch equations

In our model for the interaction between electromagnetic fields and atomic gases, we have restricted the study to the interaction of the field with the electronic degrees of freedom of the atoms. There are however interesting phenomena regarding the external degrees of freedom associated with the motion of the atoms in the gas that are strongly determined by the interaction of the field with the quantum state of the atoms in the optical medium.

The motion of the atoms can also be affected by the field via the dipole moment in two ways. On one hand, the field exerts a torque on the atoms that rearrange their orientation,

as shown by equation (2.50). This mechanism on the quantum state of the atoms an in particular it determines the dipole moment of the atom.

On the other hand, the field produces a local dipole force on the atoms [8]

$$\mathbf{F}(\mathbf{r}, t) = -\nabla U(\mathbf{r}, t) \quad (2.65)$$

where U is the dipole potential given by

$$U(\mathbf{r}, t) = \sum_{ij}^N \mu_{ij} \mathbf{d}(\mathbf{r}, t) \cdot \mathbf{E}(\mathbf{r}, t) \cdot \rho_{ij}(\mathbf{r}, t) \quad (2.66)$$

This force is responsible for the mechanical movement of the atoms that compose the optical media. It is important to note that this force does not correspond to a Lorentz force since the atoms in this case are neutrally charged but rather is connected with the impact that the quantum nature of the atoms has on their own kinematic properties.

This introduces a new type of phenomena that are quantum-optomechanical in nature, coupling the field, the quantum state of the atoms and the underlying transport processes.

2.4 Conclusions

In this chapter, we investigated the physical models that describe the propagation of electromagnetic fields through optical media, the quantum description of atoms, including the local statistical properties of atomic systems, and their interaction with the fields.

We constructed a semi-classical model that is the foundation of our numerical model and that is based on the Maxwell equations for the propagation of the field and in an adaptation of the optical Bloch equations for the description of the local atomic states. This model is similar to those found in literature but presents some distinctive aspects. First, it extends the conventional Maxwell-Bloch equations to three dimensions and therefore requires the inclusion an empirical model to compute the evolution of the spatial orientation of the transition dipole moment of the atoms and of the local polarization vector. Second, it disregards the RWA and thus it is able to describe more adequately the transient dynamics of the light-matter interaction.

In the following chapter we will revisit this model and identify the available numerical methods and algorithms to study the dynamic processes described in this chapter.

Chapter 3

Numerical model and algorithms

In the previous chapter, we have discussed how our model for the interaction between electromagnetic fields and atomic gases is based on a set of coupled equations, namely the Maxwell equations for the field and the Bloch equations for the atoms. However, these equations are different in nature as the Maxwell equations are *Partial Differential Equations* (PDEs), while the Bloch equations are *Ordinary Differential equations* (ODEs) and therefore must be treated differently by our solver.

In this chapter, we review some of the available schemes for the numerical treatment of the set of coupled dynamic equations and identify the adequate methods to solve the problem at hands. In particular, we discuss the methods to integrate the Maxwell and the Bloch equations that can be combined to address the coupled dynamics of the field and the atoms, while being compatible with GPGPU technologies and providing a solid foundation for future upgrades of the solver.

The first section is reserved for the treatment of the electromagnetic field and reviews the common methods to numerically solve the Maxwell equations, as well as, the problems related to the implementation of boundary conditions and electromagnetic field sources. Next, section two presents a careful analysis of some of the available methods for the simulation of the Bloch equations. The third and final section examines the important aspects regarding the coupling of the previous methods and presents the auxiliary schemes that must be implemented in order to develop an accurate and general numerical solver.

3.1 Numerical treatment of the Maxwell equations

3.1.1 Solving the Maxwell equations

Due to the relevance of the Maxwell equations to many scientific and engineering fields, the methods used for obtaining their numerical solutions are vastly discussed in the literature and it is possible to find a wide diversity of solvers that approach the problem using distinct techniques. In the process of choosing the appropriate numerical scheme for our solver of the Maxwell equations, it must be kept in mind the final goal of this work consists on coupling a set of solvers implemented using GPU computing for the different type of phenomena. Therefore the adopted method must be parallelizable in groups of workers and must be compatible with the physical quantities simulated by the remaining solvers.

There are three prime groups of numerical schemes to compute the solutions of the Maxwell equations, namely the *Frequency Domain* methods, the *Finite Element* methods and *Finite Differences in the Time Domain* (FDTD) methods. Succinctly, the main aspects that characterize these methods are [28]:

- *Frequency Domain* methods - They use *Fourier Transforms* to compute the evolution of the spectrum of the electromagnetic field and are usually a good alternative to study linear phenomena. They are inadequate to handle nonlinearities, such as the effect introduced by the polarization of the optical media;
- *Finite Element* methods - These use simulation meshes composed of irregular finite elements to compute the dynamics of the system and are usually suited for physical contexts with irregular geometries. They difficult the process of depositing and interpolating quantities inside the simulation mesh due to the complex cell geometries;
- *FDTD* methods - They use finite differences to compute both spatial and temporal derivatives and are compatible with phenomena sampled in regular simulation meshes. It is a direct method that handles nonlinearities and impulse responses naturally and is easily adaptable to many problems and can, therefore, be seamlessly coupled with other solvers.

Considering the advantages and disadvantages of the different methods, our choice relies upon the FDTD method mainly due to the compatibility with other solvers and the

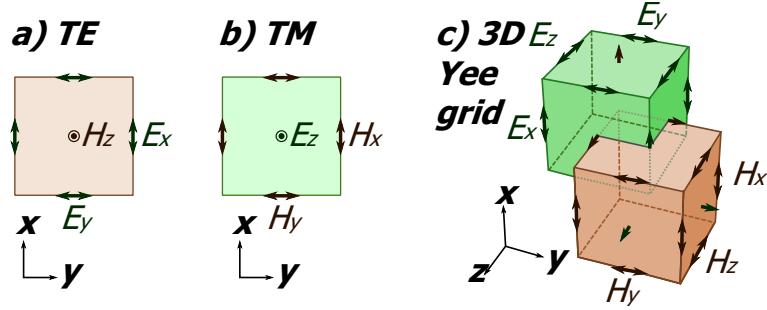


FIGURE 3.1: Design of the three-dimensional Yee cell used in our solver

smaller effort for the parallelization of the algorithm. To implement the FDTD method it is first necessary to define the simulation mesh. The common practice consists in using Yee cells as the unit structure of the mesh, as presented in figure (3.1). In our case, the electric (**E**) and electric displacement (**D**) fields are sampled on the faces of the cells, while the magnetic (**H**) and magnetic induction (**B**) fields are sampled on the edges of the cells. This design, which may be compared to a spatial leapfrog¹ between the components of the field, is a key component to the solution of the Maxwell equations since it permits the optimal calculation of *curls* between different sampling positions. Comparing this sampling technique to the one where all the fields are evaluated on the centre of the cell, this method reduces the spacing in the finite difference by half, while not incurring in more calculations, and therefore the truncation error is maintained. As a result, the precision of calculating the *curl* of the fields using this design is improved approximately by an order of magnitude.

Using the Yee cell it is possible to describe the underlying algorithm of the FDTD method as follows. In our notation the minimal faces and edges and their respective field components belong to the cells indexed by (i, j, k) . Also, $\Delta_x, \Delta_y, \Delta_z$ denote the spacings between the Yee cells in the x, y and z directions, respectively, while Δ_t corresponds to the temporal integration step. Using the sampling scheme previously mentioned, we solve equation (2.3) as

$$D_x|_{i,j,k}^{t_{n+1}} = D_x|_{i,j,k}^{t_n} + \Delta_t \left(\frac{H_z|_{i,j+1,k}^{t_{n+\frac{1}{2}}} - H_z|_{i,j,k}^{t_{n+\frac{1}{2}}}}{\Delta_y} + \frac{H_y|_{i,j,k}^{t_{n+\frac{1}{2}}} - H_y|_{i,j,k+1}^{t_{n+\frac{1}{2}}}}{\Delta_z} - J_x|_{i,j,k}^{t_{n+\frac{1}{2}}} \right) \quad (3.1)$$

¹In the literature the term *leapfrog* is used for quantities that are sampled in different instants in time. Here, we adapted the term *spatial leapfrog* for quantities that sampled on different positions inside the same cell.

$$D_y|_{i,j,k}^{t_{n+1}} = D_y|_{i,j,k}^{t_n} + \Delta_t \left(\frac{H_x|_{i,j,k+1}^{t_{n+\frac{1}{2}}} - H_x|_{i,j,k}^{t_{n+\frac{1}{2}}}}{\Delta_z} + \frac{H_z|_{i,j,k}^{t_{n+\frac{1}{2}}} - H_z|_{i+1,j,k}^{t_{n+\frac{1}{2}}}}{\Delta_x} - J_y|_{i,j,k}^{t_{n+\frac{1}{2}}} \right) \quad (3.2)$$

$$D_z|_{i,j,k}^{t_{n+1}} = D_z|_{i,j,k}^{t_n} + \Delta_t \left(\frac{H_y|_{i+1,j,k}^{t_{n+\frac{1}{2}}} - H_y|_{i,j,k}^{t_{n+\frac{1}{2}}}}{\Delta_x} + \frac{H_x|_{i,j,k}^{t_{n+\frac{1}{2}}} - H_x|_{i,j+1,k}^{t_{n+\frac{1}{2}}}}{\Delta_y} - J_z|_{i,j,k}^{t_{n+\frac{1}{2}}} \right). \quad (3.3)$$

Having obtained the new value of \mathbf{D} , it is then possible to solve equation (2.5) as

$$\mathbf{E}^{t_{n+1}} = \frac{1}{\epsilon_0} (\mathbf{D}^{t_{n+1}} - \mathbf{P}^{t_{n+1}}), \quad (3.4)$$

where \mathbf{P} is an external polarization field that may be calculated by a different solver and in our case corresponds to polarization of the atomic gas. With the updated value of \mathbf{E} it is then possible to calculate the next value of \mathbf{B} as

$$B_x|_{i,j,k}^{t_{n+\frac{3}{2}}} = B_x|_{i,j,k}^{t_{n+\frac{1}{2}}} + \Delta_t \left(\frac{E_z|_{i,j-1,k}^{t_{n+1}} - E_z|_{i,j,k}^{t_{n+1}}}{\Delta_y} + \frac{E_y|_{i,j,k}^{t_{n+1}} - E_y|_{i,j,k-1}^{t_{n+1}}}{\Delta_z} \right) \quad (3.5)$$

$$B_y|_{i,j,k}^{t_{n+\frac{3}{2}}} = B_y|_{i,j,k}^{t_{n+\frac{1}{2}}} + \Delta_t \left(\frac{E_x|_{i,j,k-1}^{t_{n+1}} - E_x|_{i,j,k}^{t_{n+1}}}{\Delta_z} + \frac{E_z|_{i,j,k}^{t_{n+1}} - E_z|_{i-1,j,k}^{t_{n+1}}}{\Delta_x} \right) \quad (3.6)$$

$$B_z|_{i,j,k}^{t_{n+\frac{3}{2}}} = B_z|_{i,j,k}^{t_{n+\frac{1}{2}}} + \Delta_t \left(\frac{E_y|_{i-1,j,k}^{t_{n+1}} - E_y|_{i,j,k}^{t_{n+1}}}{\Delta_x} + \frac{E_x|_{i,j,k}^{t_{n+1}} - E_x|_{i,j-1,k}^{t_{n+1}}}{\Delta_y} \right). \quad (3.7)$$

Finally, \mathbf{H} is updated using

$$\mathbf{H}^{t_{n+\frac{3}{2}}} = \frac{\mathbf{B}^{t_{n+\frac{3}{2}}}}{\mu_0} - \mathbf{M}^{t_{n+\frac{3}{2}}}. \quad (3.8)$$

Examining the previous equations, it is possible to observe that apart from the spatial leapfrog between the electric and magnetic fields imposed by the Yee cell design, the FDTD method also uses the temporal leapfrog technique between these fields as the electric field is sampled at the instants t_n while the magnetic field is sampled at instants $t_{n+\frac{1}{2}}$. The truncation error of this method is $\mathcal{O}(\Delta_x^2, \Delta_y^2, \Delta_z^2, \Delta_t^2)$.

An important note regarding the FDTD method is that it is limited by the *Courant-Friedrichs-Lowy condition* (CFL) for explicit methods, which for the three-dimensional Maxwell equations is given by [28]

$$\frac{v_x \Delta_t}{\Delta_x} + \frac{v_y \Delta_t}{\Delta_y} + \frac{v_z \Delta_t}{\Delta_z} \leq 1 \quad (3.9)$$

where v_x , v_y and v_z correspond to the magnitude of the speed of light in the x , y and z directions, respectively and implies that the choice of the spatial and temporal discretization steps is not arbitrary. Considering an electromagnetic field propagating across a discrete spatial mesh, the CFL condition can be interpreted as follows: if one wants to calculate the amplitude of the field at discrete temporal instants, then the interval between two consecutive instants must be smaller than the time it takes the field to travel between two adjacent mesh points. To use the FDTD method it is also recommended to use at least 20 samples points per the smallest wavelength of the simulation [28], which may limit the simulation mesh dimensions due to memory requirements.

3.1.2 Boundary conditions

A common subject that is usually discussed alongside the methods for solving the Maxwell equations is the implementation of boundary conditions in the simulation mesh. For the scope of this work, we limit the discussion to periodical and absorbing boundary conditions that are usually the most commonly used in the simulation of real systems.

The standard method to implement periodic boundary conditions is rather simple. Consider a simulation mesh with N_x , N_y and N_z cells in the x , y and z axis, respectively and the Yee cell that resides on the edge of the simulation mesh, i.e $i = N_x$, $j = N_y$ and $k = N_z$. Examining equation (3.1), the issue resides on deciding which value of \mathbf{H} one should use to solve the finite difference. However, the problem can be simply solved using

$$D_x|_{N_x, N_y, N_z}^{t_{n+1}} = D_x|_{N_x, N_y, N_z}^{t_n} + \Delta_t \left(\frac{H_z|_{N_x, 1, N_z}^{t_{n+\frac{1}{2}}} - H_z|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}}}{\Delta_y} + \frac{H_y|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}} - H_y|_{N_x, N_y, 1}^{t_{n+\frac{1}{2}}}}{\Delta_z} \right) - \Delta_t J_x|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}} \quad (3.10)$$

$$D_y|_{N_x, N_y, N_z}^{t_{n+1}} = D_y|_{N_x, N_y, N_z}^{t_n} + \Delta_t \left(\frac{H_x|_{N_x, N_y, 1}^{t_{n+\frac{1}{2}}} - H_x|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}}}{\Delta_z} + \frac{H_z|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}} - H_z|_{1, N_y, N_z}^{t_{n+\frac{1}{2}}}}{\Delta_x} \right) - \Delta_t J_y|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}} \quad (3.11)$$

$$D_z|_{N_x, N_y, N_z}^{t_{n+1}} = D_z|_{N_x, N_y, N_z}^{t_n} + \Delta_t \left(\frac{H_y|_{1, N_y, N_z}^{t_{n+\frac{1}{2}}} - H_y|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}}}{\Delta_x} + \frac{H_x|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}} - H_x|_{N_x, 1, N_z}^{t_{n+\frac{1}{2}}}}{\Delta_y} \right) - \Delta_t J_z|_{N_x, N_y, N_z}^{t_{n+\frac{1}{2}}} \quad (3.12)$$

which consists basically in defining the mesh with the topology of a high-dimensional torus. The algorithm used for the remaining fields is analogous.

The methods for introducing absorbing boundary conditions are considerably more complex. When considering absorbing boundary conditions it is desirable that there is an optimal absorption of the incident field but that there also is a minimal reflection. The standard approach in the literature consists on imposing *Perfectly Matched Layers* (PMLs) on the borders of the simulation mesh. The underlying concept behind the PMLs is that they are designed to absorb the electromagnetic field without creating significant numerical reflections inside the simulation space. A profound description of the implementation of these boundary conditions may be found on the work of Taflove *et al.* [28]

For the scope of this work we now limit the discussion to the key aspects of the method to the *Uniaxial Perfectly Matched Layers* (UPMLs), which are one of the many available types of PMLs. This method relies on an adaptation of the finite differences of the FDTD method that provides a lossless medium inside the primary simulation mesh and a set of absorbing layers which reside on the borders of the mesh and that mitigate wave reflections. To impose this medium, the method creates a stretched-coordinate space in the absorbing region of the mesh that is analogous to an adaptation of the Yee cell geometry, capable of trapping the incident and reflected fields in that region. In order to do so, the ∇ operator is defined as

$$\nabla = \frac{\partial}{s_x \partial_x} \hat{x} + \frac{\partial}{s_y \partial_y} \hat{y} + \frac{\partial}{s_z \partial_z} \hat{z}. \quad (3.13)$$

Using the previous definition, the Maxwell equations for the matched condition at the UPML border become

$$\nabla \times \tilde{\mathbf{H}} = i\omega\epsilon\bar{s}\tilde{\mathbf{E}} \quad (3.14)$$

$$\nabla \times \tilde{\mathbf{E}} = -i\omega\mu\bar{s}\tilde{\mathbf{H}}, \quad (3.15)$$

where \bar{s} is a diagonal tensor defined as

$$\bar{s} = \begin{bmatrix} s_y s_z s_x^{-1} & 0 & 0 \\ 0 & s_x s_z s_y^{-1} & 0 \\ 0 & 0 & s_x s_y s_z^{-1} \end{bmatrix}, \quad (3.16)$$

and where

$$s_x = \mathcal{K}_x + \frac{\sigma_x}{i\omega\epsilon}; \quad s_y = \mathcal{K}_y + \frac{\sigma_y}{i\omega\epsilon}; \quad s_z = \mathcal{K}_z + \frac{\sigma_z}{i\omega\epsilon}. \quad (3.17)$$

Note that the previous equations are defined in the frequency domain while the FDTD is defined in the time domain. However, one of the important aspects of the UPMLs is that they are designed to attenuate the field properly in the entire frequency domain and it is therefore easier to introduce their mathematical characterization in this domain. Their adaptation to the FDTD scheme is described by Taflove *et al.* [28].

Using these definitions, the \bar{s} tensor acts as a lossless medium inside the primary simulation mesh that is numerically matched to the absorbing layers and therefore the reflections are considerably damped.

For the UPML absorbers, the appropriate definitions for the σ and \mathcal{K} parameters are

$$\sigma_x(x) = \left(\frac{x}{d_x}\right)^m \sigma_{x,max}; \quad \mathcal{K}_x(x) = 1 + \left(\frac{x}{d_x}\right)^m (\mathcal{K}_{x,max} - 1) \quad (3.18)$$

$$\sigma_y(y) = \left(\frac{y}{d_y}\right)^m \sigma_{y,max}; \quad \mathcal{K}_y(y) = 1 + \left(\frac{y}{d_y}\right)^m (\mathcal{K}_{y,max} - 1) \quad (3.19)$$

$$\sigma_z(z) = \left(\frac{z}{d_z}\right)^m \sigma_{z,max}; \quad \mathcal{K}_z(z) = 1 + \left(\frac{z}{d_z}\right)^m (\mathcal{K}_{z,max} - 1), \quad (3.20)$$

which is commonly known as a *Polynomial Grading* [28] and where d_x, d_y, d_z represent respectively the thickness of the borders in the x, y and z axis and m correspond to the order of the grading. Note that the previous expressions are defined for borders that start at $x = y = z = 0$. For the correct implementation on the simulation mesh, the functions

need to be shifted to account for the correct border position. Taflove *et. al* [28] suggest that the thickness of the borders should correspond to 10 to 20 mesh cells and that the optimal values for the constant parameters are $m = 3$, $\mathcal{K}_{max} = 1$ and

$$\sigma_{x,y,z,max} = \frac{0.8(m+1)}{\eta_0 \Delta_{x,y,z} \sqrt{\epsilon_r \mu_r}}. \quad (3.21)$$

This method requires an adaptation of the equations (3.1 - 3.8), which can be also found on Taflove *et al.* [28]. It should be noted that this method presents a significant increase in the allocated memory for the FDTD solver due to the calculation of the tensor \bar{s} on the entire simulation mesh and that it requires a separate implementation alongside the one used for the standard FDTD method.

Due to the complexity and the drawbacks regarding the implementation of the UPML method, we have developed a method to tackle this issue denominated as *Multiplicative Absorbing Boundary Conditions* (MABCs). This method was designed to provide a lossless medium in the primary simulation mesh and a satisfactory dampening of both the incident and reflected fields at the borders of the mesh, without excessively compromising the solver in terms of memory usage and performance. This method was also developed taking into consideration the properties of the polynomial gradings used by the UPML method.

Consider now the points on the x axis of the simulation mesh with length L_x and a function of the type

$$f(x) = \begin{cases} 1, & \text{if } L_{x,min} < x < L_{x,max} \\ 1 - C \left(\frac{L_{x,min} - x}{L_{x,min}} \right)^m, & x \leq L_{x,min} \\ 1 - C \left(\frac{L_{x,max} - L_x + x}{L_{x,max}} \right)^m, & x \geq L_{x,max}, \end{cases} \quad (3.22)$$

which is illustrated in figure (3.2) and where $C \leq 1$ is a limiting constant, L_{min} and L_{max} represent the thickness of the absorbing borders on the lower and upper limits of the axis, respectively and m denotes the order of the polynomial function. If a similar function is used for the remaining axes of the simulation mesh and their values are multiplied by the local quantities of the electromagnetic field, it is possible to conclude that the field is damped as it penetrates the borders, while inside the primary mesh the field is unaffected. If the functions present a smooth profile, the reflections at the interfaces should be minimal and the amplitude of the reflections that travel from the inside of the borders into the primary mesh should be very small.

The algorithm to implement the MABCs consists then on integrating the electromagnetic field using the standard FDTD method and then updating the field components as

$$f(x, y, z) \equiv f = f_x(x)f_y(y)f_z(z) \quad (3.23)$$

$$D_x^{n+1} = f D_x^{n+1}; \quad D_y^{n+1} = f D_y^{n+1}; \quad D_z^{n+1} = f D_z^{n+1} \quad (3.24)$$

$$E_x^{n+1} = f E_x^{n+1}; \quad E_y^{n+1} = f E_y^{n+1}; \quad E_z^{n+1} = f E_z^{n+1} \quad (3.25)$$

$$B_x^{n+\frac{3}{2}} = f B_x^{n+\frac{3}{2}}; \quad B_y^{n+\frac{3}{2}} = f B_y^{n+\frac{3}{2}}; \quad B_z^{n+\frac{3}{2}} = f B_z^{n+\frac{3}{2}} \quad (3.26)$$

$$H_x^{n+\frac{3}{2}} = f H_x^{n+\frac{3}{2}}; \quad H_y^{n+\frac{3}{2}} = f H_y^{n+\frac{3}{2}}; \quad H_z^{n+\frac{3}{2}} = f H_z^{n+\frac{3}{2}}. \quad (3.27)$$

Analysing the previous result it is possible to observe that there is no need for an adaptation of the FDTD method and that the introduction of the absorption effect is straightforward. Although when using this method it is still necessary to allocate memory for the values of $f_x(x)$, $f_y(y)$, and $f_z(z)$, the increase in the total memory is similar to the allocation of an extra vector field. The numerical efficiency of these methods will be discussed in the following chapter.

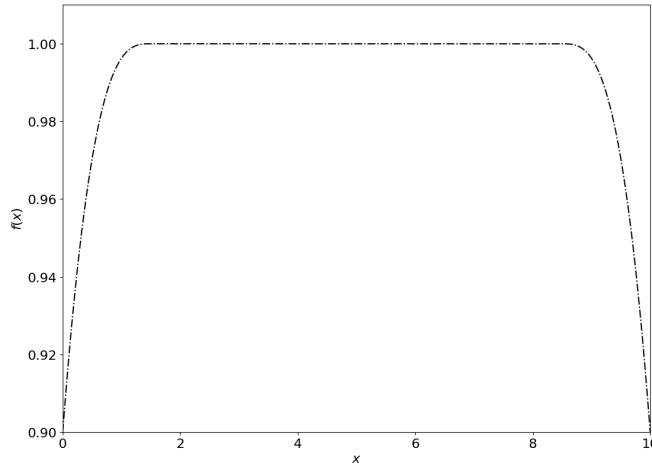


FIGURE 3.2: Example of a multiplicative boundary function used by the MABC method.
Here $L_{x,min} = 1.5$, $L_{x,max} = 8.5$ and $C = 0.9$

3.1.3 Electromagnetic field sources

Although we have examined the appropriate methods for the propagation of the field and the solutions for the associated boundary conditions problem, we have not yet discussed

the methods to implement sources that provide an electromagnetic field to the simulation mesh. This subject is also vastly studied in the literature [28] but here we will limit the discussion to the basic sources of electric field, which are sufficient for the scope of this work.

The simplest field source consists in defining the initial values of \mathbf{E} and \mathbf{B} for all points of the mesh as

$$\mathbf{E}(\mathbf{r}, 0) = -\frac{\partial}{\partial t} \mathbf{A}(\mathbf{r}, 0) - \nabla V(\mathbf{r}, 0) \quad (3.28)$$

$$\mathbf{B}\left(\mathbf{r}, \frac{\Delta t}{2}\right) = \nabla \times \mathbf{A}\left(\mathbf{r}, \frac{\Delta t}{2}\right) \quad (3.29)$$

where $\mathbf{A}(\mathbf{r}, 0)$ is the initial state of the vector potential. After this initial definition it necessary to solve the constitutive relations (2.5) and (2.6) to obtain the values of $\mathbf{D}(\mathbf{r}, 0)$ and $\mathbf{H}\left(\mathbf{r}, \frac{\Delta t}{2}\right)$ respectively and then the FDTD method will propagate field through the mesh.

Another possible solution to introduce a field into to the mesh is to observe that the current density \mathbf{J} that is present in equation (2.3) is not solved by the FDTD method and therefore any current source analytically defined as

$$\mathbf{J}(\mathbf{r}, t) = \mathbf{F}(\mathbf{r}, t) \quad (3.30)$$

may be used as an electromagnetic field source solvable with the FDTD method.

It is also important to consider the case where an external source to the system imposes a well-determined electromagnetic field in a certain region of the simulation mesh, which may also behave as a field source for the simulation. Considering then that this external field is defined as

$$\mathbf{E}(\mathbf{r}, t) = -\frac{\partial}{\partial t} \mathbf{A}(\mathbf{r}, t) - \nabla V(\mathbf{r}, t) \quad (3.31)$$

$$\mathbf{B}(\mathbf{r}, t) = \nabla \times \mathbf{A}(\mathbf{r}, t) \quad (3.32)$$

and that the analytical expressions of the vector potential $\mathbf{A}(\mathbf{r}, t)$ is known, it can be coupled to the field provided by the FDTD method as long as it is also a solution of the Maxwell equations. This source is very useful when the intent of the simulation is not to study the dynamics of the electromagnetic field but to observe how specific fields drive the dynamics of other systems.

3.2 Solving the optical Bloch equations

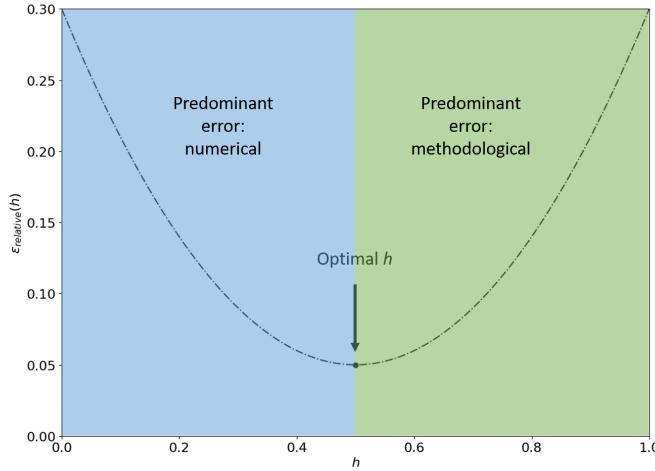


FIGURE 3.3: Schematic representation of the general numerical error per integration step. For large integration steps the error is mostly imposed by the numerical method as for very small values of h the error is predominantly governed by the computer precision.

Analysing equations (2.49) and (2.50) it is possible to conclude that both the optical Bloch equations and the dynamics of the orientation of the atomic dipoles represent a system of *Ordinary Differential Equations* (ODEs). Also, the underlying algebraic operation that describes the dynamics of density matrix elements consists in simple matrix-vector multiplications, for which the proper algorithms are well studied in the field of computer science and many programming languages and APIs present implicit solutions for the task.

Numerically speaking, the simplest approach to estimate the derivative of a function consists on the finite difference

$$\dot{y}(\vec{r}, t_n) = \frac{y(\vec{r}, t_{n+1}) - y(\vec{r}, t_n)}{h}, \quad (3.33)$$

where h corresponds to the time interval between consecutive samples of the function. Using this concept, it is possible to approximate $y(\vec{r}, t_{n+1})$ as

$$y(\vec{r}, t_{n+1}) = y(\vec{r}, t_n) + h\dot{y}(\vec{r}, t_n), \quad (3.34)$$

which is known as the *Euler Method* and presents a truncation error of $O(h)$. To reduce the error one could use an infinitesimal h but doing so increases the rounding error and

consequently the total error. Typically, the relative error per integration step as a function of h follows a trend illustrated on figure (3.3).

In order to reduce the error involved in calculating finite differences without using a very small h , it is possible to use other methods that present a significantly better truncation error. Among a considerably sized list of algorithms created to solve ODE systems, there are two families of methods that are worth discussing.

The first methods family to be discussed are the *Linear Multistep Methods*. The name of these algorithms is justified by the linear combination of previous points and derivatives to compute the value of a function in a future instant. From this family, a method that stands out is the *Two-step Adams-Basforth* algorithm, which calculates the temporal evolution of an ordinary differential equation using the intrinsic properties of the *Fundamental Theorem of Calculus* [29]:

$$\mathbf{y}(t_{n+1}) = \mathbf{y}(t_n) + \int_{t_n}^{t_{n+1}} \mathbf{y}'(t) dt, \quad (3.35)$$

where the integral term in the equation is approximated by interpolation techniques as follows

$$\mathbf{A} = \int_{t_n}^{t_{n+1}} \mathbf{y}'(t) dt = \int_{t_n}^{t_{n+1}} f(\mathbf{y}(t), \mathbf{Z}, t) dt \quad (3.36)$$

$$\mathbf{P}(t) = f(\mathbf{y}_n, \mathbf{Z}_n, t_n) \frac{t - t_{n-1}}{t_n - t_{n-1}} + f(\mathbf{y}_{n-1}, \mathbf{Z}_{n-1}, t_{n-1}) \frac{t - t_n}{t_{n-1} - t_n} \quad (3.37)$$

$$\mathbf{A} \approx \int_{t_n}^{t_{n+1}} \mathbf{P}(t) dt = \int_{t_n}^{t_{n+1}} \left(f(\mathbf{y}_n, \mathbf{Z}_n, t_n) \frac{t - t_{n-1}}{t_n - t_{n-1}} + f(\mathbf{y}_{n-1}, \mathbf{Z}_{n-1}, t_{n-1}) \frac{t - t_n}{t_{n-1} - t_n} \right) dt, \quad (3.38)$$

Since t_{n-1} , t_n and t_{n+1} are equally spaced, equations (3.35) and (3.38) become

$$\mathbf{A} = \frac{3}{2}hf(\mathbf{y}_n, \mathbf{Z}_n, t_n) - \frac{1}{2}hf(\mathbf{y}_{n-1}, \mathbf{Z}_{n-1}, t_{n-1}) \quad (3.39)$$

$$\mathbf{y}(t_{n+1}) \approx \mathbf{y}(t_n) + \frac{3}{2}hf(\mathbf{y}_n, \mathbf{Z}_n, t_n) - \frac{1}{2}hf(\mathbf{y}_{n-1}, \mathbf{Z}_{n-1}, t_{n-1}), \quad (3.40)$$

where f is a system of equations that described the ODEs, \mathbf{y} is the state of the system, \mathbf{Z} are a set of time-dependent variables that affect the state of the system but may not be directly integrated by the algorithm, and t represents the temporal instant. Note that the index n indicates the discrete temporal instants where the state of the system is sampled and therefore $\mathbf{y}_{n+1} = \mathbf{y}(t_n + h)$.

Analysing this algorithm with a Taylor series expansion of $f(t_n, \mathbf{y}_n)$ is possible to conclude that this method has a $\mathcal{O}(h^3)$ truncation error [30].

An alternative to the Linear Multistep Methods are the *Runge-Kutta family* methods. In the literature, the most widely known methods from this family are the *Second and Fourth Order Runge-Kutta Methods* and usually simply referred as *RK2* and *RK4*, respectively. Both these methods assume that the ODEs that describe the dynamics of the system and the corresponding initial state are known. More concisely,

$$\dot{\mathbf{y}} = f(\mathbf{y}, \mathbf{Z}, t) \quad \text{and} \quad \mathbf{y}(t_0) = \mathbf{y}_0, \quad (3.41)$$

are the necessary conditions to implement the RK2 and RK4 methods and where the function f , \mathbf{y} , \mathbf{Z} and t have the same interpretation as presented in the previous method.

If we now consider the RK2 method it is possible to determine the evolution of the state of the system as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + hf \left(\mathbf{y}_n + \frac{1}{2}hf(\mathbf{y}_n, \mathbf{Z}_n, t_n), \mathbf{Z}_{n+\frac{1}{2}}, t_n + \frac{1}{2}h \right), \quad (3.42)$$

where $\mathbf{Z}_{n+\frac{1}{2}} = \mathbf{Z}(t_n + \frac{1}{2}h)$. This method consist then on approximating the value of \mathbf{y}_{n+1} by estimating the slope of function f at $t_n + \frac{1}{2}h$ and the order of the total accumulated error per integration step is $\mathcal{O}(h^2)$.

Similarly, using the RK4 method it is possible to approximate the evolution of state of the system as

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(\mathbf{K}_1 + 2\mathbf{K}_2 + 2\mathbf{K}_3 + \mathbf{K}_4), \quad (3.43)$$

where the terms \mathbf{K}_1 , \mathbf{K}_2 , \mathbf{K}_3 and \mathbf{K}_4 are defined as

$$\mathbf{K}_1 = f(\mathbf{y}_n, \mathbf{Z}_n, t_n) \quad (3.44)$$

$$\mathbf{K}_2 = f\left(\mathbf{y}_n + \frac{h}{2}\mathbf{K}_1, \mathbf{Z}_{n+\frac{1}{2}}, t_n + \frac{h}{2}\right) \quad (3.45)$$

$$\mathbf{K}_3 = f\left(\mathbf{y}_n + \frac{h}{2}\mathbf{K}_2, \mathbf{Z}_{n+\frac{1}{2}}, t_n + \frac{h}{2}\right) \quad (3.46)$$

$$\mathbf{K}_4 = f(\mathbf{y}_n + \mathbf{K}_3, \mathbf{Z}_{n+1}, t_n + h), \quad (3.47)$$

which estimate the slope specified by function f on equation (3.41) at different temporal instants. From a mathematical standpoint it is possible to interpret these terms as follows

- K_1 is the increment based on the slope at the beginning of the interval (equivalent to the Euler method);
- K_2 is the increment based on the slope at the midpoint of the interval, using $y + \frac{h}{2}k_1$;
- K_3 is again the increment based on the slope at the midpoint, but now using $y + \frac{h}{2}k_2$;
- K_4 is the increment based on the slope at the end of the interval, using $y + hk_3$

The fourth-order Runge-Kutta method exhibits a local truncation error of $O(h^5)$ and a total accumulated error of $O(h^4)$.

Although both families of methods provide good solutions for the implementation of a Bloch equations solver, analysing equations (3.40), (3.42) and (3.43) it is possible to conclude that while the three methods require that at least two values of the electric field are preserved in memory, the Adams-Bashforth algorithm also requires that the previous value of y are kept in memory, which in our case corresponds to storing values the density matrix and the orientation of the atoms in space and therefore represents a considerable increase in the memory requirements. Examining the truncation error of each method it is also possible to conclude that the RK4 surpasses the Adams-Bashforth method. Finally, due to the similarities between the RK2 and the RK4 methods it is easy to implement a solution that fits both algorithms. Considering these properties, we opted to implement the Runge-Kutta family methods with the possibility to chose whether the simulation should be performed using the RK2 or the RK4. It is important to note that although the RK4 presents a better truncation error, it requires more memory and performs more algebraic operations and thus there must be a balance between numerical accuracy, integration speed and simulation dimension.

3.3 Coupling the solvers

Equipped with the appropriate numerical methods for both the Maxwell and the Bloch equations it is now important to discuss the correct approach to couple the solvers.

In our implementation, the simulation mesh is composed of Yee cells and is used for sampling both the electromagnetic field and also the elements of the density matrix, the orientation vector of the atoms in space and the macroscopic polarization of the atomic gas. For simplicity, the latter quantities are sampled on the centre of the Yee cells. However, this choice implies that it is necessary to interpolate the values of the electric field

from the faces to the centre of the cells to solve equations (2.49) and (2.50) numerically. The interpolated electric field is calculated as

$$E_x|_{i,j,k}^{Centre} = \frac{1}{2} (E_x|_{i,j,k}^{Face} + E_x|_{i+1,j,k}^{Face}) \quad (3.48)$$

$$E_y|_{i,j,k}^{Centre} = \frac{1}{2} (E_y|_{i,j,k}^{Face} + E_y|_{i,j+1,k}^{Face}) \quad (3.49)$$

$$E_z|_{i,j,k}^{Centre} = \frac{1}{2} (E_z|_{i,j,k}^{Face} + E_z|_{i,j,k+1}^{Face}). \quad (3.50)$$

$$(3.51)$$

To solve equation (2.5) numerically it is also necessary to interpolate the polarization of the atomic gas from the centre to the faces of the cells, which can be computed according to

$$P_x|_{i,j,k}^{Face} = \frac{1}{2} (P_x|_{i,j,k}^{Centre} + P_x|_{i-1,j,k}^{Centre}) \quad (3.52)$$

$$P_y|_{i,j,k}^{Face} = \frac{1}{2} (P_y|_{i,j,k}^{Centre} + P_y|_{i,j-1,k}^{Centre}) \quad (3.53)$$

$$P_z|_{i,j,k}^{Face} = \frac{1}{2} (P_z|_{i,j,k}^{Centre} + P_z|_{i,j,k-1}^{Centre}). \quad (3.54)$$

$$(3.55)$$

Continuing the discussion of the integration of equations (2.49) and (2.50) using the Runge-Kutta family methods, the term \mathbf{Z} in equations (3.42 - 3.47) refers to the electric field \mathbf{E} . Further analysing equations (3.42 - 3.47) it is possible to observe that both Runge-Kutta methods require the value $\mathbf{E}_{n+\frac{1}{2}}$ and considering the FDTD method it is only possible to obtain the values \mathbf{E}_n . The value can however be approximated as

$$\mathbf{E}_{n+\frac{1}{2}} = \frac{1}{2} (\mathbf{E}_n + \mathbf{E}_{n+1}), \quad (3.56)$$

which for small values of Δ_t should not induce a significant error in the calculations. This approximation also yields that the Maxwell equations should be solved before the Bloch and the dipole orientation equations for each integration step.

Finally, an important aspect relevant for the choice of the numerical parameters of the simulation is that we opted to use the same value of Δ_t for both solvers. However, each solver is responsible for different physical phenomena which may have very distinct timescales. Thus, the value of Δ_t should be chosen taking into consideration the faster

dynamics present on the system of coupled equations, which in turn may have an impact on the values of Δ_x , Δ_y and Δ_z due to the CFL condition.

3.4 Conclusions

In this chapter, we analysed some of the available numerical methods to implement the simulation of our model and concluded that the best-suited options for our software were the FDTD method and the Runge-Kutta family methods to solve the Maxwell and the Bloch equations, respectively. Furthermore, we discussed the important aspects regarding the coupling of the methods. This combination of methods serves as a good solution for the implementation of a general solver of light-matter interactions with moderate memory consumption and that supports GPGPU computing technologies.

Moreover, we also discussed the methods to introduce fields sources in the simulation and the boundary conditions problem, for which we reviewed the concepts of UPMLs and introduced a method developed during this thesis and denominated by MABCs.

In the next chapter, we will review some concepts of the GPGPU computing paradigm and we will use the methods presented in this chapter to discuss the implementation of our solver.

Chapter 4

Implementation of the solver

The previous chapters were dedicated to the discussion of the physical and numerical models relevant to the implementation of a solver for the study of the interaction of electromagnetic fields with atomic gases.

This chapter describes the more relevant aspects behind the engineering project of developing the solver, including the main technical features. Following the best practices associated with project management, we begin by identifying the objectives, constraints and the available resources and technologies. Next, we discuss the appropriate implementation scheme and present the conceived solution. Lastly, we validate the outputs of the project through a characterization of the numerical and computational performance of the solver as well as its limitations.

In particular, the first section reviews the underlying concepts of the GPU computing technologies and the available APIs that simplify the development of the solver. Section two discusses the design of our solver and describes the implementation details of the various classes and libraries that enhance its modular architecture. The closing sections present the general tests of our solver, namely the comparison of the GPU and CPU performances, the stability of the numerical methods and the characterization of the implemented boundary conditions.

4.1 GPU computing

While current day CPUs are very powerful for common usage such as document editing and other everyday tasks, their capability to process large sets of data is low, which stands as big issue for scientific computing. Due to this fact, through the last decades, there have

been several proposals and attempts of imposing parallel computing paradigms with distributed computing protocols such as MPI or shared memory clusters. The general problem with these solutions is that they are very expensive and mostly only available in research facilities.

To solve this problem, researchers delved into the multicore architecture of consumer grade GPUs in order to create affordable high-performance computation machines. In the beginning of this change in mindset software developers had to mask their data and calculations as textures and calculations of shader geometry since GPUs were only capable of performing such operations. This paradigm once again changed when in 2007 Nvidia Corporation® launched the *CUDA Application Programming Interface* (CUDA API). This API enabled the execution of standard algebraic operations within Nvidia GPUs without having to adopt complex data masking strategies and the term *General Purpose Graphical Processing Units* (GPGPUs) was created.

The main idea of using GPUs instead of CPUs for performing dense computations comes from the core architecture of the graphics processing unit. These pieces of hardware were created to accelerate the processing of computer graphics and as so, their architecture and instruction pipeline is very different from the ones used in CPUs [31]. To process graphics, GPUs use a large number of small processing units known as cores to compute the geometry of two or three-dimensional textures in parallel. Each core processes a small fraction of the global graphical data. While these operations could be performed by a multicore CPU (and possibly faster when comparing core-to-core performance), due to the large number of "slow" cores working in parallel inside a GPU, the overall computation is significantly faster on a GPU.

One significant disadvantage of GPUs is that in most cases they only support a parallel computation paradigm known as *Single Instruction on Multiple Threads* (SIMT) [32], while most consumer grade CPUs support *Multiple Instruction on Multiple Data* (MIMD) [33]. In short, these models impose a base scheme for the development of parallel software. The MIMD model admits that each core or worker in a parallel application can execute a different task from the one issued to the remaining workers, while the SIMT model corresponds to a set of workers where the whole group executes the same task over a different set of input data.

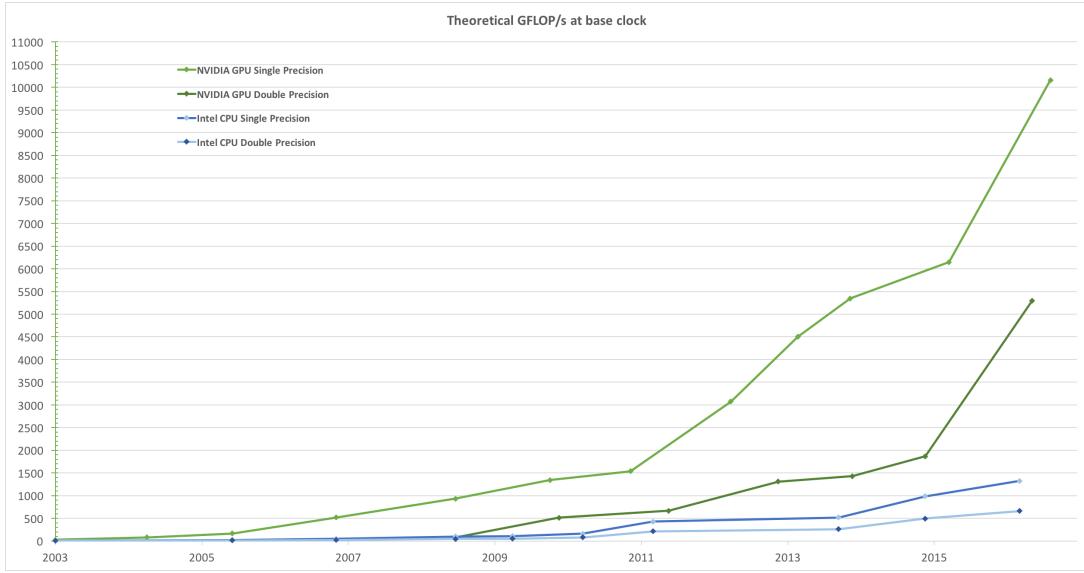


FIGURE 4.1: Comparison between CPU and GPU floating point operations per second performance (source: [34])

The development of the CUDA technology permitted significant speedups of mathematical operations in general household computers. Figure 4.1 contains a recent performance comparison of CPU and GPU calculations. Although the development of code for GPUs became fairly simpler with CUDA, there are still some difficulties associated with programming them and it only becomes reasonable to do so when one needs to parallelize software that handles datasets that are excusively large to be treated using CPUs. For this reason, the parallelization of tasks traditionally follows the scheme seen on figure 4.2 where the CPU is responsible for the single-thread tasks and for the coordination of the GPU workers.

Apart from CUDA, there is another API for GPGPU computing named OpenCL which is open-source and compatible with several GPU chipset brands such as Nvidia[®], AMD[®] and Intel[®]. Although OpenCL could become an overall better platform for this type of computing, as of today it still is outperformed by CUDA and also lacks some its compelling features ranging from proper debugging and profiling tools to data manipulation methods.

To incorporate GPGPU functionalities into a standard program the common methodology is to declare *Kernel* functions that can only process variables declared on the device memory. The authors of [35] and [36] present a good introduction for declaring such methods with the CUDA and OpenCL API respectively, as well as, implementations of

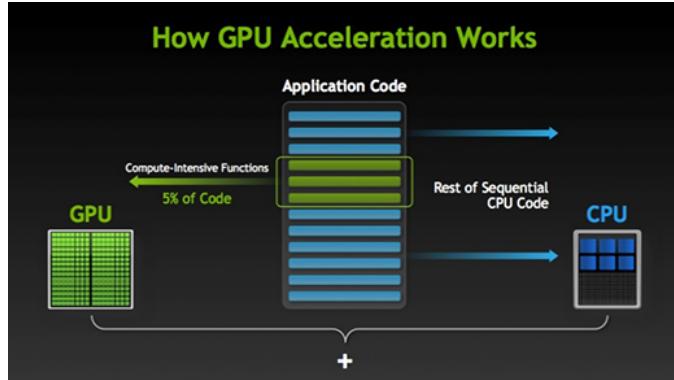


FIGURE 4.2: Fundamental scheme of GPU computing (source:[39])

the many standard algorithms. An alternative to explicitly declaring *Kernels* is to use libraries such as *Thrust* [37] and *ArrayFire* [38], which stand as frameworks for standard data structures and their manipulation. When adopting these libraries is important to keep in mind the scope of the final program since for example, *Thrust* is a CUDA library and as so is well optimized for Nvidia® cards but is restricted to them, while *ArrayFire* is capable of working with both CUDA and OpenCL but works on a higher level implementation and therefore in some cases may cripple the speedups. In our case we opted to use *ArrayFire* as it provides the following benefits to our solver:

- Simple solutions that handle matrix and vector algebra;
- Simple data manipulation methods;
- Automatic determination of kernel geometries that grant optimal thread occupancy;
- Syntax is similar to the one used by other APIs such as Numpy;
- Available in C++ and Python which renders the processing of the output files easy;
- The multiple back-end strategies permits to compare the GPU and CPU implementations without having to develop the code twice.

4.2 Solver design and implementation scheme

Having decided the numerical methods to study the Maxwell-Bloch equations and the underlying computing technology, it becomes necessary to define a software development model to approach the problem. Our implementation of the solver must be able to address several challenges, in particular it must

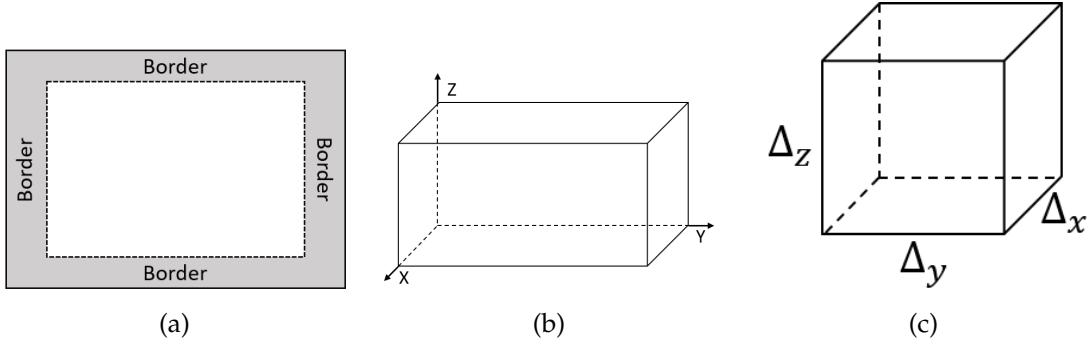


FIGURE 4.3: Components of the spatial simulation domain. (a) is a two-dimensional representation of the scheme of the boundary conditions in the simulation domain; (b) represents the main simulation grid; (c) portrays the scheme of the Yee cells that compose the mesh.

- be developed as a modular software capable of supporting the integration of other modules, each constituting a solver on its own and focused on a different aspect of atomic gases. In fact, this modular approach is already present in our solver of the Maxwell-Bloch equations since the Maxwell equations and the Bloch equations have specific sub-solvers which exchange data;
- adapt the specific aspects of the numerical models and algorithms discussed in the previous chapter to the GPU computing paradigm;
- support multiple boundary conditions and electromagnetic field sources;
- be equipped with a diverse set of standard structures that permit the user to develop new modules and implement arbitrary numerical methods;
- provide some level of abstraction, which allows the user not to be concerned with the specific aspects of how the calculations are accurately done on the multiple backends, including the GPU.

It is important to note that, as discussed in the previous chapter, our model imposes that all solvers share a general simulation domain, which is represented in figure (4.3). This implies that the communication between the solvers must be performed through an exchange of quantities that are sampled on the same unit cells. This general simulation domain is also used to represent the imposed boundary conditions for all solvers.

In terms of the execution model, the structure behind the solver is depicted in figure (4.4). Although this scheme is very close to the structure of the models discussed in the previous chapter, for the solver to overcome the preceding challenges, it must be implemented using a scheme which is designed to be performant, numerically accurate and to

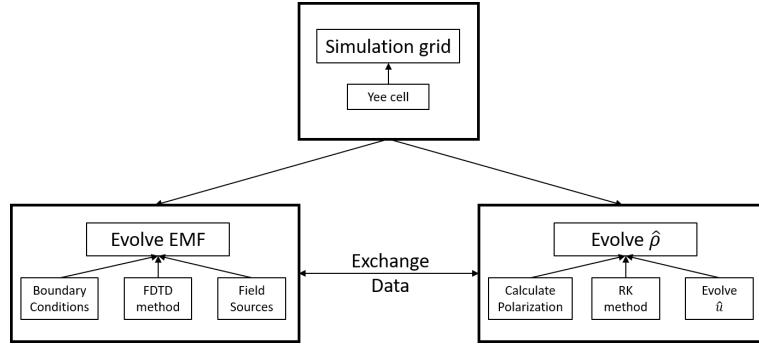


FIGURE 4.4: Overview of the simulation execution model

render easy the combination with other solvers in the future. This design implies a laborious engineering and computational physics exercise since the solver has to be developed as a modular architecture that supports various C++ modules responsible for solving the physical equations and handling data operations.

As a result, the implementation scheme differs from the scheme of the numerical model and in practice, the implementation is composed of various modules. The modules are designed to create abstraction layers and communicate seamlessly with each other through a main script that deploys the simulation and is rather simple to be developed by the end user. Figure (4.5) presents a view of the hierarchy of the C++ classes and libraries that compose our code and which may be succinctly described as:

- **Units** - class that converts the values of the simulation variables between unit systems;
- **YeeMesh** - class that contains the information regarding the simulation mesh geometry;
- **Field** - class used to handle arbitrary vector and scalar quantities;
- **Matrix** - class that manipulates constant matrices used in the simulation;
- **TensorField** - class that handles arbitrary tensor fields and space-dependent matrices;
- **Boundary** - class related to the definition of the boundary conditions of the mesh;
- **EMF** - class that contains the solver of the Maxwell equations;
- **Bloch** - class used to solve the Bloch equations;

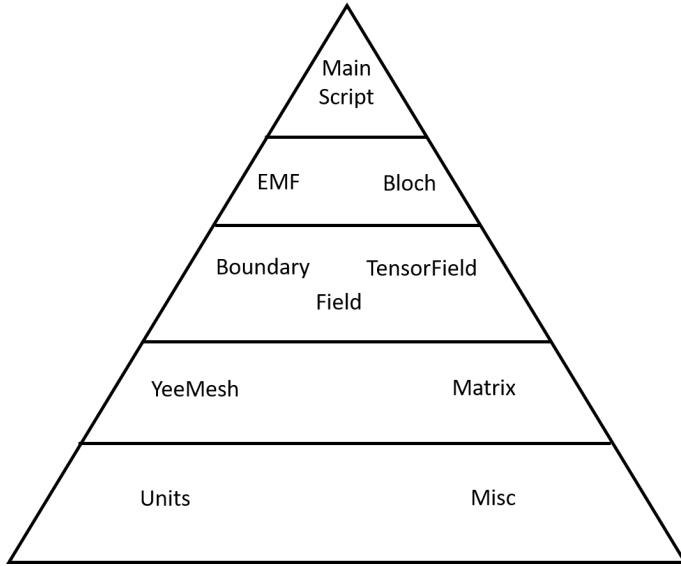


FIGURE 4.5: Solver classes and libraries hierarchy.

- **Misc** - library that contains auxiliary functions that are not related to any particular class.

In the following sections, we present an in-depth description of each class and the corresponding auxiliary functions. It should be noted that from now on, the term **array** refers to variables of type **af::array** which are the arrays provided by the ArrayFire API and may be stored both on the GPU and CPU depending on the back-end that the user selects. The term **vector** implies the usage of the C++ standard vectors.

4.2.1 Scaling of physical equations and natural units

A common issue to all branches of computational physics is the scaling of the physical equations and quantities of a model so that there is minimal numerical error introduced by the software. The elemental effect behind this problem is the rounding effect that occurs when a computer performs arithmetic operations between two quantities, due to the finite numerical representation precision intrinsic of digital systems. In fact, for modern computers capable of representing numbers using 64 bits compliant with the *IEEE 754 - 2008* standard [40] the *numerical unit roundoff error* also known as *machine epsilon* is 2^{-52} .

The typical quantities involved in the simulation of the Maxwell-Bloch equations vary from very large to very small numbers when using the *SI units system*, typically varying over 10 orders of magnitude. For this reason, it is advisable to use a natural system of units where the orders of magnitude of the quantities are not so dissimilar.

As an example, consider equation (2.5) where the typical values for the electric and polarization fields are 10^4 and 10^{-8} respectively, in the SI units system. However, using an units system as the *particle physics natural Lorentz-Heaviside* system where $\epsilon_0 = 1$ and $\mu_0 = 1$, the values of the electric and polarization fields become approximately 10^{-3} and 10^{-6} respectively, greatly reducing the difference of the orders of magnitude.

On our solver there is a class denominated **Units** that contains the information regarding the conversion scales between the natural and SI unit systems, for each of the following fundamental quantities¹: **length**, **mass**, **time**, **temperature** and **charge**. With this set of values and using this class, it is possible to obtain the conversion factor of any physical quantity defined in the SI system into the natural units systems and vice-versa, as long as the user provides a vector that expresses the physical quantity in terms of the fundamental quantities and a vector with the corresponding exponents. For simplicity, the names of the fundamental quantities are defined inside an *enum class* called **UnitsNames**. It is also possible to use this class to directly obtain the value of a quantity in the defined system of units.

As an example of usage, consider the physical quantity **A** whose unit is $C \cdot m$. To obtain the value of **A** in the natural system the user needs only to create an instance of the **Units** class, defining the desired conversion scales and then perform the operation: $A_{\text{natural}} = \text{Units}.\text{convertToNatural}(A_{\text{SI}}, \{\text{UnitsNames} :: \text{CHARGE}, \text{UnitsNames} :: \text{LENGTH}\}, \{1, 1\})$.

To further simplify the construction of an instance of the **Units** class, it is also possible to call the class constructor using a pre-specified system of units. The currently implemented systems are the SI system, the *particle physics natural Lorentz-Heaviside* system and the *Hartree atomic units* system.

Finally, due to relevancy of some physical constants to our simulation, the **Units** class provides methods to easily obtain the values of c , \hbar , e , G , K_B , m_e , μ_0 and ϵ_0 in both the SI and the user defined system of units.

4.2.2 The Yee mesh

As discussed in the previous chapter, our solver uses the FDTD method to determine the evolution of the electromagnetic field, which relies on a mesh composed of Yee cells to simulate the propagation of the electromagnetic field and the same mesh is used to sample all the quantities simulated by our solver. This mesh is stored on an instance of

¹In Portuguese this term refers to the concept of "grandezas fundamentais"

the **YeeMesh** class that is created by providing the class constructor with an instance of the **Units** class, the number of points in each axis and the corresponding values of Δ_x , Δ_y , Δ_z and Δ_t converted to the corresponding system of units. The class constructor also receives as arguments regarding the numerical precision of the quantities relevant to simulation which by default is defined as 64 bits.

An important aspect regarding the **YeeMesh** class is that by virtue of wanting to develop a general solver, the minimum number of points per axis is one, which allows the simulation of physical phenomena in one, two and three dimensions.

The **YeeMesh** class provides methods to obtain the points on each dimensional axis. However, since the design of the Yee cell permits the sampling of quantities in the centre, faces and edges of the structure, the values of the points are sensible to the sampling spots and therefore the user must provide the desired position types upon the usage of the **YeeMesh.x**, **YeeMesh.y** and **YeeMesh.z** methods. For simplicity the possible positions of sampling are **CENTER**, **FACE_X**, **FACE_Y**, **FACE_Z**, **EDGE_X**, **EDGE_Y** and **EDGE_Z** which are elements of a *enum class* denominated **Grid**.

Finally, it is also possible to save the details of the simulation mesh using the method **YeeMesh.save()**. This method saves the information regarding the number of points per axis and the corresponding spacings and dimension. It is also possible to decide whether this information is saved using the natural systems of units of the SI system since the instances of this class are equipped with an instance of the **Units** class.

4.2.3 The Field class

The **Field** class is designed to fulfil the need for a standard structure to sample both arbitrary vector and scalar quantities or fields on the simulation mesh. The instances of this class contain the information regarding the **YeeMesh** object and the positions of the Yee cells that are used to sample the field quantity and also an array with the local values of the quantity. The specification of the sampling positions is imposed using a variable with type **FieldType** that corresponds to an *enum class* which permitted values are: **FACE**, **EDGE**, **CENTER** and **CENTER_VECTOR**. This distinction is not only important to the correct sampling on the mesh but also because it constrains the shape of the array used to store the values of the field. As so, instances of **Field** with type **FieldType::CENTER** are limited to scalar quantities while the remaining types are limited to vector quantities.

This class is also designed to prevent the user from performing arithmetic operations between quantities that are not sampled on the same **YeeMesh** or the same sampling positions. For this reason, all the arithmetic operators for this class are *overloaded* and perform the necessary verifications to ensure that the fields are numerically compatible. However, there are many cases where it is necessary to perform arithmetic operation between fields with different sampling positions and to simplify this process, the instances of the **Field** class have a method called **Field.as** that takes the desired **FieldType** as an argument and returns new **Field** whose values are interpolated to the correct positions. To preserve time and memory, if the **Field** object is already of the correct type, the method returns the original instance. This method is crucial to ensure the modularity of the solver since, for example, the Polarization provided by the Bloch equations and the electric field used in equation (3.4) are sampled on centres and the faces of the Yee cells, respectively.

4.2.4 Tensor Fields

Apart from the **Field** class there is another standard support class in our solver denominated **TensorField**. The instances of this class consist on a wrapper for a matrix of scalar and vector quantities sampled on the simulation mesh and are primarily used to store the values of the density matrix and the values of Ω_{ij} present on equation (2.49).

The constructor of this class takes as arguments an instance of the **YeeMesh** where the quantities are sampled, the number of rows and columns of the matrix and a variable of type **TensorFieldType** that, like the type **FieldType**, is defined by an *enum class* that specifies the sampling positions of the matrix elements on the Yee cells.

Analysing this structure it is possible to argue why the element of the matrix of a **TensorField** are not simply instances of **Field**. To better understand this design choice consider a matrix with $N = N_{rows} \times N_{cols}$ elements. Although the **TensorField** objects gather information regarding many quantities, it is imposed beforehand that they are sampled in the same **YeeMesh** and in the same Yee cell positions and therefore if the matrix elements are **Field** objects, the information regarding the **YeeMesh** object and the sampling position is stored N times, which in this case represents a misuse of memory. Another important detail is that the elements of a **TensorField** matrix may be an empty array in order to preserve memory.

When an instance of the **TensorField** class is created, all the elements of the matrix correspond to empty arrays. However, the class is equipped with the method **TensorField.setValue()** that sets a given array on a specified element of the matrix, as long as the array has the appropriate shape for the simulation mesh.

Finally, all the arithmetic operators are overloaded also for this class and as for the **Field** class, it is possible to safely perform operations between **TensorField** objects, variables of types **double** and also **Field** instances. An important note regarding these operations is that the overloaded operators account for the existence of empty arrays in the matrix of a **TensorField** object and simply ignore any calculations that involve those arrays.

4.2.5 Constant Matrix

To further simplify the implementation of the solver of the Bloch equations we have a simple class called **Matrix**. This class encloses a matrix of $N = N_{rows} \times N_{cols}$ **double** type values and is primarily used to store the values of ω_{ij} and Γ_{ij} present in equation (2.49). The class constructor is very simple and only receives the matrix information as a vector of vectors.

The main advantage of using this class instead of a simple vector of vectors to store the matrices is that due to the overloading of the arithmetic operators it is possible to perform operations between **Matrix** and **TensorField** objects without any effort. The overloaded operators also verify that the dimensions of both the **Matrix** and **TensorField** objects respect the algebraic operation conditions.

4.2.6 General auxiliary functions

The implementation of our solver requires the development of several methods that deploy and simplify the usage of mathematical operations that are crucial to this work. However, these methods are quite general and handle different data types and thus they are implemented on an auxiliary library called **Misc** rather than as member functions of specific classes. The methods supported by this library as described as follows:

- **grad** - Function that calculates the gradient of a scalar quantity. The arguments of this method are an instance of **Field** and the **FieldType** desired for the output. The function returns a **Field** object whose vector quantity is sampled on the desired Yee cell positions;

- **div** - Function that calculates the divergence of a vector quantity. The arguments of this method are an instance of **Field** and the **FieldType** desired for the output. The function returns a **Field** object whose scalar quantity is sampled on the desired Yee cell positions;
- **curl** - Function that calculates the curl of a vector quantity. The arguments of this method are an instance of **Field** and the **FieldType** desired for the output. The function returns a **Field** object whose vector quantity is sampled on the desired Yee cell positions;
- **cross** - Function that calculates the cross product of two vector quantities. The arguments of this method are two instances of **Field** and the **FieldType** desired for the output. The function ensures that both input **Field** objects have the same **FieldType** and are sampled on the same **YeeMesh**. The return value is an instance of **Field** whose values are a vector quantity corresponding to the cross product of the inputs.
- **interpolate** - Function that interpolates the values of a **Field** object into a specific sampling position on the Yee cells. The arguments of this method are an instance of **Field** and a **FieldType** variable that specifies the sampling positions of the output. The function returns a **Field** object interpolated into the desired positions.
- **zeros** - Function used to create a uniform null **Field**. The arguments of the method are an instance of **YeeMesh**, and a variable of type **FieldType**. The return value of this function is a **Field** of the provided type and whose values are zero on every Yee cell.
- **normalizeVector** - Function that normalizes a vector quantity. The input of the method is an array that represents a vector quantity. The function returns an array with the normalized local vectors.
- **magnitude** - Function that calculates the magnitude of a vector quantity. The input of the method is an array that represents a vector quantity. The function returns a scalar array with the magnitude of the local vectors.

This library also provides two auxiliary methods used to simplify the creation of boundaries in the simulation mesh called **makeABCBoundary** and **makeUPMLBoundary**. These methods verify that the input parameters are compatible with the simulation properties and will be further analysed in the following section.

4.2.7 Implementation of boundary conditions

In the previous chapter, we discussed the boundary condition problem and presented three different methods to handle both periodic and absorbing simulation boundaries. Considering the case of a three-dimensional simulation, there are six boundaries or borders surrounding the mesh. However, it is possible that user desires to have a different boundary condition for each border or for each axis. This case is also relevant for the simulation of one and two-dimensional systems, since, in our solver, these simulations are performed by “collapsing” the non-relevant axis to a single point and thus their boundary conditions must be periodic in order to avoid numerical aberrations. These difficulties are solved using the **Boundary** class which is designed to support MABCs, UPMLs and periodic boundary conditions.

By default, all borders are associated with periodic boundary conditions. Nonetheless, it is possible to assign a **Boundary** object to the desired simulation borders and change the boundary condition scheme. The **Boundary** class constructor receives three arguments: a variable of type **BoundaryType** that specifies if the boundary condition corresponds to the periodic, the MABC or the UPML type; a variable of type **BoundaryAxis** that indicates the desired border for the boundary condition and **BoundaryUnion** type variable that corresponds to a C++ *union* and that contains the information regarding the boundary condition numerical parameters. For the UMPL and MABCs boundaries the **BoundaryUnion** variable represents a C++ *structure* with the values required by equations (3.18 - 3.20) and (3.22), respectively. To obtain the array of values that defines the boundary condition the user needs only to use the method **Boundary.values**.

Finally, the methods **makeABCBoundary** and **makeUPMLBoundary** of the **Misc** library simplify the creation of **Boundary** objects for MABCs and UMPLs, respectively. The user must only specify the simulation border and the numerical parameters of the border and the methods return the corresponding **Boundary** instances.

4.2.8 Implementation of electromagnetic field sources

The previous chapter discussed the types of electromagnetic field sources that are relevant for our solver and we now present their implementation.

For the simplest case, the user needs only to define the initial state of the electric and magnetic fields, as presented on equations (3.28) and (3.29), by creating instances of the

Field class. In the following section, we revisit this subject with methods that further simplify the initialization of typical electromagnetic field profiles.

As for the remaining source types, their implementation is notably similar. The user needs to define a function that returns an array with the field values using the **std::function** type from the C++ standard library and whose arguments are three arrays containing the points in x , y and z axis and a **double** type variable that represents temporal instant t . To simplify the declaration of the methods, it is possible to use the **fieldFunctionTimeDep** type that is structured for user defined field source functions. It is important to note that for the current sources presented on equation (3.30) the user need only to define the functions that describe J_x , J_y and J_z , while for the external field sources the user needs to specify the functions that describe the vector components of both electric and magnetic fields.

4.2.9 The solver of the Maxwell equations

We now present our solver of the Maxwell equations using the FDTD method that is implemented under the **EMF** class which is also responsible for every aspect of the manipulation of the electromagnetic field. The arguments of the constructor of the class are a **YeeMesh** object and a set of **boolean** variables that specify whether the previous values of each component of the electromagnetic field are kept in memory or not. This constructor is responsible for creating a **Field** object for each component of the electromagnetic field on the correct positions of the Yee cells using the **zeros** function and for setting the initial values of some control variables including the information regarding the current iteration number and the types of boundary conditions used in the simulation.

Using the method **EMF.addBoundaryCondition** it is possible to assign a **Boundary** object to the desired border of the simulation mesh. This method verifies the **BoundaryType** and determines if it is possible to assign the **Boundary** to the mesh since axis with a single point only support periodic boundaries. Using the **BoundaryType** variable, the method also updates the value of the control variables regarding the types of the existing boundary conditions on the simulation mesh.

Considering now the specification of electromagnetic field sources, there are five member functions of the **EMF** class that implement this functionality. The first two methods called **EMF.addPlaneWave** and **EMF.addGaussianPulsePlaneWave** are very similar and add a field source to the simulation by specifying the initial states of the electric and

magnetic fields. The arguments of these functions are a **PropagationDir** variable that defines the direction of propagation of the field, **EFieldPol** that indicates the axis of polarization of the electric field, the wave-number that characterizes the plane wave, the initial phase and the electric field intensity. The latter method also receives the position of the peak and the *Full Width Half Maximum* (FHW) that characterize the Gaussian function used to modulate the spatial properties of the field. The remaining methods called **EMF.addExternalCurrent**, **EMF.addExternalEField** and **EMF.addExternalBField** implement the electromagnetic field sources via external fields and their arguments are three variables of type **fieldFunctionTimeDep** that correspond to the functions that define the vector components of the external sources. These methods also accept a **nullptr** for the components of the field with null value.

Equipped with the specified boundary conditions and field sources, it is possible to initialize the simulation of the electromagnetic field using the method **EMF.initialize**. Note that although in the scope of this work, only the polarization is non-null, this class is a general solver of the Maxwell equations and therefore it needs to account for the remaining physical quantities. Using the provided **Field** instances, the method updates the values of **D(r,0)**, **E(r,0)**, **B(r,0)** and **H(r,0)** and imposes the leapfrog scheme between the electric and magnetic fields using the Euler method. This method also verifies the existence of MABC type boundaries on the simulation mesh and if there are any it calculates and stores the appropriate values of the multiplicative factors using equation (3.22) for both edge and face fields. This distinction is important since the values of the multiplicative factors are slightly different depending on the sampling positions. Finally, this method also locks the addition of new boundary conditions and field sources to avoid user implementation bugs that may cause instabilities on the simulation.

After the initialization routine, it is possible to simulate the propagation of the electromagnetic field using the **EMF.push** method. This method receives three **Field** objects that represent the external current density (**J**), polarization (**P**) and magnetization (**M**) and is divided in two main parts called **EMF.pushElectric** and **EMF.pushMagnetic** that solve the electric and magnetic components of the Maxwell equations, respectively. If the simulation mesh does not contain any UPML boundary conditions, these sub-methods simply apply the FDTD algorithm described in equations (3.1 - 3.8). However, if this boundary condition type is present the solver uses the adapted FDTD method discussed on section (3.1.2). To preserve memory the values of $\sigma_{x,y,z}$ and $\mathcal{K}_{x,y,z}$ are only calculated when needed

and destroyed after each integration step. This process slows down the integration speed but favours the execution of the solver on less resourceful hardware. After each integration step, the **EMF.push** method is also responsible for applying the MABCs using the values obtained in the initialization routine and for incrementing the internal iterations counter.

An important note regarding this solver is that when there are external field sources, the total electromagnetic field inside the simulation mesh corresponds to the superposition of field obtained using the FDTD solver (which may not be null due to the presence of a polarization as in the case of the Bloch equations) and the field imposed by the external source. For this reason, the instances of the **EMF** class provide the methods **EMF.Efield**, **EMF.Bfield**, **EMF.Dfield**, **EMF.Hfield** and **EMF.EfieldPrev**, **EMF.DfieldPrev**, **EMF.BfieldPrev**, **EMF.HfieldPrev** that return a **Field** object containing the total components of the fields on the current or previous iterations, respectively.

Finally, this class is also equipped with a method called **EMF.saveToAF** that saves the components of the electromagnetic field to local files using the **af::saveArray** function provided by the ArrayFire API. The arguments of this method are a **string** with the path to the folder where the user wants to save the data, a vector of **booleans** specifying which components of the field are be saved, a **boolean** value that indicates if the fields are saved on the SI unit system, a **af::dtype** variable that represents the floating point precision of the saved arrays and a **FieldType** variable that specifies the sampling position on the Yee cells of the saved quantities.

4.2.10 The solver of the Bloch equations

Our solver of the Bloch equations is embedded in the **Bloch** class. This class is responsible for simulating not only the dynamics of the elements of the density matrix but also the evolution of the orientation of the transition dipole moments and the corresponding macroscopic polarization. The class constructor receives the following list of arguments that define the simulation:

- Mesh - Instance of the **YeeMesh** class;
- N - Definition of the number of available energy levels;
- η - Particle density, assumed uniform in the simulation mesh;
- α - Constant that defines the precession speed of the atomic transitions dipoles;

- β - Constant that defines the rotation speed of the atomic transitions dipoles;
- μ - Vector of vectors that containing the values of μ_{ij} ;
- Γ - Vector of vectors that containing the values of Γ_{ij} ;
- ω - Vector of vectors that containing the values of ω_{ij} ;
- RenormalizationInterval - Number of iteration between the forced renormalization of the transition dipole versor. Default value is 1.

Note that for a number of energy levels N , the μ , Γ and ω arguments must have N^2 elements. This design choice permits to use the same solver for any arbitrary number of energy levels but requires that the user specifies all the interaction terms between the density matrix elements, even if they are null. The organization of the elements of the matrices is also arbitrary and therefore the user must be careful upon the creation of the provided arguments to ensure that there is a coherent definition. Using the provided arguments, the constructor verifies that the matrices have the appropriate dimensions and creates an instance of the **Matrix** class with each vector of vectors. The class constructor also creates two **TensorField** objects and a **Field**, which refer to the Ω tensor containing the local values of Ω_{ij} , the state of the local density matrix ρ and the versors of the transition dipole moment orientations, respectively.

Like the **EMF** class, the **Bloch.Initialize** initializes the simulation of the Bloch equations. This method receives a vector of vectors with $N^2 \times 2$ elements that define the initial state of the real and imaginary parts of the elements of the density matrix and a variable of type **DipoleDir** that belongs to an *enum class* and specifies the initial orientation of the transition dipole versors. For simplicity, it is assumed that both the initial state and dipole orientation are uniform in the simulation mesh.

Using the **Bloch.push** method it is possible to evolve both the state of the density matrix and the orientation of the transition dipole versors. The arguments of this function are two **Field** objects containing the values of E^n and E^{n+1} and a **boolean** variable that for a **true** value imposes that the solver uses the RK4 method instead of the RK2 method. For both Runge-Kutta methods, the solver uses equation (3.56) to interpolate the necessary values of the electric field and the functions **Bloch.dipoleVesorEquationSystem** and **Bloch.rhoEquationSystem** to evolve the state of the system. These functions describe equations (2.49) and (2.50) and the latter also uses an auxiliary method called

Bloch.updateDynamicMatrix to update the **TensorField** Ω ; The **Bloch.push** method is also responsible for renormalizing the dipole orientation versors using the function **normalizeVector**. This renormalization is relevant due to the fact that non-optimal integration steps may introduce small errors on the magnitude of the versors that propagate to the remaining integration steps.

The **Bloch** class objects also calculates the macroscopic polarization of the optical medium and the corresponding transition dipole force. However, these quantities are not relevant for the dynamics of density matrix nor the dipole orientation versors and therefore they are not kept in memory permanently inside the **Bloch** class instances. Instead, the class provides the **EMF.calculatePolarization** and **EMF.dipoleForce** methods that return a **Field** containing the corresponding quantities, according to equations (2.51) and (2.65), respectively.

Finally, it is possible to save the state of the density matrix and the macroscopic polarization to local files using the **Bloch.saveToAF** and **Bloch.savePolToAF** methods respectively, which are similar to **EMF.saveToAF**. The arguments of the first method are a **string** with the path to the folder where the user wants to save the data, two vectors of **booleans** that specify which real and imaginary parts of the density matrix elements are saved and a **af::dtype** variable that represents the floating point precision of the saved arrays. The second method simply receives the **string** with the path to the folder where the user wants to save the data and a **boolean** value that indicates whether the saved array is converted to the SI unit system or not.

4.2.11 Launching a simulation

We now review the process for deploying a simulation of the Maxwell-Bloch equations using the presented classes and libraries. Figure (4.6) represents the general scheme that users should follow to develop the main script that launches the simulation. For example, to study the interaction of a field with a two-level atomic gas, the user must only specify the units system appropriate for the simulation, the parameters of the **YeeMesh** and the physical context of the problem, which in this case refers to the desired electromagnetic field sources, either defined via an external source or by indication of the initial state, the μ , ω and Γ vectors of vectors, the constants α and β that govern the dynamics of the transition dipoles orientation vedor, the particle density η and the initial state of the elements

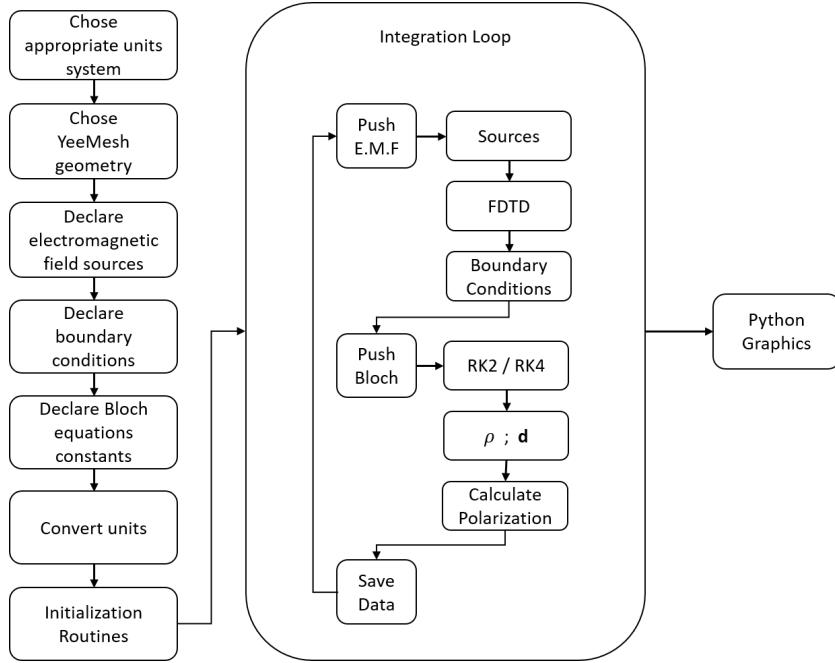


FIGURE 4.6: Scheme of the solver implementation layout and execution flow

of the density matrix. An example of the main script used to launch the simulation of a Gaussian pulse interacting with a two-level atomic gas can be found on appendix A.

4.3 Performance analysis

In this section, we present the analysis of the computational performance of our solver. The main intention is to study both the general performance of our software and the advantages of the GPU computing technologies to simulate the interaction of electromagnetic field with atomic gases. Besides testing the performance benefits of the GPU implementation against the single-thread CPU equivalent, we also explore the advantages of GPU computing for the simulation of Maxwell-Bloch equations on three representative computers in order to compare the advantages of using this technology on distinct hardware budgets ranging from a low-end laptop to an high-end desktop. The specifications of these computers are:

- **Laptop:** Intel core i5 6200U, NVIDIA GT 920M, 8GB DDR3
- **Common desktop:** Intel core i5 4590, NVIDIA GTX 970, 8GB DDR3
- **High-end desktop:** Intel core i7 4930k, NVIDIA GTX Titan (Kepler), 64GB DDR3

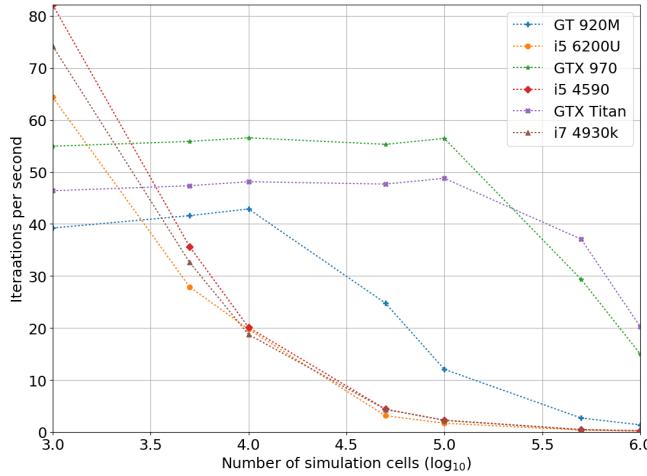


FIGURE 4.7: Comparison of iterations per second on different CPUs and GPUs. Notice that for small numbers of iteration cells the CPUs have a clear advantage over the GPUs but for large numbers os cells we observe the opposite result.

where all the computers run *Windows 10* with *CUDA Toolkit 8.0* [34] and *ArrayFire 3.4*. The code is compiled using *Microsoft Build Tools* and *NVIDIA CUDA Compiler*.

We compare the computation performance of our solver by running a benchmark test on our hardware setups that consists in measuring the elapsed time between consecutive integrations using the *chrono* library [41] and averaging the value of a specific number of iterations.

Analysing figure (4.7) it is possible to conclude that for a small number of cells there is no benefit in using GPUs since the CPU cores are significantly faster. However, as the number of cells in the simulation box grows, the CPU performance decays rapidly with the weakest GPU outperforming the best CPU. This results also show that when maximum core occupancy is achieved the communication between CPU and GPU affects the performance of the latter due to the latency involved in data communication. It is also important to note that even though the GTX Titan has a larger number of compute cores than the GTX 970, the latter is slightly faster before the maximum core occupancy is reached since it has a faster clock frequency (1050 MHz vs. 837 MHz).

It is also interesting to directly compare the performances of GPU and CPU computing on different hardware scenarios. Figure (4.8) presents the speedups obtained from using the GPU rather than the CPU of our computers to simulate the Maxwell-Bloch equation. These results show that even for our low-end computer the GPU is almost ten times faster

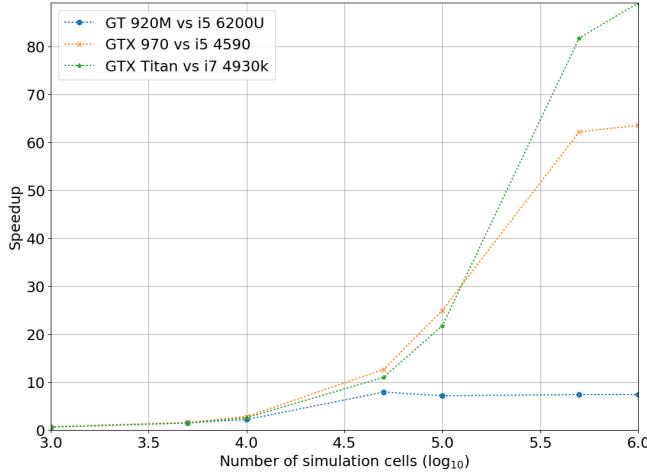


FIGURE 4.8: Speedup comparison on different hardware configurations. Notice that in all configurations the GPUs provide a good speedup over the CPUs and that for high-end computers the speedup is more significant.

than the CPU on simulation meshes with a large number of Yee cells and that on the high-end computer the GPU reveals a performance over ninety times faster than the CPU, reassuring that our solver explores the benefits of this technology.

4.4 Numerical error analysis

To test the numerical stability of our solver, we study the evolution of the trace of the density matrix on a two-level atomic gas inside a one-dimensional simulation mesh. This test is of special relevance since one of the physical restrictions of the density matrix is that the diagonal terms correspond to probabilities of occupations of each atomic level and since no atoms are removed from the system during the simulation the sum of the probabilities corresponding to the trace of ρ must be constant and equal to 1, as presented in equation (2.15). We test both RK2 and RK4 methods with different integration steps defined as fractions of Δ_x/v , where v is the speed of light in vacuum. Observing the results in figure (4.9) it is possible to conclude that the error in the trace is close to the machine epsilon and that it does not tend to increase over time. These results also show that although the RK4 is slightly better than the RK2 for smaller integration steps, the errors are both dominated by the computer unit roundoff and therefore it is more beneficial in computational terms to use RK2 due to the lower memory usage and a smaller number of floating point operations.

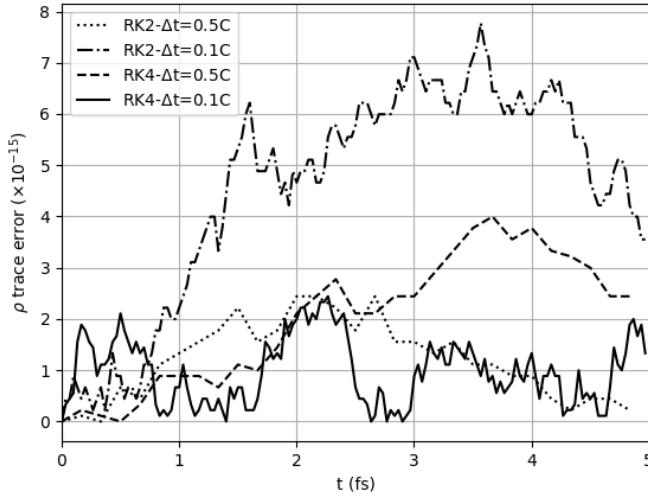


FIGURE 4.9: Comparison of numerical stability of the trace of the density matrix using different Runge-Kutta methods. C represents $\frac{\Delta x}{v}$, where v is the speed of light in vacuum.
This test shows that the numerical problem is well-conditioned.

4.5 Boundary conditions performance

The physical relevance of the simulation results is not only affected by the numerical errors, but also by the performance of the boundary conditions, in particular of the case of absorbing boundary conditions. The periodic boundary conditions are easy to implement with the FDTD method due to the periodic nature of the arrays and are well conditioned.

On the other hand, the absorbing boundary conditions in the solver employed two types of implementations, namely the MABCs and the UPMLs. The MABCs method was developed in the scope of this work, it is necessary to study the behaviour of this absorbing boundary type and understand how the parameters C and L present in equation (3.22) affect the attenuation coefficient of the electromagnetic field which is defined as

$$\log_{10} \left(\frac{|E(\mathbf{r}, t_f)|_{max}}{|E(\mathbf{r}, 0)|_{max}} \right) \quad (4.1)$$

where $E(\mathbf{r}, t_f)$ is the amplitude of the residual electric field inside the mesh after the propagation through the border. Therefore, this attenuation factor represents the difference in orders of magnitude between the incident and reflected fields. The simulation setup for this test consists in the propagation of a linearly polarized Gaussian pulse in a one-dimensional mesh with 1000 Yee cells and with borders of different thickness's and C factors. For simplicity, $m = 3$ was used for all test cases. Observing figure (4.10) it is

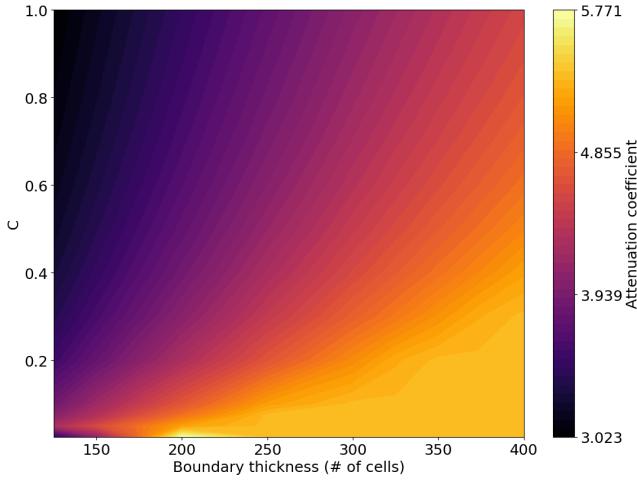


FIGURE 4.10: Study of the attenuation coefficient of the MABCs as a function of the parameters L and C . This study reveals that for normal incidence the optimal range is centred at $C = 0.1$ and $L = 200$ cells.

possible to conclude that for very thin borders there is minimal absorption. The optimal cases occurs for a thickness of $L=200$ cells and $C=0.1$ where the attenuation coefficient is maximum. Note that the border thickness in our code was defined as a fraction of the axis length and therefore the optimal value of L represents $\frac{L_x}{5}$.

Using the parameters obtained from the previous study, we now characterize the attenuation factors of the MABCs and the UPMLs for different angles of incidence. This test consists on the propagation of a linearly polarized Gaussian pulse through a two-dimensional simulation mesh where the FWHM of the pulse is equal on both axis and the angle of incidence corresponds to the angle between the wave vector of the pulse and normal direction to the plane of incidence, as presented on figure (4.11). Figure (4.12) contains the results of this characterization and it is possible to observe that the MABCs in this have a slightly worse performance than in the one-dimensional case case which may be a consequence of the dispersion of the field. It is also important to note that the numerical parameters used for the UPMLs correspond to the optimal values defined by Taflove *et al.* [28] but our results are not consistent with the ones presented by the authors since it is expected that for this type of boundary condition to present an attenuation coefficients greater than ten. For this reason, we have thoroughly reviewed our implementation of this boundary type and found no evident sources of error. However, the optimal implementation of boundary conditions is beyond the scope of this work, but this issue should

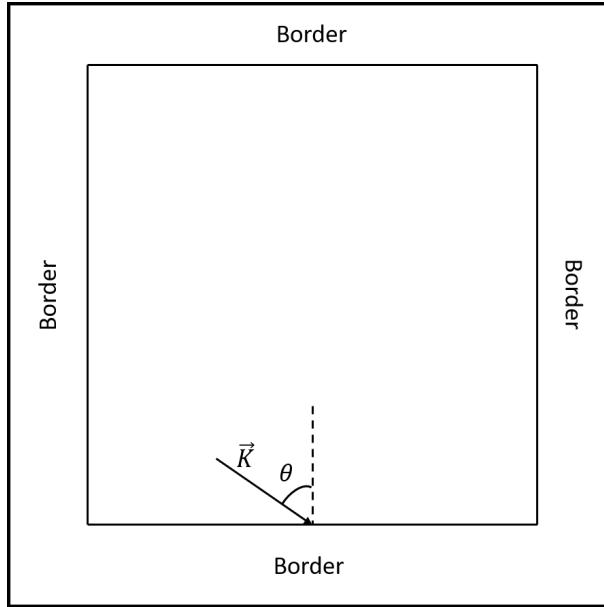


FIGURE 4.11: Illustration of the angle of incidence used for the boundary characterization test.

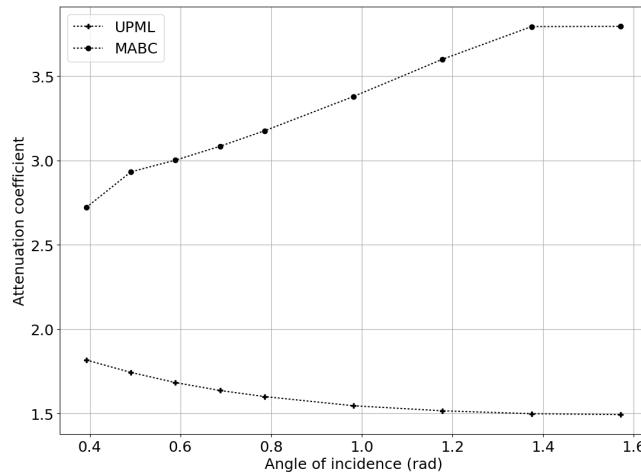


FIGURE 4.12: Study of the attenuation coefficient of UPMLs and MABCs. Notice that the performance of the MABCs is not as good as in the parameter characterization test, which may be a consequence of the dispersion of the field.

be revisited in the future, nonetheless.

Finally, it is also important to study the impact caused by the usage of the different absorbing boundary conditions on the overall computational performance of the solver. Analysing figure (4.13) it is possible to conclude that using our implementation of the UPML method drastically reduces the performance of the solver of the electromagnetic

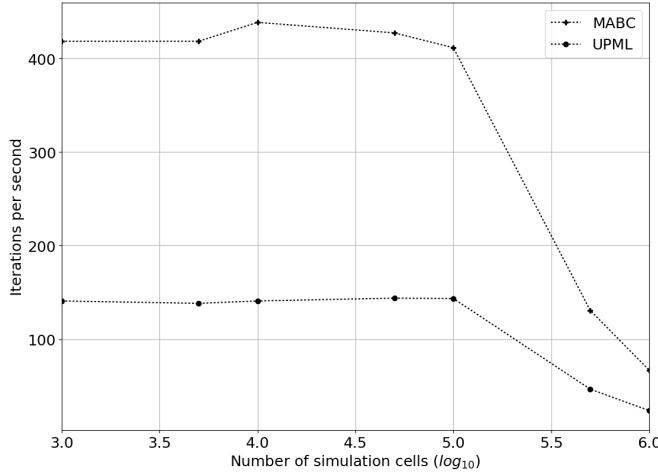


FIGURE 4.13: Comparison of the impact of using MABCs and UPMLs on the electromagnetic field solver running on the Nvidia GTX 970. Notice that the MABCs have an overall better iteration performance than our implementation of the UPML method.

field when compared to the MABC method. This effect is caused not only by the nature of the UPML method, which increases the number of floating point operations performed by the FDTD method but also, by our choice to calculate the values of $\sigma_{x,y,z}$ and $\mathcal{K}_{x,y,z}$ only when needed to preserve memory and therefore also increasing the total number of operations of the solver. However, we still favour our design of MABCs as it allows the execution of the code on less resourceful hardware.

4.6 Conclusions

This chapter described the technical component behind the development of this simulation software and discussed the advantages and disadvantages of the GPGPU computing paradigm, as well as, the design of our solver. Furthermore, we explained the general architecture of the solver and presented an in-depth analysis of the developed classes and libraries that permit the modular behaviour of the software. Additionally, this chapter also serves as a manual for future upgrades of the software.

Moreover, we compared the performance of our solver using GPGPU computing with the equivalent CPU solution in multiple hardware environments and concluded that this software benefits from the parallelized execution in all the tested scenarios, with our high-end GPU exhibiting a speedup of ninety over the respective CPU. We also tested the

numerical accuracy of the solver and determined that the integration errors are of the order of the machine epsilon, implying that the problem is numerically well-conditioned.

In the following chapter, we apply our solver to the study of multiple physical scenarios that serve as examples of the functionality of our software and show how it may be used to enhance the current state of the art of the simulations regarding the interaction between light and matter.

Chapter 5

Physical test cases

This chapter provides four examples of the application of the simulation software developed in this thesis, each dedicated to the demonstration of a specific feature and capability of the solver.

The first example studies the interaction of a Gaussian pulse with a three-level atomic gas in a one-dimensional mesh and aims to demonstrate both the features of our physical model that set it apart from the ones found in the literature and the capability of our solver to simulate transient dynamics. In the second example, we impose an external field over a two-level atomic gas in a thin slab geometry and show that the solver is capable of simulating the absorption of the electric field in atomic gas media characterized by a refractive index with an imaginary part. Furthermore, in the third example, we study the propagation of a Gaussian pulse in a two-level atomic gas, displaying the effectiveness of the solver to simulate three-dimensional systems and the interaction of electromagnetic fields with dispersive optical media. Additionally, in the fourth example, we couple our solver with a many body solver of the collisionless Boltzmann equation to demonstrate the potential of the modular behaviour of our software and how it may pave the way for the simulation of the Maxwell-Bloch equations coupled with transport phenomena in atomic gases.

Obviously, the possible applications of this solver are not limited by these case studies. Instead, they provide a small example of what can be done. Also, we stress that the goal of this chapter is to show the features and functionalities of our software, rather than explore all the physical implications of the tests.

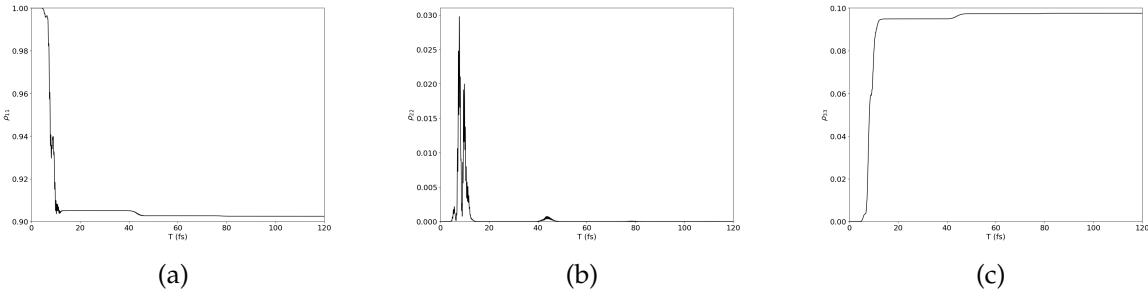


FIGURE 5.1: Evolution of the states of a three-level atomic system in a one-dimensional simulation. (a), (b), (c) represent the evolution of the ground, excited and metastable states respectively. Observe that the atomic system evolves towards a dark state.

5.1 Three-level atomic gas in a one-dimensional mesh

To ensure that our software is capable of solving the dynamics of multi-level atomic systems and also that the obtained results are in accordance with the theoretical models, we created a physical simulation that consists in a three-level atomic gas initially in the ground state and inside a one-dimensional mesh with $L_x = 10\mu\text{m}$ interacting with linearly polarized laser pulse whose initial state is defined as

$$\mathbf{E}(x, 0) = E_0 [\sin(k_1 x) + \sin(k_2 x)] \exp^{-\left(\frac{x-x_0}{2\sigma_x}\right)^2} \hat{\mathbf{z}}, \quad (5.1)$$

$$\mathbf{B}(x, 0) = -\frac{E_0}{c} [\sin(k_1 x) + \sin(k_2 x)] \exp^{-\left(\frac{x-x_0}{2\sigma_x}\right)^2} \hat{\mathbf{y}}, \quad (5.2)$$

where $E_0 = 1 \times 10^{10}\text{V} \cdot \text{m}^{-1}$ represents the amplitude of the electric field, $k_1 = c/\omega_{12}$ and $k_2 = c/\omega_{32}$ are the wave vectors of the plane waves resonant with the atomic transitions and $L_x = 5\mu\text{m}$ and $\sigma_x = 0.5\mu\text{m}$ are the initial displacement and the FWHM of the Gaussian modulation, respectively. Note that for this test case, the vacuum wavelengths associated with the atomic transitions are $\lambda_{12} = 500\text{nm}$ and $\lambda_{32} = 700\text{nm}$, the magnitudes of the transition dipoles are $\mu_{12} = \mu_{32} = 1 \times 10^{-29}\text{C} \cdot \text{m}$ and the effective decay rates are $\Gamma_{12} = 1 \times 10^{15}\text{Hz}$ and $\Gamma_{32} = 2 \times 10^{15}\text{Hz}$.

Figure (5.1) depicts the evolution of the population of the ground, excited and metastable states for an atomic ensemble situated at $x = 7.5\mu\text{m}$ upon multiple passages of the laser pulse. The results allow to conclude that density operator evolves towards a solution where there is no population in the excited state, which represents the occupation of a *dark state* as discussed in section (2.3.4) of chapter two.

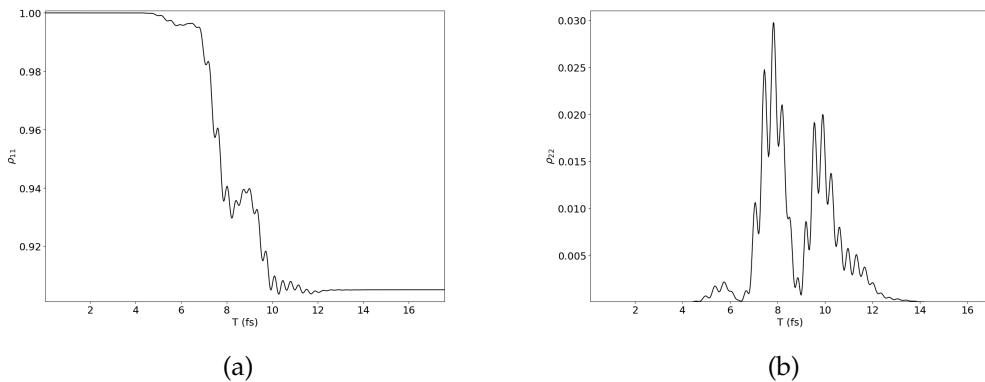


FIGURE 5.2: Observation of transient response in a three-level atomic system. (a) and (b) represent the evolution of the ground and excited states respectively. Notice that this transient regime presents the combination of multiple oscillatory effects, namely the Rabi frequency and the effective decay rates.

Further analysis of figures (5.1) and (5.2) show that in the initial moments of the simulation the population of the ground and excited state oscillate rapidly, corresponding to a combination of Rabi oscillations with other effects present in the simulation, including the oscillation of the electric field, the natural atomic state transitions and the effective decays that are only observable via simulation of the transient dynamics.

5.2 Two-level atomic gas in a thin slab

To further study the capabilities of our solver, we created a thin slab of optical media represented by a three dimensional mesh with $L_x = 1\mu m$, $L_y = 10\mu m$ and $L_z = 100nm$ and simulated the interaction of a two-level atomic gas initially in the ground state, with an electromagnetic field imposed by an external source and defined as

$$\mathbf{E}(x, y, z, t) = E_0 \sin(\omega_{12}t - kx) \exp^{-\left(\frac{y-y_0}{\sigma_y}\right)^2} \hat{\mathbf{z}}, \quad (5.3)$$

$$\mathbf{B}(x, y, z, t) = -\frac{E_0}{c} \sin(\omega_{12}t - kx) \exp^{-\left(\frac{y-y_0}{\sigma_y}\right)^2} \hat{\mathbf{y}}, \quad (5.4)$$

where $\sigma_y = 1\mu\text{m}$ and ω_{12} represents the angular frequency associated with the transition between the atomic energy levels characterized, in this case, by a wavelength of $\lambda_0 = 500\text{nm}$. Note that in the context of this test the atomic density is constant in time and uniform in the simulation mesh.

As discussed in chapters three and four, this resonant electromagnetic field is not simulated by the FDTD method. However, it is expected that upon the interaction with the

atomic gas, it affects the dynamics of the local states of optical media and produces a macroscopic polarization \mathbf{P} , which behaves as a field source. This results in a secondary electric field propagated by our solver. Figure (5.3) presents a view of the results of this simulation where the top, centre and bottom images, from left to right, correspond to the evolution of the magnitude of the electric field, the magnitude of the macroscopic polarization and the local population of the ground state, respectively. These results show that as the external field propagates through the thin slab, it drives the atoms between the ground state and the excited state, resulting in Rabi oscillations which occur alongside the transitions caused by the decay mechanisms and that impose a local polarization in the atomic system. It is also possible to observe that the amplitude of the total electric field decreases, which is characteristic of a optical medium with an imaginary refractive index and a consequence of the phase difference between the external electric field and the polarization of the atoms. This polarization is also delayed in respect to the electric field which is a consequence of the non-zero response time of the atoms.

5.3 Gaussian pulse propagation in a three-dimensional mesh

We also tested our solver in a different physical scenario that consisted on the propagation of a laser pulse through a cubic grid ($L = 1\mu\text{m}$) with a two-level atomic gas initially in the ground state. Figure (5.4 (a)) shows the initial state of the simulation. In the lower part is represented the amplitude of the total electric field, including the contribution from the field emitted by the atoms (which for $t = 0$ is null). On the middle part is represented the polarization field generated by the atoms, which for $t = 0$ is null since initially the atoms are all in the ground state. Finally, in the back plane of figure (5.4 (a)) is depicted the local fraction of atoms that occupy in the ground state. Figure 5.4 (b) illustrates the state of the system at end of the simulation. First, we notice that part of the atoms have evolved into the excited state, leaving a hole in the distribution of atoms in the ground state. This effect is the equivalent of spatial hole burning observed in lasers [42]. It is interesting to notice that the population hole is lagging behind the maximum of the pulse, again denoting the delay in the response time of the atoms to the field. This effect is also observed in the polarization field, which is delayed relatively to the maximum of the pulse. The influence of the polarization is also noticed in the total electric field where it is possible to observe an increase of the duration, which indicates a decrease of the group velocity of the pulse as a result of the dispersive effective refractive index, characteristic of an atomic gas.

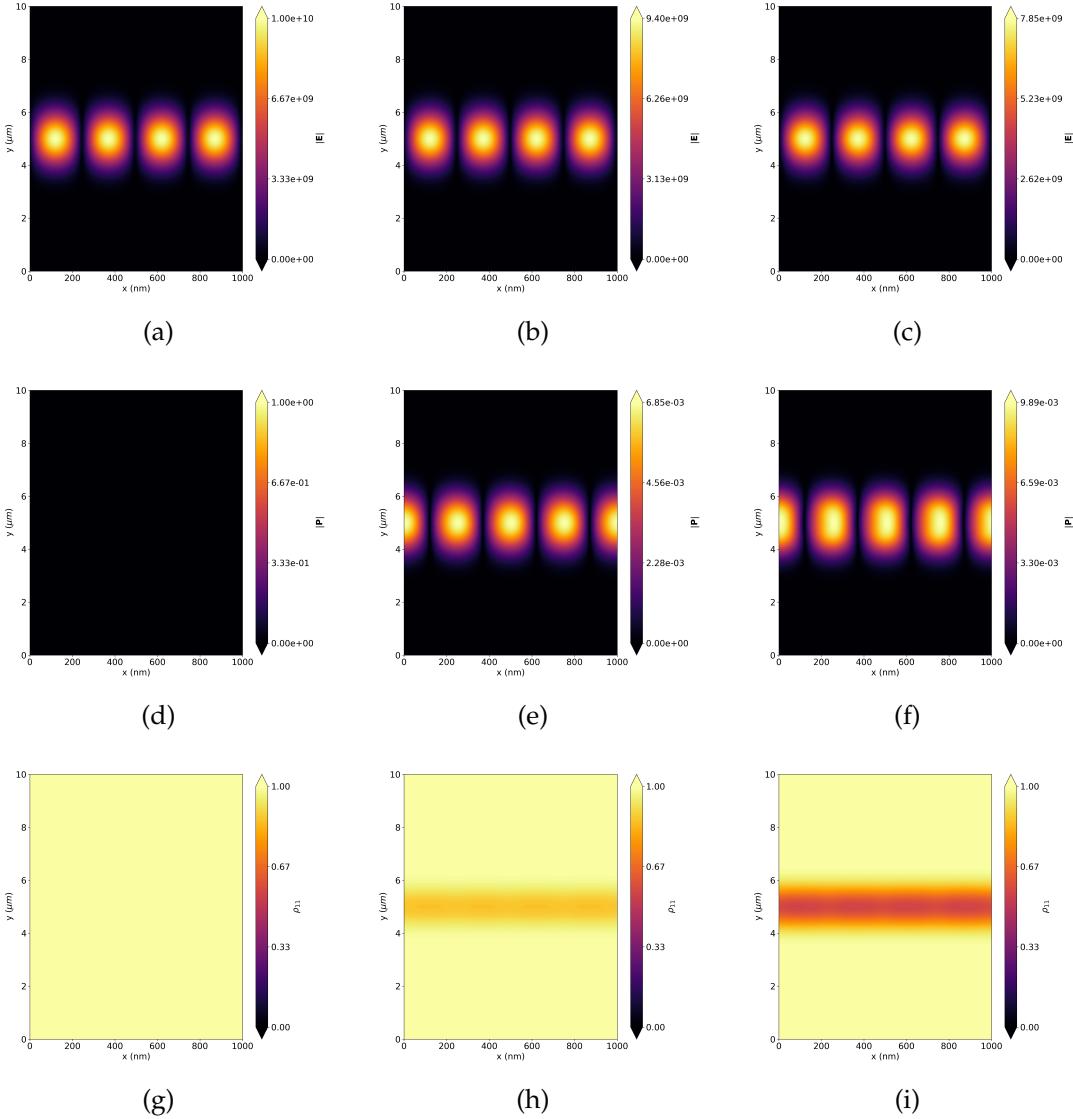


FIGURE 5.3: Cut-plane of the system normal to the Z axis at the beginning, middle and end of the simulation. (a), (b) and (c) represents the evolution of the magnitude of the electric field; (d), (e) and (f) correspond to the evolution of the magnitude of the polarization; (g), (h) and (i) contain the evolution of ρ_{00} . The values are in the SI units system. Observe that the atomic polarization is delayed in respect to the electric field and that the magnitude of the latter decreases over time.

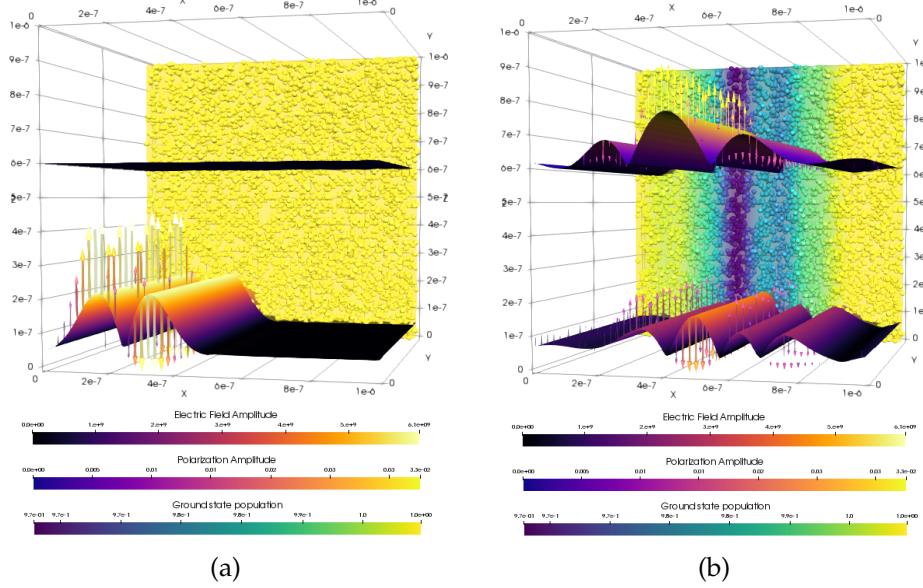


FIGURE 5.4: Evolution of pulse propagating through a two-level atomic gas. The lower part represents the amplitude of the electric field, the middle part illustrates the amplitude of the polarization field and the back plane contains the information regarding the local population of the ground state of the atoms. (a) and (b) represent the state of the system at $t = 0$ and at the end of the simulation, respectively. The values are in the SI units system. Notice that there is an increase in the duration of the pulse indicating a decrease of its group velocity as a result of the dispersive effective refractive index, characteristic of an atomic gas.

5.4 Coupling of the solver with a PIC code

To test the operation of the modular aspects of our solver and to assert the possibility of simulating new physical effects regarding the interaction of electromagnetic fields with atomic gases, we coupled our code to a module containing a *Particle in Cell* (PIC) code [43], which was developed by Miguel Gomes and that is responsible for simulating the kinematic properties of the atoms in the optical media. The acute details regarding the development of the PIC code and the underlying physical model are beyond the scope of this work and therefore we limit the discussion of this simulation to a general overview of the problem and the relevant aspects of the interaction.

The PIC method does not solve the kinematic properties of the individual atoms but considers instead the statistical properties of a collection of superparticles, each corresponding to a local population of atoms forming a cloud that moves without deformation or collisions throughout space under the action of an external force, and relies on a numerical approach to solve the Boltzmann equation for collisionless gas [44]

$$\dot{f} + \frac{1}{m} (\mathbf{p} \cdot \nabla_{\mathbf{r}}) f + (\mathbf{F} \cdot \nabla_{\mathbf{p}}) f = 0, \quad (5.5)$$

where \mathbf{F} represents an external force, which in this case is the transition dipole force

$$\mathbf{F}(\mathbf{r}, t) = -\nabla U(\mathbf{r}, t), \quad (5.6)$$

\mathbf{r} and \mathbf{p} are the positions and momenta of the atoms respectively and m is the mass of the atoms. The parameter f corresponds to a probability density function defined so that

$$N = f(\mathbf{r}, \mathbf{p}, t) d^3 \mathbf{r} d^3 \mathbf{p}, \quad (5.7)$$

represents the total number of particles per unit volume of the phase space of the system.

Note that these systems consist, in practice, in a dipole gas but unlike the cases typically described in the literature [45], the dipole interaction between the atoms is not modelled as a direct interaction but rather is intermediated by the electromagnetic field, thus rendering the model with an increased level of realism.

Our simulation scenario consisted on a two-dimensional mesh containing particles modelled as two-level atoms initially in the ground state, distributed in the grid according to a uniform random distribution and where the initial state of the electromagnetic field corresponded to a stationary wave defined as

$$\mathbf{E}(\mathbf{r}, 0) = E_0 \left[\sin \left(\frac{2\pi}{\omega} x + \frac{\pi}{2} \right) + \sin \left(\frac{2\pi}{\omega} x - \frac{\pi}{2} \right) \right] \hat{\mathbf{z}} \quad (5.8)$$

$$\mathbf{B}(\mathbf{r}, 0) = \frac{E_0}{c} \left[\sin \left(\frac{2\pi}{\omega} x - \frac{\pi}{2} \right) - \sin \left(\frac{2\pi}{\omega} x + \frac{\pi}{2} \right) \right] \hat{\mathbf{y}}, \quad (5.9)$$

where ω is the field frequency.

Figure (5.5) depicts the results obtained using $\omega_{12} = 0.5\omega$, which implies a blue detuning between the electromagnetic field and the atomic transition, while on the other hand, figure (5.6) represents the evolution of the system using $\omega_{12} = 1.5\omega$ that corresponds to a red detuning. Besides being possible to observe that the polarization is delayed with respect to the electric field, as in the previous cases, it is also possible to note that for a blue detuning the density of the atoms evolves towards a state where there is maximum local atomic density at the nodes of the electric field while for a red detuning the opposite occurs, with the maximum local atomic density tending to align with the antinodes of the field. This behaviour was previously observed in the work of Schmittberger *et al* [46].

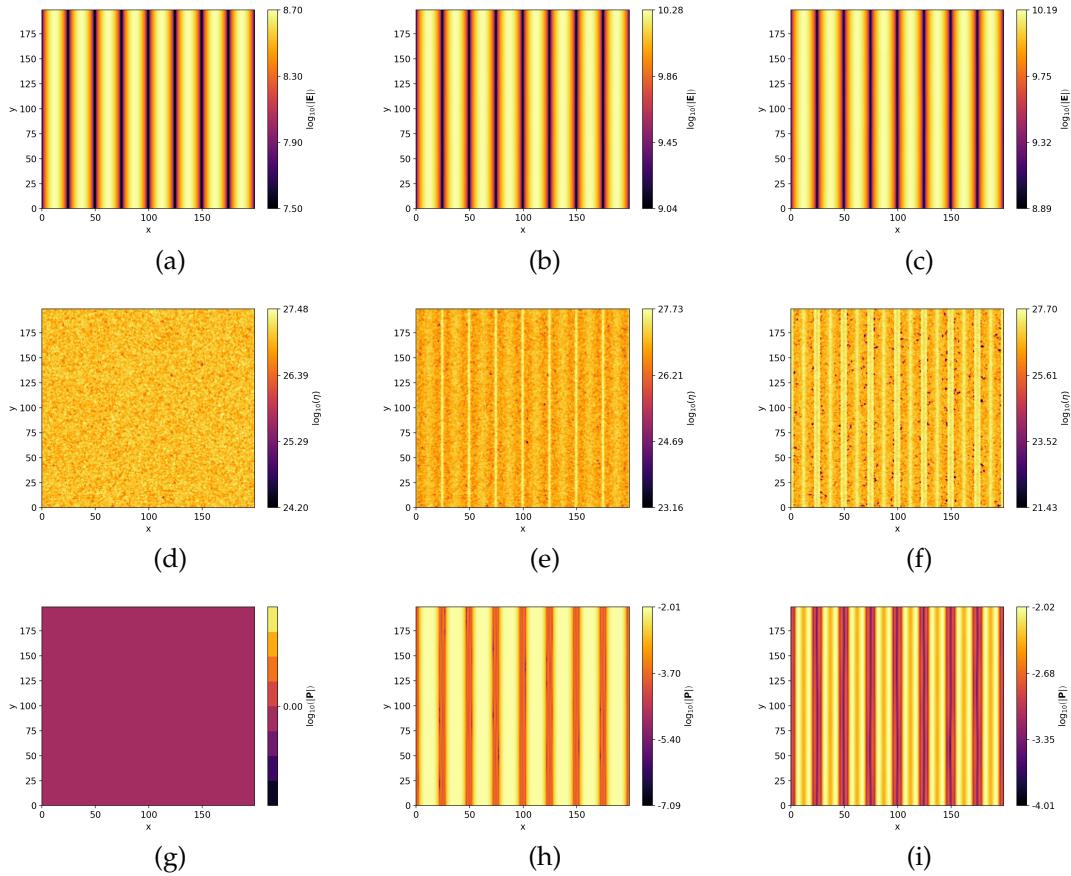


FIGURE 5.5: Evolution of the polarization and local density of an atomic gas interacting with a blue detuned stationary wave using $\omega_{12} = 0.5\omega$. (a), (b) and (c) represent the magnitude of the electric field at the beginning, middle and end of the simulation. (d), (e) and (f) contain the evolution of the local atomic density. (g), (h) and (i) depict the evolution of the magnitude of the local polarization of the atomic gas. The values are in the SI units system. Note that there is an increase of the atomic density at the nodes of the electric field

This simulation demonstrates that using our solver and its modular features it becomes possible to easily integrate new solvers into our software that compute the dynamics of the desired physical phenomena regarding the interaction of electromagnetic fields with atomic gases.

5.5 Conclusions

In this chapter, we demonstrated the functionality of our solver by applying it to the simulation of simple but yet relevant physical scenarios.

The first example consisted in a Gaussian pulse interacting with a three-level atomic gas in a one-dimensional mesh, where it was possible to observe a combination of various

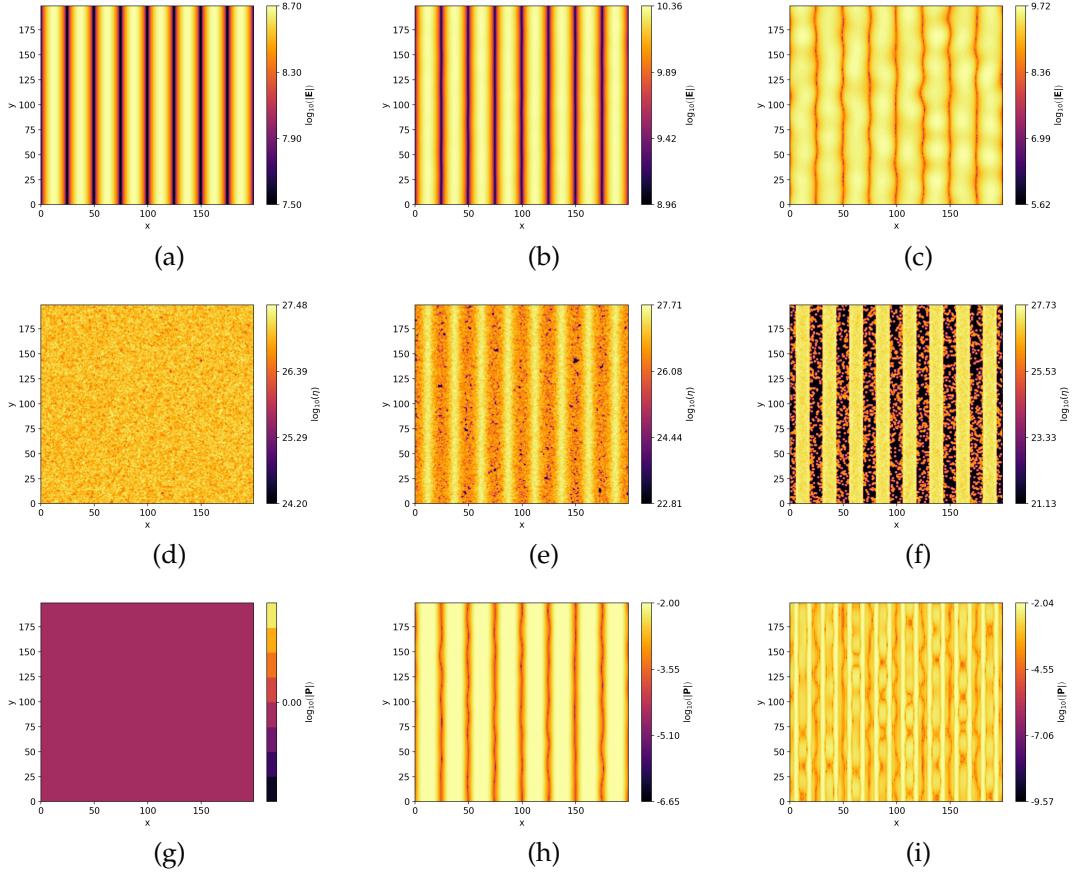


FIGURE 5.6: Evolution of the polarization and local density of an atomic gas interacting with a red detuned stationary wave using $\omega_{12} = 1.5\omega$. (a), (b) and (c) represent the magnitude of the electric field at the beginning, middle and end of the simulation. (d), (e) and (f) contain the evolution of the local atomic density. (g), (h) and (i) depict the evolution of the magnitude of the local polarization of the atomic gas. The values are in the SI units system. Note that there is an increase of the atomic density at the antinodes of the electric field

oscillatory phenomena, namely the Rabi and the effective decays and the evolution of the system towards a dark state, similar to the situation discussed in section (2.3.4) of chapter two. This simulation demonstrated that our solver is capable of studying new features and effects that go beyond the RWA.

The second example illustrated the interaction of an external field with a two-level atomic gas, initially in the ground state, inside a thin slab and it was possible to observe the evolution of the local macroscopic polarization that is delayed with respect to the propagation of the field. This test also shows that despite the physical model does not include a quantum description of photon absorption, the solver is capable of simulating absorption in media characterized by a refractive index with an imaginary part such as two-level atomic gases. In this case, the absorption is obtained as an attenuation of the

total electric field in the simulation mesh due to a phase difference between the external field and the field re-emitted by the atomic polarization, which interfere destructively.

In the third example, we demonstrated the capability of our solver to simulate three-dimensional systems. It consisted in the propagation of a pulse in a two-level atomic gas which revealed an increase of the pulse duration, which indicates a decrease in its group velocity. This result also shows that our solver is capable of simulating the propagation of short pulses through dispersive media.

Finally, in the last example, we demonstrated the potential of the modular character of this code by combining our Maxwell-Bloch solver with a PIC module responsible for simulating the kinematic properties of the atomic gas. With this test we were able to study the interplay between the field, the quantum states of the atoms and the transport phenomena of the gas. The results of this simulation show that depending on the detuning the atomic density evolves towards the nodes or the antinodes of the electric field.

This short list of examples represents but a small fraction of the type of systems, effects and problems that can be studied with this solver or by combining it with others that explore different aspects of the interaction between light and atomic gases.

Chapter 6

Concluding remarks and future work

This work presents the development and test of a modular solver for the three-dimensional Maxwell-Bloch equations that also behaves as a software tool for the simulation of physical phenomena regarding the interaction of electromagnetic fields and atomic gases using C++ and GPGPU computing.

As discussed in chapter two, we began by investigating the physical models that govern the propagation of electromagnetic fields through optical media and their interaction with the internal degrees of freedom of atomic gases and opted for a semi-classical description of the problem, which treats the field classically using the Maxwell equations, while describing the atoms with a quantum model using a finite number of energy levels. This led us to develop a model that looks very similar to those found in specialised literature but with some distinctive features and augments. First, our physical model extends the conventional Maxwell-Bloch equations to three dimensions, a feature not usually considered given the difficulty in solving these equations with higher dimensions. This required to include an empirical model to compute the evolution of the spatial orientation of the transition dipole moment of the atoms and of the local polarization vector. Second, our model disregards the RWA and thus is able to evaluate in detail the transient dynamics for short and ultrashort times and phenomena.

In chapter three, we then proceeded to analyse some of the available numerical methods for the simulation of our model and concluded that the best-suited options for our software were the FDTD method and the Runge-Kutta family methods to solve the Maxwell and the Bloch equations, respectively. In this discussion, we also approached the methods to introduce fields sources in the simulation and the boundary conditions problem,

for which we reviewed the concepts of UPMLs and introduced a method developed during this thesis and denominated by MABCs.

In chapter four, we presented the design of our solver and discussed the advantages and disadvantages of the GPGPU computing paradigm alongside the available APIs that simplify the development of the code. Furthermore, we explained the general architecture of our solver and presented an in-depth analysis of the developed classes and libraries that permit the modular behaviour of the software. This analysis also serves as a manual for future upgrades of the solver.

Additionally, we compared the performance of our solver using GPGPU computing with the equivalent CPU solution in multiple hardware environments and concluded that this software benefits from the parallelized execution in all the tested scenarios, with our high-end GPU exhibiting a speedup of ninety over the respective CPU. We also tested the numerical accuracy of the solver and determined that the integration errors are of the order of the machine epsilon. Although this is the largest chapter in this document, it only represents a fraction of the work involved in developing the solver, which comprises approximately nine thousand lines of code and accounts for more than eight months of dedicated programming and development time. A small part of the code can be found in appendix A that presents the script responsible for the launch of the simulation.

Finally, in chapter five, we tested our solver by simulating multiple physical scenarios. The first case consisted on a one-dimensional Gaussian pulse interacting with a three-level atomic gas where the system evolved towards a dark state, as expected. The second case corresponded to the interaction of an external field with a two-level atomic gas, initially in the ground state, inside a thin slab and it was possible to observe the transitions between the energy levels and the evolution of the local macroscopic polarization that, due to a phase difference, affected the total magnitude of the total electric field in the simulation mesh. The third case was similar, and consisted in the propagation of a Gaussian pulse in a three-dimensional mesh containing a two-level atomic gas. It demonstrated the interplay between the field and the atomic system, revealing an increase of the pulse duration and indicating a decrease in the group velocity. In the fourth and final test case, we set the interaction of a stationary wave with a two-level atomic gas and coupled our solver with a PIC module that simulated the effect of the dipole force on the local density of atoms. We were able to observe that for red detunings between the field and the atomic transition frequency the atoms moved toward the antinodes of the electric field while for a blue

detuning the atoms concentrated around the nodes of the electric field, as predicted by the analytical models [46].

In conclusion, the solver developed during this thesis introduces several contributions to the state of the art in this field. It not only fulfils the need for a fast simulation tool of the Maxwell-Bloch equations, but also shows that using GPGPU computing it is possible to incorporate features in the physical model that usually are left out, such as three-dimensional systems and transient dynamics. Also, this solver poses as a first step for the development of a modular and general software platform regarding the numerical study of light-matter interactions. However, the solver is not yet a full-featured software solution and, in the future, it should be upgraded to support multi GPU environments that due to the increased memory permit the simulation of larger physical scenarios, either via the creation of bigger simulation meshes or the inclusion of other solver modules. Another interesting addition to the solver would be the development of a graphical interface to deploy the simulations instead of requiring the users to write a simulation script. Finally, there is still room for optimizations of the code including reducing the memory consumption of the developed modules and increasing the UPML absorption performance.

Appendix A

Simulation launch example

The following code is an example of the main script developed to simulate the interaction of a Gaussian pulse with a two-level atomic gas in a one-dimensional mesh. Note that the *core.h* header file links to all the auxiliary classes and libraries developed in this thesis and whose implementation comprises approximately nine thousand lines of C++ code.

```
#include <core.h>

void main(){
    /// General simulation properties
    int saveInterval = 100;
    int plotInterval = 1;
    bool save = false;
    Units units = Units(UnitSystems::NATURAL_LH);
    std::string resultsDirectory;

    /// YeeMesh constants declaration
    int Nx = 1000;
    int Ny = 1;
    int Nz = 1;
    double dx = 1.0e-8;
    double dy = 1.0e-8;
    double dz = 1.0e-8;
    double dt = (dx / Units::si_c()) / 2.0;
    double Tmax = 3.0e-15;
```

```
/// Electro-magnetic field constants declaration
PropagationDir direction = PropagationDir::X_P;
EFieldPol EPol = EFieldPol::Z;
double R0 = Nx*dx / 2.0;
double waveNumber = 2.0 * PI /(50.0*dx);
double phase = 0.0;
double FWHM = Nx*dx / 10.0;
double intensity = 1.0e10;
std::vector<bool> fieldSaveSelector = {false,true,false,false};

/// Atomic gas constants delcaration
int levels = 2;
double mu12 = 1.0e-29;
double eta = (intensity / mu12) / 1.0e12;
double omega12 = Units::si_c() * waveNumber;
double gamma12 = (intensity*mu12/Units::si_hbar())/100.0;
double alpha = 0.0;
double beta = 1.0 / (intensity*dt) / 100.0;;
DipoleDir initialPolDir = DipoleDir::Z;
bool userRK4 = true;
bool useInterpolation = true;

std::vector<std::vector<double>> mu = {
    { 0.0, -mu12, mu12, 0.0 },
    { -mu12, 0.0, 0.0, mu12 },
    { mu12, 0.0, 0.0, -mu12 },
    { 0.0, mu12, -mu12, 0.0 }
};

std::vector<std::vector<double>> omega = {
    { 0.0, 0.0, 0.0, 0.0 },
    { 0.0, omega12, 0.0, 0.0 },
    { 0.0, 0.0, omega12, 0.0 },
    { 0.0, 0.0, 0.0, omega12 }
};
```

```
{ 0.0, 0.0, -omega12, 0.0 },
{ 0.0, 0.0, 0.0, 0.0 }
};

std::vector<std::vector<double>> gamma = {
    { 0.0, 0.0, 0.0, gamma12 },
    { 0.0, -gamma12 / 2.0, 0.0, 0.0 },
    { 0.0, 0.0, -gamma12 / 2.0, 0.0 },
    { 0.0, 0.0, 0.0, -gamma12 }
};

std::vector<std::vector<double>> initValues = {
    { 1.0, 0.0 },
    { 0.0, 0.0 },
    { 0.0, 0.0 },
    { 0.0, 0.0 }
};

std::vector<double> polTerms = {0.0,mu12,mu12,0.0};
std::vector<double> dipoleForceTerms = {0.0,mu12,mu12,0.0};

/// Choose which values to save
std::vector<bool> saveBlochReal = {true,true,true,true};
std::vector<bool> saveBlochImag = {false,false,true,true};
bool savePol = true;

//// Units conversion

std::vector<UnitsNames> lengthUnit = { UnitsNames::LENGTH };
std::vector<int> lengthPower = { 1 };
std::vector<int> densityPower = { -3 };

std::vector<UnitsNames> timeUnit = { UnitsNames::TIME };
```

```
std::vector<int> timePower = { 1 };

std::vector<int> frequencyPower = { -1 };

std::vector<UnitsNames> intensityUnit = {
    UnitsNames::MASS,
    UnitsNames::LENGTH,
    UnitsNames::TIME,
    UnitsNames::CHARGE
};

std::vector<int> intensityPower = { 1, 1, -2, -1 };

std::vector<UnitsNames> dipoleUnit = {
    UnitsNames::CHARGE,
    UnitsNames::LENGTH
};

std::vector<int> dipolePower = { 1, 1 };

std::vector<UnitsNames> dipoleVensorUnit = {
    UnitsNames::MASS,
    UnitsNames::LENGTH,
    UnitsNames::TIME,
    UnitsNames::CHARGE
};

std::vector<int> dipoleVensorPower = { -1, -1, 1, 1 };

/// Mesh constants

dx = units.convertToNatural(dx, lengthUnit, lengthPower);
dy = units.convertToNatural(dy, lengthUnit, lengthPower);
dz = units.convertToNatural(dz, lengthUnit, lengthPower);
dt = units.convertToNatural(dt, timeUnit, timePower);
Tmax = units.convertToNatural(Tmax, timeUnit, timePower);

/// EMF constants
```

```
R0 = units.convertToNatural(R0, lengthUnit, lengthPower);
waveNumber = units.convertToNatural(
    waveNumber,
    lengthUnit,
    frequencyPower
);
FWHM = units.convertToNatural(FWHM, lengthUnit, lengthPower);
intensity = units.convertToNatural(
    intensity,
    intensityUnit,
    intensityPower
);

/// Density operator constants
for (int i = 0; i < mu.size(); i++) {
    polTerms[i] = units.convertToNatural(
        polTerms[i],
        dipoleUnit,
        dipolePower
    );
    for (int j = 0; j < mu[0].size(); j++) {
        mu[i][j] = units.convertToNatural(
            mu[i][j],
            dipoleUnit,
            dipolePower
        );
        omega[i][j] = units.convertToNatural(
            omega[i][j],
            timeUnit,
            frequencyPower
        );
        gamma[i][j] = units.convertToNatural(
            gamma[i][j],
```

```
        timeUnit ,
        frequencyPower
    );
}

eta = units.convertToNatural(eta, lengthUnit, densityPower);
alpha = units.convertToNatural(
    alpha,
    dipoleVensorUnit,
    dipoleVensorPower
);
beta = units.convertToNatural(
    beta,
    dipoleVensorUnit,
    dipoleVensorPower
);

/// Mesh initialization
YeeMesh M = YeeMesh(units, Nx, Ny, Nz, dx, dy, dz, dt);

/// Field initialization
EMF EMField = EMF(M, true, false);
EMField.addGaussianPulsePlaneWave(
    direction,
    EPol,
    R0,
    waveNumber,
    phase,
    FWHM,
    intensity
);
EMField.initialize();
```

```
/// Densop initialization
Bloch densOp = Bloch(
    levels,
    eta,
    alpha,
    beta,
    M,
    mu,
    gamma,
    omega
);
densOp.Initialize(initValues, initialPolDir);

/// Aux current density and magnetization for EMF push
Field JField = zeros(EMField.mesh, FieldType::FACE, M.dtype());
Field MField = zeros(EMField.mesh, FieldType::EDGE, M.dtype());

/// Initial saves and directory creation
int i = 0;
if (save && resultsDirectory.empty()) {
    /// Create dir
    try {
        resultsDirectory = generateDirectoryPath("Maxwell-Bloch");
        MKDirectory(resultsDirectory);
    }
    catch (std::invalid_argument e) {
        std::cout << e.what() << std::endl;
        system("pause");
        return;
    }
}
if (save) {
```

```
    M.save(resultsDirectory, true);
    densOp.saveProperties(resultsDirectory, true);
}

while (i*dt<=Tmax) {
    std::cout << '\r' << i*dt/Tmax * 100 << std::flush;
    if (i % saveInterval == 0 && save) {
        densOp.saveToAF(
            resultsDirectory,
            saveBlochReal,
            saveBlochImag
        );
        EMField.saveToAF(
            resultsDirectory,
            fieldSaveSelector,
            true
        );
        if(savePol) {
            densOp.savePolToAf();
        }
    }
    EMField.push(
        JField,
        densOp.calculatePolarization(polTerms),
        MField
    );
    densOp.push(
        EMField.EfieldPrev(),
        EMField.Efield(),
        useRK4,
        useInterpolation
    );
    i++;
}
```

}

Appendix B

Thesis outputs

Oral presentations

- Developing numerical solvers using GPGPU computing to simulate light-atom interactions in atomic gases out of equilibrium, 2^as Jornadas em Engenharia Física, Física e Astronomia 2017
- Solving the multi-level Maxwell-Bloch equations using GPGPU computing for the simulation of nonlinear optics in atomic gases, III International Conference On Applications of Optics and Photonics

Poster presentations: First author

- Developing numerical solvers using GPGPU computing to simulate light-atom interactions in atomic gases out of equilibrium, 2^as Jornadas em Engenharia Física, Física e Astronomia 2017
- Fast physical ray-tracing method for gravitational lensing using heterogeneous supercomputing in GPGPU, III International Conference On Applications of Optics and Photonics

Poster presentations: Co-author

- SPaCe-GEM: solver of the Einstein equations using GPUs under the gravitoelectromagnetic approximation, III International Conference On Applications of Optics and Photonics

- Tunable light fluids using quantum atomic optical systems, III International Conference On Applications of Optics and Photonics
- Space-time refraction of light in time dependent media: the analogue within the analogue, III International Conference On Applications of Optics and Photonics
- Quantum wires as sensors of the electric field: a model into quantum plasmonics, 25th International Conference on Optical Fiber Sensors
- Doppler broadening effects in plasmonic quantum dots, III International Conference On Applications of Optics and Photonics
- Dissipative solitons in 4-level atomic optical systems, III International Conference On Applications of Optics and Photonics
- Physical ray-tracing method for anisotropic optical media in GPGPU, III International Conference On Applications of Optics and Photonics

Conference proceedings: First author

- Solving the multi-level Maxwell-Bloch equations using GPGPU computing for the simulation of nonlinear optics in atomic gases, III International Conference On Applications of Optics and Photonics
- Fast physical ray-tracing method for gravitational lensing using heterogeneous supercomputing in GPGPU, III International Conference On Applications of Optics and Photonics

Conference proceedings: Co-author

- SPaCe-GEM: solver of the Einstein equations using GPUs under the gravitoelectromagnetic approximation, III International Conference On Applications of Optics and Photonics
- Space-time refraction of light in time dependent media: the analogue within the analogue, III International Conference On Applications of Optics and Photonics
- Tunable light fluids using quantum atomic optical systems, III International Conference On Applications of Optics and Photonics

- Space-time refraction of light in time dependent media: the analogue within the analogue, III International Conference On Applications of Optics and Photonics
- Quantum wires as sensors of the electric field: a model into quantum plasmonics, 25th International Conference on Optical Fiber Sensors
- The analogue quantum mechanical of plasmonic atoms, III International Conference On Applications of Optics and Photonics
- Development of a quantum particle in cell algorithm in GPU for solving Maxwell-Bloch equations, III International Conference On Applications of Optics and Photonics
- Doppler broadening effects in plasmonic quantum dots, III International Conference On Applications of Optics and Photonics
- Pinching optical potentials for spatial nonlinearity management in Bose-Einstein condensates, III International Conference On Applications of Optics and Photonics
- Dissipative solitons in 4-level atomic optical systems, III International Conference On Applications of Optics and Photonics
- Physical ray-tracing method for anisotropic optical media in GPGPU, III International Conference On Applications of Optics and Photonics

Bibliography

- [1] P. Butcher and D. Cotter, *The Elements of Nonlinear Optics*, Cambridge Studies in Modern Optics (Cambridge University Press, 1991).
- [2] M. Fleischhauer, A. Imamoglu, and J. P. Marangos, "Electromagnetically induced transparency: Optics in coherent media," *Reviews of Modern Physics* **77** (2005).
- [3] M. Blaauboer, "Steady state behaviour in atomic three-level lambda and ladder systems with incoherent population pumping," *Phys. Rev. A* **55** (1997).
- [4] C. Hang and V. V. Konotop, "Spatial solitons in a three-level atomic medium supported by a Laguerre-Gaussian control beam," *Physical Review A* **83** (2011).
- [5] C. Hang, V. V. Konotop, and G. Huang, "Spatial solitons and instabilities of light beams in a three-level atomic medium with a standing-wave control field," *Phys. Rev. A* **79** (2009).
- [6] G. Huang, L. Deng, and M. G. Payne, "Dynamics of ultraslow optical solitons in a cold three-state atomic system," *Phys. Rev. E* **72** (2005).
- [7] R. G. Beausoleil, W. J. Munro, D. A. Rodrigues, and T. P. Spiller, "Applications of Electromagnetically Induced Transparency to Quantum Information Processing," *arXiv:quant-ph/0403028v3* (2005).
- [8] A. K. Sarma and P. Kumar, "High and uniform coherence creation in Doppler-broadened double lambda-like atomic system by a train of femtosecond optical pulses," *Laser Phys.* **25**, 056001 (2015).
- [9] T. Hong, M. W. Jack, M. Yamashita, and T. Mukai, "Enhanced Kerr nonlinearity for self-action via atomic coherence in a four-level atomic system," *Optics Communications* , 371 (2002).

- [10] T. Colin and V. Torri, "Numerical scheme for the 2D Maxwell-Bloch equations modeling ultrashort pulses," Preprint (2005).
- [11] J. Xiong, M. Colice, F. Schlottau, K. Wagner, and B. Fornberg, "Numerical solutions to 2D Maxwell–Bloch equations," Optical and Quantum Electronics , 7 (2008).
- [12] B. Bidégaray, A. Bourgeade, D. Reignier, and R. Ziolkowski, in *Fifth International Conference on Mathematical and Numerical Aspects of Wave Propagation* (Santiago de Compostela, Spain, 2000) pp. 221–225.
- [13] G. Demeter, "Solving the Maxwell-Bloch equations for resonant nonlinear optics using GPUs," Comput. Phys. Commun. **184**, 1203–1210 (2013).
- [14] A. Guerreiro and N. A. Silva, in *Second International Conference on Applications of Optics and Photonics* (2014).
- [15] R. Feynman, *Quantum Electrodynamics*, Advanced Books Classics (Avalon Publishing, 1998).
- [16] D. J. Griffiths, *Introduction to electrodynamics* (Prentice Hall, 1962).
- [17] J. Jackson, *Classical electrodynamics* (Wiley, 1975).
- [18] E. Hecht, "Optics," Addison Wesley **997**, 213 (1998).
- [19] J. J. Sakurai and E. D. Commins, "Modern quantum mechanics, revised edition," (1995).
- [20] C. Cohen-Tannoudji, B. Diu, and F. Laloë, *Quantum mechanics*, Quantum Mechanics (Wiley, 1977).
- [21] C. Cohen-Tannoudji, J. Dupont-Roc, and G. Grynberg, *Photons and Atoms: Introduction to Quantum Electrodynamics* (Wiley, 1997).
- [22] W. H. Louisell, *Quantum Statistical Properties of Radiation* (Wiley, 1973).
- [23] K. Hecht, *Quantum Mechanics*, Graduate Texts in Contemporary Physics (Springer New York, 2012).
- [24] M. Suzuki, "On the convergence of exponential operators—the Zassenhaus formula, BCH formula and systematic approximants," Communications in Mathematical Physics **57**, 193 (1977).

- [25] C. C. Gerry and P. L. Knight, *Introductory Quantum Optics*, edited by V. V. Dodonov and V. I. Man'ko (Cambridge University Press, 2005).
- [26] D. Walls and G. Milburn, *Quantum Optics*, SpringerLink: Springer e-Books (Springer Berlin Heidelberg, 2008).
- [27] L. Landau and E. Lifshitz, "On the theory of the dispersion of magnetic permeability in ferromagnetic bodies," *Phys. Z. Sowjetunion* **8**, 101 (1935).
- [28] A. Taflove and S. C. Hagness, *Computational electrodynamics: the finite-difference time-domain method*, 3rd ed. (Artech House, Norwood, 2005).
- [29] M. Spivak, *Calculus*, Calculus (Cambridge University Press, 2006).
- [30] H. Pina, *Métodos Numéricos*, edited by E. Editora (Escolar Editora, 2010).
- [31] P. K. Das and G. C. Deka, in *Emerging Research Surrounding Power Consumption and Performance Issues in Utility Computing* (IGI Global, 2016) pp. 109–135.
- [32] F. NVidia, "Nvidia's next generation cuda compute architecture," NVidia, Santa Clara, Calif, USA (2009).
- [33] P. Pacheco, *An Introduction to Parallel Programming*, 1st ed. (Morgan Kaufmann, 2011).
- [34] C. T. Documentation, "v8.0," NVIDIA Corporation, February (2017).
- [35] J. Sanders and E. Kandrot, *CUDA by example*, 1st ed. (NVIDIA, 2010).
- [36] R. Banger and K. Bhattacharyya, *OpenCL Programming by Example*, 1st ed. (Packt Publishing, 2013).
- [37] Nvidia, "Thrust quick start guide," (2016).
- [38] P. Yalamanchili, U. Arshad, Z. Mohammed, P. Garigipati, *et al.*, "ArrayFire - A high performance software library for parallel computing with an easy-to-use API," (2015).
- [39] N. Corporation, "What is gpgpu computing," <http://www.nvidia.com/object/what-is-gpu-computing.html>, (seen on: 24-01-17).
- [40] "IEEE Standard for Floating-Point Arithmetic," IEEE Std. 754 - 2008 (2008).
- [41] N. M. Josuttis, *The C++ standard library: a tutorial and reference* (Addison-Wesley, 2012).

- [42] W. Rabinovich and B. Feldman, "Spatial hole burning effects in distributed feedback lasers," IEEE Journal of Quantum Electronics **25**, 20 (1989).
- [43] H. Fehske, R. Schneider, and A. Weiße, *Computational Many-Particle Physics*, Lecture Notes in Physics (Springer Berlin Heidelberg, 2007).
- [44] K. Huang, *Statistical mechanics* (Wiley, 1987).
- [45] T. Lahaye, C. Menotti, L. Santos, M. Lewenstein, and T. Pfau, "The physics of dipolar bosonic quantum gases," Reports on Progress in Physics **72**, 126401 (2009).
- [46] B. L. Schmittberger and D. J. Gauthier, "Transverse optical and atomic pattern formation," Journal of the Optical Society of America B **33**, 1543 (2016).