DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Simulation of a Bayard-Alpert ionization gauge with the PIC code Warp

ALEXANDER TEGERUP

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Simulation of a Bayard-Alpert ionization gauge with the PIC code Warp

ALEXANDER TEGERUP

Degree Projects in Scientific Computing (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2018
Supervisor at RISE: Olle Penttinen, Johan Anderson
Supervisor at KTH: Patrik Henning
Examiner at KTH: Michael Hanke

TRITA-SCI-GRU 2018:001
MAT-E 2018:01

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Abstract

At *RISE Research Institutes of Sweden*, there is an interest in computer simulations of the physics related to ionization gauges. The objective of this thesis is to find out if the open source code *Warp* can be used for simulating the physics of interest.

In this thesis it is explained what an ionization gauge is and the physics and the governing mathematical equations of the simulations are described. How those governing equations are solved and the algorithms used in Warp is also discussed in this thesis.

The results of the simulations are presented in the thesis and a discussion of which parts of Warp that need to be further developed to successfully simulate the physics is carried through.

Referat

Simulering av en Bayard-Alpert joniserande tryckgivare med PIC-koden Warp

På *RISE Research Institutes of Sweden*, är man intresserad av att göra datorsimuleringar av fysiken bakom joniserande tryckgivare. Målet med denna uppsats är att ta reda på om det är möjligt att använda den öppna källkoden *Warp* för att genomföra simuleringar av fysiken som man är intresserad av.

I den här uppsatsen förklaras det vad en joniserande tryckgivare är och fysiken och de styrande matematiska ekvationerna bakom simuleringarna beskrivs. Hur dessa styrande ekvationer löses och algoritmerna som används i Warp diskuteras också i denna uppsats.

Resultaten från simuleringarna presenteras i uppsatsen och det förs en diskussion om vilka delar utav Warp som behöver vidareutvecklas för att framgångsrikt kunna simulera fysiken.

Acknowledgements

I would like to thank my supervisors Olle Penttinen and Johan Anderson for giving me so much feedback on the work I have done on this thesis. I have had giving discussions with Johan and Olle regarding how to solve problems that I have run in to during this work and their suggestions and ideas have helped me a lot.

I would also like to thank my supervisor Patrick Henning at Numerisk Analys at KTH for guiding me and for giving me constructive suggestions of how to improve this thesis.

Finally, I would like to thank my parents for supporting and encourage me throughout my years of study. Thank you.

Author
Alexander Tegerup

Contents

1 Introduction **1**

1.1 Objective . . . 1

1.2 Ionization gauges and the principles behind them . . . 2

1.3 Warp . . . 3

1.4 Model description . . . 3

1.5 Input and output parameters . . . 6

2 Physics related to the simulations **7**

2.1 Maxwell's equations . . . 7

2.1.1 Gauss law for electrical fields . . . 7

2.1.2 Gauss law for magnetism . . . 8

2.1.3 Faraday's law in integral form . . . 8

2.1.4 Ampère's circuital law . . . 9

2.1.5 Summary of Maxwell's equations . . . 10

2.2 Physics derived from Maxwell's equations . . . 11

2.2.1 Electromagnetic case . . . 11

2.2.2 Electrostatic case . . . 11

2.3 Particle movement . . . 13

2.4 Additional physics needed for the simulations . . . 14

2.4.1 Debye length . . . 14

2.4.2 Electron speed . . . 15

2.4.3 Boltzmann electrons . . . 15

2.4.4 Plasma frequency and impact ionization . . . 16

3 Mathematical equations and algorithms **17**

3.1 Known physical quantities . . . 17

3.2 Physical quantities to compute . . . 18

3.2.1 Boundary conditions . . . 19

3.3 The Particle in Cell (PIC) method . . . 20

3.3.1 Motivation for using the PIC method . . . 20

3.3.2 The basics of PIC . . . 21

3.3.3 Electromagnetic PIC . . . 23

3.3.4 Electrostatic PIC . . . 25

3.4 Field solvers 29
3.4.1 The electrostatic MultiGrid3D solver 29
3.5 Particle push 32
3.5.1 What the particle push is 32
3.5.2 Considerations regarding the push algorithm 33
3.5.3 The leapfrog method 34
3.5.4 The Boris pusher 35
3.5.5 Particle push in Warp 41
3.6 Calculation of ionization 42

4 Simulation **47**
4.1 Simulation setup 47
4.1.1 Conducting objects 47
4.1.2 Classes created for this work 47
4.2 Tests of different parts of Warp 50
4.2.1 Absorption of particles 50
4.2.2 Field solvers 51
4.2.3 Particle movement and the ion collector 53
4.3 Approach to simulate the ionization gauge 58
4.4 CFL-condition and stability 63
4.5 Results of the simulations 65
4.5.1 Issue with the absorption of electrons 66
4.5.2 The ionization 67
4.5.3 Long simulations 71
4.6 Discussion and future work 72

Appendices **74**

A Definitions and notation **75**

B Used mathematical identities **77**

C Installation of Warp **81**

D Debugging of the Warp code **83**

Bibliography **85**

Chapter 1

Introduction

1.1 Objective

At *RISE Research Institutes of Sweden*, there is an interest in making computer simulations of ionization gauges. There are certain properties of the ionization gauges that are of special interest, such as *the sensitivity of the gauges* and *the ion collection efficiency*. These concepts will be explained in this thesis. RISE also wants to investigate if the open source code *Warp* can be used for making the simulations of the ionization gauges.

The main objective of this thesis is to investigate if it is possible to carry through simulations of ionization gauges and obtain reasonable results of certain properties of the gauges. To understand the outputs of the simulations, some physical and mathematical theory will be explained in this thesis as well. Some concepts that will be described are

- The input parameters of the model.
- Physics related to ionization gauges and which simplifications of the physics that have been made.
- Mathematical equations used in the simulations and some explanation of what those equations mean.
- Which algorithms that are used in the simulations and what advantages and disadvantages they have.
- Which numerical methods that are used to solve the mathematical equations and to make the simulations.

1.2 Ionization gauges and the principles behind them

According to [1], an *ionization gauge* is a device used to measure pressures up to $10^{-1}\ Pa$. According to [2], pressures of size $10^{-12}\ Pa$ have been measured by ionization gauges, but there are several variants of ionization gauges and the limit of the lowest measurable pressure depend on which type of ionization gauge that is being used. In this work, the gauge type of interest is called *Bayard-Alpert ionization gauge* and an example of a gauge of such kind is shown in Figure 1.1. Important principles of a gauge of such kind for the simulations in this work, is explained further below in this section. Further details can be found in [3].

Important parts of the ionization gauge simulated in this work are:

- the hot cathode
- the ion collector
- the anode grid

These parts are illustrated in Figure 1.1 and their functionality in the ionization gauge is explained below. More information about Bayard-Alpert ionization gauges can be found in for example [3].

An electron current run through the hot cathode and due to the high temperature of the cathode, electrons are emitted from it. According to [4], the emission depends on the work function. The work function is the minimum amount of energy needed to eject an electron from a surface, cf. [5]. In [3] it described that the anode grid is positively charged and will attract the electrons emitted from the cathode, even though many of the electrons that are traveling from the cathode towards the anode grid do not collide with the anode grid. Instead, some of these electrons ionize some of the gas molecules that exist in the environment where the cathode, anode grid and ion collector are placed.

The ion collector is placed inside of the anode grid and the ion collector is grounded. Positive ions that are generated by the electrons from the hot cathode are collected by the ion collector. This way, a positive ion current that can be measured is produced. If the electron current from the hot cathode is constant, the ion current is directly proportional to the gas density

$$I_i \propto I_e \cdot \rho \tag{1.1}$$

where I_e is the electron current from the hot cathode, ρ is the gas density and I_i is the ion current. According to [6], at constant temperature the gas density is directly proportional to the low pressure p that is to be measured. This means that the ion current can be used as an indicator of the pressure:

$$I_i \propto I_e \cdot p \tag{1.2}$$

In [3], equation (1.2) is expressed as

$$I_i = S \cdot I_e \cdot p \tag{1.3}$$

where S is the *gauge sensitivity factor* and the product $I_e \cdot S$ is the *sensitivity* of the gauge. There are several parameters that can affect the sensitivity of a gauge, such as the pressure in the gauge, which type of gas that is used and the geometry of the gauge. One important objective of this work is to investigate this gauge sensitivity factor by doing computer simulations.

Another interesting parameter related to ionization gauges is the *ion collection efficiency*. It is defined as the number of ions collected by the ion collector divided by the total number of ions created in the ionization gauge.

1.3 Warp

Warp is the name of the software used for the simulations in this work. In [7] there is information about the software Warp and how to use it. They state that Warp is an open source code that have been developed since the 1980s. It is designed to simulate charged particle beams and the name Warp comes from the codes ability to simulate bent ("warped") Cartesian meshes. In this work however, no particle beams nor bent Cartesian meshes will be implemented.

In [8] it is mentioned that Warp uses a *particle in cell (PIC)* model to describe electrostatic or electromagnetic fields, and the core routines of Warp solves finite-difference representations of Maxwell's equations. The code that handles computationally intensive tasks is written in Fortran while Python is used for the high level controlling framework.

1.4 Model description

In [9] there is a description of a model of a Bayard-Alpert ionization gauge. That model was used as a template for the model of the ionization gauge implemented in the Warp code in this thesis. The anode grid of the ionization gauge that was implemented in the simulations in this work, consists of several toruses placed right above each other, instead of the spiral formed anode grid used in [9]. A CAD model of the ionization gauge implemented in Warp were made in the software FreeCAD, and is shown in Figure 1.1.

In Figure 1.1, the arcs shaped as a "V" placed upside down represent the filaments and electrons are emitted only from the filament to the left in the picture. The two sticks inside of the anode grid represent the modulators and the thin stick at the centre represents the ion collector. The thin toruses are together referred to as the anode grid. The thicker sticks that join the anode grid together are made for holding up the anode grid and are referred to as the grid supports.

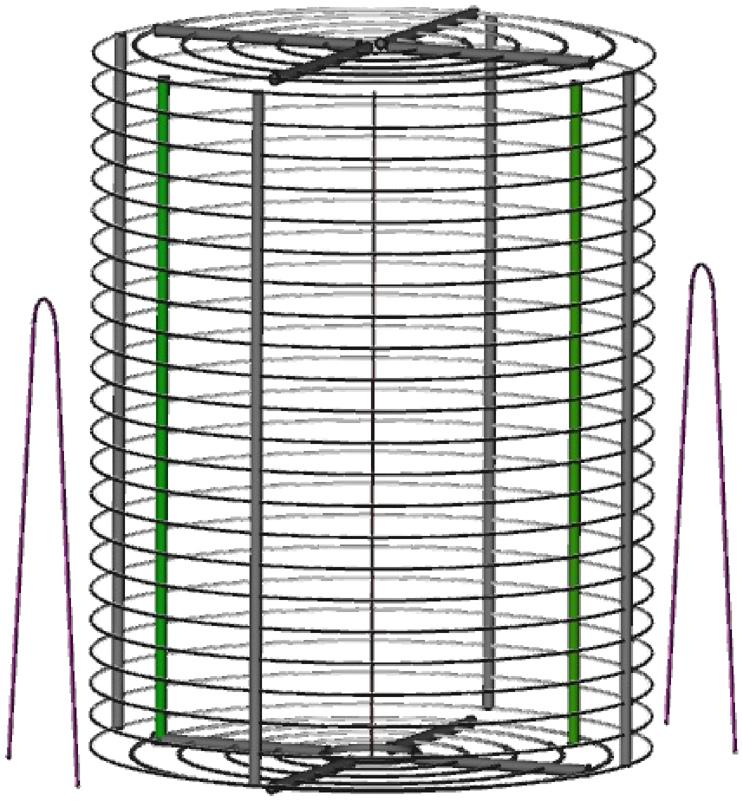


Figure 1.1: A model of the Bayard-Alpert ionization gauge implemented in the Warp simulations in this work.

Table 1.1: Numerical data over the model of the ionization gauge implemented in Warp. The columns x, y and z show where the items of the gauge are placed in the domain. All the units are given in millimeters if nothing else is stated.

Item	x	y	z	Height	Diameter	Potential [Volt]
Ion collector	0	0	5	42	0.05	0
Emitting cathode	-22	0	5	30	0.18	+50
Non emitting cathode	22	0	5	30	0.18	+50
Modulator 1	13.5	0	5	42	0.7	+150
Modulator 2	-10.3	10.3	5	42	0.7	+150
Grid support 1	-12.4	12.4	5.5	44	0.7	-
Grid support 2	12.4	12.4	5.5	44	0.7	-
Grid support 3	12.4	-12.4	5.5	44	0.7	-
Grid support 4	-12.4	-12.4	5.5	44	0.7	-
Anode grid torus *	0	0			0.26	+150

* The poloidal radius of the anode grid torus is 0.065 millimeter and the toroidal radius is 17.5 millimeter. There are 23 anode grid toruses and their centres in the toroidal direction are separated 2 millimeters from each other. The poloidal and toroidal directions are shown in Figure 1.2.

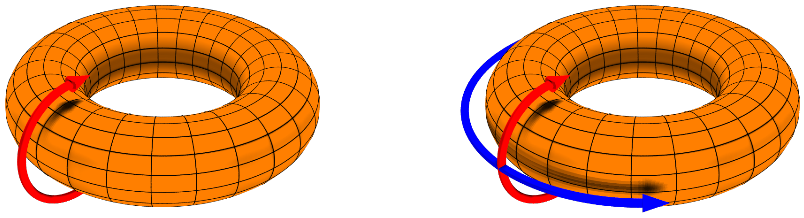


Figure 1.2: In picture (a), the arrow shows the poloidal direction of the torus. In picture (b), an additional arrow shows the toroidal direction of the torus. Pictures by DaveBurke - Own work, CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=10416598

1.5 Input and output parameters

The input parameters in the simulations carried through in this work are the number of emitted electrons from the hot cathode (the electron current I_e), the pressure inside the domain, which type of gas molecules the particles represents and the temperature in the domain. The number density n stands for number of moles of the gas particles in the domain. n is calculated from the chosen pressure via the *ideal gas law*

$$pV = nRT \tag{1.4}$$

where V is the volume that contains the gas, T is the temperature and R is the *ideal-gas constant*, cf. [5].

The parameters that are to be computed in this work are the sensitivity of the gauge and the ion collection efficiency. The ion current I_i is an output of the simulations and if the ion current is known, the sensitivity of the gauge can be calculated from equation (1.3).

The computational domain in which the ionization gauge is placed have reflective boundaries, which means that the particles are reflected by the walls of the domain.

Chapter 2

Physics related to the simulations

In this work, the presence of an electric field $\boldsymbol{E}$ and a magnetic field $\boldsymbol{B}$ will give rise to motion of the particles in the domain Ω. All relationships between an electric field and a magnetic field and their sources, can be expressed by *Maxwell's equations* [5]. Maxwell's equations are used for calculating the electric and magnetic field in the simulations in this work and together with *Newtons second law*, Maxwell's equations govern the motion of the particles in the simulations. Therefore, Maxwell's equations will be stated and explained in Section 2.1.

Other physical laws that are used for calculating the fields and the motion of the particles are derived from Maxwell's equations. In Section 2.2 and Section 2.3, those derivations will be carried through. Some additional physics that need to be considered in the particle simulations in this work is described in Section 2.4.

2.1 Maxwell's equations

Maxwell's equations consist of four equations that all can be expressed in integral form or in differential form [10]. The integral form and the differential form are mathematically equivalent, meaning that if the integral form holds then the differential form holds and vice-versa.

2.1.1 Gauss law for electrical fields

Gauss law for electrical fields in integral form is stated as

$$\oint_{\partial\nu} \boldsymbol{E} \cdot \boldsymbol{n} \, d\sigma(x) = \frac{1}{\epsilon_0} \int_{\nu} \rho \, dx \tag{2.1}$$

where $\partial\nu$ is a closed surface and ν is a volume bounded by the surface $\partial\nu$ [10]. ρ is the charge density and ϵ_0 is the vacuum permittivity. Equation (2.1) says that the total electric flux through a closed surface is proportional to the total electric charge inside of that surface [5]. According to the divergence theorem stated in

Appendix B, the left hand side of equation (2.1) can be expressed as

$$\oint_{\partial\nu} \boldsymbol{E} \cdot \boldsymbol{n} \, d\sigma(x) = \int_{\nu} (\nabla \cdot \boldsymbol{E}) \, dx \tag{2.2}$$

and from equations (2.1) and (2.2) one gets that

$$\int_{\nu} (\nabla \cdot \boldsymbol{E}) \, dx = \int_{\nu} \frac{\rho}{\epsilon_0} \, dx \tag{2.3}$$

Since the test volume ν is arbitrary, equation (2.3) implies that

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\epsilon_0} \tag{2.4}$$

Equation (2.4) is referred to as *Gauss law in differential form* [10].

2.1.2 Gauss law for magnetism

Gauss law for magnetism in integral form is stated as

$$\oint_{\partial\nu} \boldsymbol{B} \cdot \boldsymbol{n} \, d\sigma(x) = 0 \tag{2.5}$$

where $\partial\nu$ again is a closed surface. Equation (2.5) is comparable to equation (2.1), where one difference is that the right hand side of equation (2.5) always is zero. The reason is that no magnetic monopoles that could be sources of magnetic fields have empirically been found [5]. According to the divergence theorem stated in Appendix B, the integral in equation (2.5) can be rewritten as

$$\oint_{\partial\nu} \boldsymbol{B} \cdot \boldsymbol{n} \, d\sigma(x) = \int_{\nu} (\nabla \cdot \boldsymbol{B}) \, dx \tag{2.6}$$

and then it must hold that

$$\int_{\nu} (\nabla \cdot \boldsymbol{B}) \, dx = 0 \tag{2.7}$$

Equation (2.7) holds for an arbitrary test volume ν if

$$\nabla \cdot \boldsymbol{B} = 0 \tag{2.8}$$

and equation (2.8) is referred to as *The differential form of Gauss law for magnetic fields* [10].

2.1.3 Faraday's law in integral form

Faraday's law in integral form is expressed as

$$\oint_{\partial S} \boldsymbol{E} \cdot \boldsymbol{\tau} \, dx = -\frac{\partial}{\partial t} \int_{S} \boldsymbol{B} \cdot \boldsymbol{n} \, d\sigma(x) \tag{2.9}$$

where S is a surface in 3 dimensions and ∂S is the closed curve that makes up the boundary of S. The total *magnetic flux* Φ_B through the surface S is expressed as

$$\Phi_B = \int_S \boldsymbol{B} \cdot \boldsymbol{n} \, d\sigma(x) \tag{2.10}$$

and equation (2.9) states that an electric field is induced by changing the magnetic field or the magnetic flux in time. If the magnetic field is not constant in time, the line integral in equation (2.9) is non-zero, which means that the induced electric field is nonconservative. Such fields are called nonelectrostatic fields [5]. According to *Stokes theorem* in Appendix B, is

$$\oint_{\partial S} \boldsymbol{E} \cdot \boldsymbol{\tau} \, dx = \int_S (\nabla \times \boldsymbol{E}) \cdot \boldsymbol{n} \, d\sigma(x) \tag{2.11}$$

which means that

$$\int_S (\nabla \times \boldsymbol{E}) \cdot \boldsymbol{n} \, d\sigma(x) = \int_S \big(- \frac{\partial}{\partial t}(\boldsymbol{B})\big) \cdot \boldsymbol{n} \, d\sigma(x) \tag{2.12}$$

Equation (2.12) holds for an arbitrary surface S if

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{2.13}$$

and equation (2.13) is referred to as *Faraday's law in differential form* [10].

2.1.4 Ampère's circuital law

The fourth of Maxwell's equations is *Ampère's circuital law* which can be stated in integral form as

$$\oint_{\partial S} \boldsymbol{B} \cdot \boldsymbol{\tau} \, dx = \mu_0 \big(\int_S \boldsymbol{J} \cdot \boldsymbol{n} \, d\sigma(x) + \epsilon_0 \frac{\partial}{\partial t} \int_S \boldsymbol{E} \cdot \boldsymbol{n} \, d\sigma(x)\big) \tag{2.14}$$

where S is a surface in 3 dimensions and ∂S is a closed curve that makes up the boundary of the surface S. μ_0 is the magnetic constant, $\mathbf{J}$ is *the volume current density* and the first surface integral on the right hand side of equation (2.14) represent the total current passing through the surface S. The second term within the parenthesis in the right hand side of equation (2.14) is called *the displacement current*. This current is proportional to the time derivative of the electrical flux that is passing through the surface S. The displacement current in equation (2.14) means that changing an electric field induces a magnetic field.

Hence, equation (2.14) states that the line integral of $\mathbf{B}$ around the closed curve ∂S is proportional to the total current that passes through the surface S plus the time derivative of the electric flux that passes through the surface S. This is further explained in [5]. By Stokes theorem in Appendix B, the left hand side of equation (2.14) can be rewritten as

$$\oint_{\partial S} \boldsymbol{B} \cdot \boldsymbol{\tau}\, dx = \int_S (\nabla \times \boldsymbol{B}) \cdot \boldsymbol{n}\, d\sigma(x) \tag{2.15}$$

and equation (2.14) can therefore be stated as

$$\int_S (\nabla \times \boldsymbol{B}) \cdot \boldsymbol{n}\, d\sigma(x) = \mu_0 \big(\int_S \boldsymbol{J} \cdot \boldsymbol{n}\, d\sigma(x) + \epsilon_0 \frac{\partial}{\partial t} \int_S \boldsymbol{E} \cdot \boldsymbol{n}\, d\sigma(x) \big). \tag{2.16}$$

Since S is an arbitrary surface, equation (2.16) holds if

$$\nabla \times \boldsymbol{B} = \mu_0 (\boldsymbol{J} + \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t}). \tag{2.17}$$

Equation (2.17) is referred to as *the differential form of Ampère's circuital law* [10].

2.1.5 Summary of Maxwell's equations

Below is a summary of Maxwell's equations.

Gauss law for electric fields

Integral form	*Differential form*
$\oint_{\partial \nu} \boldsymbol{E} \cdot \boldsymbol{n}\, d\sigma(x) = \frac{1}{\epsilon_0} \int_\nu \rho\, dx$	$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\epsilon_0}$

Gauss law for magnetism

Integral form	*Differential form*
$\oint_{\partial \nu} \boldsymbol{B} \cdot \boldsymbol{n}\, d\sigma(x) = 0$	$\nabla \cdot \boldsymbol{B} = 0$

Faraday's law

Integral form	*Differential form*
$\oint_{\partial S} \boldsymbol{E} \cdot \boldsymbol{\tau}\, dx = -\frac{\partial}{\partial t} \int_S \boldsymbol{B} \cdot \boldsymbol{n}\, d\sigma$	$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}$

Ampère's circuital law

Integral form	*Differential form*
$\oint_{\partial S} \boldsymbol{B} \cdot \boldsymbol{\tau}\, dx = \mu_0 \big(\int_S \boldsymbol{J} \cdot \boldsymbol{n}\, d\sigma + \epsilon_0 \frac{\partial}{\partial t} \int_S \boldsymbol{E} \cdot \boldsymbol{n}\, d\sigma \big)$	$\nabla \times \boldsymbol{B} = \mu_0 (\boldsymbol{J} + \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t})$

2.2 Physics derived from Maxwell's equations

2.2.1 Electromagnetic case

In the electromagnetic case, the electric and magnetic fields $\boldsymbol{E}$ and $\boldsymbol{B}$ are time-dependent. Faraday's law in integral form stated in equation (2.9), Section 2.1, shows that in this time-dependent case the electrical field is not a conservative field if the magnetic field changes in time. It was also stated in Section 2.1 that the magnetic field $\boldsymbol{B}$ is divergence free, so from the Helmholtz theorem in Appendix B one get that $\boldsymbol{B}$ can be expressed as

$$\boldsymbol{B} = \nabla \times \boldsymbol{A} \tag{2.18}$$

where $\boldsymbol{A}$ is the *magnetic vector potential.* By combining equation (2.18) with Faraday's law in differential form in equation (2.13), one get that

$$\nabla \times (\boldsymbol{E} + \frac{\partial \boldsymbol{A}}{\partial t}) = \boldsymbol{0}. \tag{2.19}$$

In Appendix B it is stated that if the curl of a vector field is zero everywhere, that vector field can then be written as the gradient of a scalar potential. This means that one can write

$$\boldsymbol{E} + \frac{\partial \boldsymbol{A}}{\partial t} = -\nabla \phi \tag{2.20}$$

and the electrical field can be expressed as

$$\boldsymbol{E} = -\nabla \phi - \frac{\partial \boldsymbol{A}}{\partial t} \tag{2.21}$$

By applying the divergence to Ampère's circuital law, expressed by equation (2.17), one gets

$$\begin{aligned}
\nabla \cdot (\nabla \times \boldsymbol{B}) &= \nabla \cdot (\mu_0 \boldsymbol{J} + \mu_0 \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t}) && \text{(2.22a)}\\
\nabla \cdot \boldsymbol{J} &= -\epsilon_0 \frac{\partial (\nabla \cdot \boldsymbol{E})}{\partial t} && \text{(2.22b)}\\
\nabla \cdot \boldsymbol{J} &= -\frac{\partial \rho}{\partial t} && \text{(2.22c)}
\end{aligned}$$

Here, Gauss law (equation (2.4)) and the fact that the divergence of any vector is zero was used. Equation (2.22c) is referred to as the *continuity equation*, which states that the charge per unit time leaving a volume V will decrease the charge left inside of volume V. See for instance [10] for further details about the continuity equation and the electromagnetic case.

2.2.2 Electrostatic case

Problems involving time-independent fields are referred to as electrostatic problems. The reason for that name is that the time derivative of the magnetic field is zero, which means that the magnetic field is static. In this case, Faraday's law in differential form given by equation (2.13), reduces to

$$\nabla \times \boldsymbol{E} = \mathbf{0} \tag{2.23}$$

and from the Helmholtz theorem in Appendix B, it holds that

$$\mathbf{E} = -\nabla\phi \tag{2.24}$$

From equation (2.24) and equation (2.4) one get that

$$\triangle\phi = -\frac{\rho}{\epsilon_0} \tag{2.25}$$

Equation (2.25) is Poisson's equation.

Also, Ampère's law in differential form given by equation (2.17), gets reduced to

$$\nabla \times \boldsymbol{B} = \mu_0 \boldsymbol{J} \tag{2.26}$$

See for instance [10] for further details.

In an electrostatic case, a potential difference between a point **a** and a point **b** can be expressed as

$$\phi(\boldsymbol{b}) - \phi(\boldsymbol{a}) = -\int_{\boldsymbol{a}}^{\boldsymbol{b}} \mathbf{E} \cdot \boldsymbol{\tau}\, dx \tag{2.27}$$

where the electric potential ϕ is a scalar function dependent on the x-, y- and z-coordinates in space. If the potential on the surfaces of conducting objects is constant, equation (2.27) reduces to

$$\int_{\boldsymbol{a}}^{\boldsymbol{b}} \mathbf{E} \cdot \boldsymbol{\tau}\, dx = 0 \tag{2.28}$$

which means that the electric field is perpendicular to those surfaces. This is explained further in [5].

Summary of important equations in the electrostatic case

The Maxwell's equations in differential form in the electrostatic case are summarized as

$$\begin{aligned} \nabla \cdot \boldsymbol{E} &= \frac{\rho}{\epsilon_0} \\ \nabla \cdot \boldsymbol{B} &= 0 \\ \nabla \times \boldsymbol{E} &= 0 \\ \nabla \times \boldsymbol{B} &= \mu_0 \boldsymbol{J} \end{aligned}$$

and the equations

$$\begin{aligned} \boldsymbol{E} &= -\nabla\phi \\ \triangle\phi &= -\frac{\rho}{\epsilon_0} \end{aligned}$$

are derived from Maxwell's equation in the electrostatic case.

2.3 Particle movement

In this work, motion of particles such as neutral molecules, ions and electrons are simulated. Therefore, this section will describe the physics that govern the motion of such particles.

It is explained in [5] that the electric force on a particle p with charge q_p and velocity $\boldsymbol{v}_p$ is

$$\boldsymbol{F}_E = q_p \boldsymbol{E}_p \tag{2.31}$$

and the magnetic force on the same particle is

$$\boldsymbol{F}_B = q_p(\boldsymbol{v}_p \times \boldsymbol{B}_p) \tag{2.32}$$

If both an electric and a magnetic field affect the particle, the total force on that particle is

$$\boldsymbol{F}_p = \boldsymbol{F}_E + \boldsymbol{F}_B = q_p(\boldsymbol{E}_p + \boldsymbol{v}_p \times \boldsymbol{B}_p) \tag{2.33}$$

which is referred to as the *Lorentz force.* Newton's second law states that

$$\boldsymbol{F}_p = \frac{d\boldsymbol{p}_p}{dt} \tag{2.34}$$

where $\boldsymbol{p}_p = m_p \boldsymbol{v}_p$ is the particles momentum and m_p is the particles mass. By combining equation (2.33) and equation (2.34), an expression relating the particle velocity to the electric and magnetic fields $\boldsymbol{E}$ and $\boldsymbol{B}$ is stated as

$$\frac{d\boldsymbol{v}_p}{dt} = \frac{q_p}{m_p}(\boldsymbol{E}_p + \boldsymbol{v}_p \times \boldsymbol{B}_p) \tag{2.35}$$

If constant magnetic and electric fields are present, the electric fields can be divided into two components as

$$\boldsymbol{E} = \boldsymbol{E}_\perp + \boldsymbol{E}_{||} \tag{2.36}$$

where $\boldsymbol{E}_\perp$ is the component perpendicular to the magnetic field and $\boldsymbol{E}_{||}$ is the component parallel to the magnetic field [11]. If only the component $\boldsymbol{E}_{||}$ is present, it will give the particle a constant acceleration in the direction of the magnetic field. This is because the cross product

$$\boldsymbol{v}_p \times \boldsymbol{B}_p \tag{2.37}$$

given in equation (2.35) becomes zero. The velocity vector parallel to the magnetic field will be denoted $\boldsymbol{v}_{||}$.

According to [12], if $\boldsymbol{E}_\perp$ is zero, the particle will only make a circular motion in the plane perpendicular to the magnetic field, due to the cross product in (2.37). The velocity vector that describes this circular motion will be denoted $\boldsymbol{v}_g$ and the radius of this circular particle motion is called the *gyration radius*, expressed as

$$r_g = \frac{m_p|\boldsymbol{v}_g|}{|q_p||\boldsymbol{B}|} \tag{2.38}$$

This radius is also known as the *Larmor radius* or the *cyclotron radius.* The frequency of the circular particle motion is called *gyrofrequency*, expressed as

$$\omega_g = \frac{q_p|\boldsymbol{B}|}{m_p} \tag{2.39}$$

and the *gyroperiod* is

$$\tau_g = \frac{1}{\omega_g}. \tag{2.40}$$

If $\boldsymbol{E}_\perp \neq \boldsymbol{0}$, the particle will start to drift in the plane perpendicular to the magnetic field while keeping the gyration motion. This *drift velocity* is called $E \times B$ *drift* and is expressed as

$$\boldsymbol{v}_d = \frac{\boldsymbol{E} \times \boldsymbol{B}}{|\boldsymbol{B}|^2} \tag{2.41}$$

which means that it is perpendicular to $\boldsymbol{E}_\perp$ and independent of the particle charge. The velocity of the particle can finally be expressed as the sum of the three vectors

$$\boldsymbol{v}_p = \boldsymbol{v}_\parallel + \boldsymbol{v}_d + \boldsymbol{v}_g \tag{2.42}$$

where the velocity vectors $\boldsymbol{v}_d$ and $\boldsymbol{v}_g$ express the particle motion in a plane perpendicular to the magnetic field. The velocity of the particle in that plane is expressed by the vector

$$\boldsymbol{v}_\perp = \boldsymbol{v}_d + \boldsymbol{v}_g \tag{2.43}$$

does not change if the electric and magnetic field remain constant.

2.4 Additional physics needed for the simulations

2.4.1 Debye length

Plasma is defined as ionized gas that approximately is electrically neutral in average [11]. This electrical neutrality of plasma is referred to as *quasi-neutrality* and is fulfilled for spatial dimensions larger than a characteristic length called the *Debye length*, denoted λ_d. At distances shorter than λ_d, charges can be separated so that clouds of positively charged particles and clouds of negatively charged particles appear. These clouds give rise to *space charges* and local electrical fields in the plasma [13]. It is stated in [14] that the mass of the ions is much greater than the mass of the electrons which leads to that the electrons move much faster than the ions. If the ions are considered to be fixed in space compared to the movements of the electrons, the Debye length can be expressed as

$$\lambda_d = \sqrt{\frac{k_B T_e \epsilon_0}{n_e q_e^2}} \tag{2.44}$$

where k_B is Boltzmann's constant, T_e is the temperature of the electrons, n_e is the number density of the electrons and q_e is the elementary charge.

In this work the plasma is made up by neutral molecules, positive ions and electrons. The pressure in the simulations is in the range $10^{-8} - 1\ Pa$, hence the plasma is defined as *low-pressure plasma.*

2.4.2 Electron speed

The work function, denoted W, is the minimum amount of energy needed to eject an electron from a surface, cf. [5]. The kinetic energy of an electron is

$$E_k = \frac{m_e v_e^2}{2} \tag{2.45}$$

where m_e is the mass and v_e is the speed of an electron. If an electron should be able to escape from the hot cathode, it is necessary that

$$E_k > W \tag{2.46}$$

and the initial speed of an electron emitted from the hot cathode is calculated as

$$v = \sqrt{\frac{2W}{m_p}} \tag{2.47}$$

There are list of values of the work function for different types of materials available, for example in [15]. In this work the value 5 eV is used. It is also taken into account that the electrons have to escape from the applied potential on the emitter, which is 50 V. This way, the speed the electrons when they are emitted is calculated to be $4.42{\cdot}10^6$ m/s. For more information about the work function, see for instance [15].

The mass of an electron is about $9.1 \cdot 10^{-31}$ kg and its charge is about $1.6 \cdot 10^{-19}$ C, cf. [5]. For an electron, the fraction $\frac{q_p}{m_p}$ in equation (2.35) is of magnitude $10^{11}\ \frac{\text{C}}{\text{kg}}$ and it can be concluded that an electron will move very fast if a force is acting on it. The mass of the hydrogen ions (one proton) that are simulated in this work, is several orders of magnitudes larger than the mass of the electrons and from the electrons perspective, the ions are almost not moving at all.

2.4.3 Boltzmann electrons

To simulate the motion of the electrons correctly, it is necessary to have a small time step. But with a small time step many iterations of the simulation has to be carried through to simulate the motion of the ions. For large scale systems, the computations may be heavy if the electrons are represented as macro particles [16].

The computations can be simplified by considering the electrons as a fluid [16]. According to [17], the charge density from the charged particles in the plasma is expressed as

$$\sigma = n_i \cdot q_i + n_e \cdot q_e \tag{2.48}$$

where n_i and n_e is the number density of the ions and electrons and q_i and q_e is the charge of the ions and electrons, respectively. The number density of the electrons is given by

$$n_e = n_0 \exp\left(\frac{q_e \phi}{k_B T_e}\right) \tag{2.49}$$

where ϕ is the potential due to the charged particles in the plasma and n_0 is the initial number density of the electrons, unperturbed by the potential ϕ. The relation 2.49 is called the *Boltzmann relation*, cf. [12]. Since the charge of the ions is

$$q_i = -q_e$$

the charge density from the particles in the plasma is expressed as

$$\sigma = -q_e \Big[n_i - n_0 \exp\left(\frac{q_e \phi}{k_B T_e}\right) \Big] \tag{2.50}$$

2.4.4 Plasma frequency and impact ionization

According to [12], a perturbation of the electron density in a plasma generates an electric field, which brings back the electrons towards their original position. The electrons will pass through their original position and then return towards their original position again. This back and forth oscillation of the electrons happen with high frequency and is referred to as the *electron plasma frequency*, ω_{pe}. This frequency is defined as

$$\omega_{pe} = \sqrt{\frac{n_0 \, q_e}{\epsilon_0 \, m_e}} \tag{2.51}$$

where m_e is the mass of an electron.

According to [11], *electron impact ionization* means that an electron interact with a molecule so that the molecule loses one of its electrons:

$$e^- + A \Rightarrow A^+ + 2e^- \tag{2.52}$$

where A is a neutral molecule, A^+ is a positive ion and e^- is an electron.

Chapter 3

Mathematical equations and algorithms

3.1 Known physical quantities

This section describes the known input parameters in the simulations. The output of the simulations will depend on the values of those input parameters. Important physical quantities that are known are

- the voltage of the conducting objects
- the electron current I_e from the hot cathode
- which type of gas that fills the domain and the gas density

The voltages of the conductive objects are set to the same values as the voltages of the conductive parts of the ionization gauge described in [9]. Those voltages can also be found in Table 1.1.

The electron current I_e is set to 10 mA. The electrons are represented by so called *macro particles*, a concept described in more detail in Section 3.3.2. The number of such macro particles emitted from the hot cathode is a known input parameter. The initial speed of the electrons emitted from the hot cathode is also known and is calculated from the work function, described in Section 2.4. The velocities of the electrons and their starting positions on the hot cathode are randomized.

The gas that fills the domain Ω consist of hydrogen molecules exclusively. The density of the gas is $2.47 \cdot 10^{16}$ particles per m^3 and is calculated from the ideal gas law, equation (1.4). The volume V in the ideal gas law is just the volume of the domain Ω, the pressure p was set to 10^{-4} Pa and the temperature T was 293.15 K.

The time step Δt is given a value at the beginning of the simulations. The size of Δt must however fulfill some criteria regarding stability of the solutions of the governing equations.

In this work Dirichlet boundary conditions are applied to all the electric fields at the boundaries of Ω. Dirichlet boundary conditions are described in Section 3.2.1. Reflective boundary conditions can be applied to the particles, which means that the particles will bounce back if they collide with a wall that make up a boundary of Ω. There are also absorbing boundary conditions, so that the particles are absorbed if they collide with a boundary wall. Both reflective boundary conditions and absorbing boundary conditions were used in different simulations in this work.

The ionization cross sections of the gas molecules can be calculated by the source code in Warp. One can also choose to set a value to the ionization cross sections explicitly. Both ways were tried out in different simulations carried through in this work.

There are also other parameters that can affect the outcome of the simulations and can be set before a simulation is initialized. Some examples are the size of the mesh applied to Ω and how many electrons or gas molecules that are represented by the macro particles.

3.2 Physical quantities to compute

Maxwell's equations on both integral and differential form are stated and explained in Section 2.1. Here are Maxwell's equations in differential form summarized:

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\epsilon_0} \tag{3.1a}$$

$$\nabla \cdot \boldsymbol{B} = 0 \tag{3.1b}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{3.1c}$$

$$\nabla \times \boldsymbol{B} = \mu_0(\boldsymbol{J} + \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t}) \tag{3.1d}$$

While a simulation is running, the particles keep changing their positions and velocities and these quantities have to be updated in every time step. The position $\boldsymbol{x}_p$ and the velocity $\boldsymbol{v}_p$ of a particle are related by

$$\frac{d\boldsymbol{x}_p}{dt} \equiv \boldsymbol{v}_p \tag{3.2}$$

and it is explained in Section 2.3 that the acceleration of a particle is given by

$$\frac{d\boldsymbol{v}_p}{dt} = \frac{q_p}{m_p}(\boldsymbol{E}_p + \boldsymbol{v}_p \times \boldsymbol{B}_p) \tag{3.3}$$

Maxwell's equations and equation (3.3) describe how charged particles and electromagnetic fields interact [18]. Equations (3.2) and (3.3) are referred to as the *equations of motion*, cf. [19].

Some other important equations in the electrostatic case are

$$\boldsymbol{E} = -\nabla\phi \tag{3.4}$$

and Poisson's equation

$$\triangle\phi = -\frac{\rho}{\epsilon_0} \tag{3.5}$$

They are explained in Section 2.2.2.

3.2.1 Boundary conditions

Dirichlet, Neumann and periodic boundary conditions are available in Warp. The boundary conditions are applied on the boundary of Ω, denoted $\partial\Omega$. In Warp, different types of boundary conditions can be applied on different parts of $\partial\Omega$ [7].

Dirichlet boundary conditions

Here is a definition of what a Dirichlet boundary condition is. The definition is taken from [20], but can also be found in a lot of other literature, cf. [21].

Definition: *Let*

$$L(u) = f \tag{3.6}$$

be a second order differential equation on a domain $\mathcal{D} \in \mathbb{R}^n$ with boundary $\partial\mathcal{D}$. Boundary conditions on the form

$$u(\boldsymbol{r}) = \varphi(\boldsymbol{r}), \quad \boldsymbol{r} \in \partial\mathcal{D} \tag{3.7}$$

are called Dirichlet boundary conditions.

In this work, the function u represents the electrostatic potential ϕ and the differential equation (3.6) represents Poisson's equation, given by equation (3.5). The function φ in equation (3.7) is given by a constant value on all the boundaries on the domain Ω in the simulations carried through in this work, which means that the boundary condition

$$\phi_{|\partial\Omega} = V \tag{3.8}$$

where V is a constant, is applied in the simulations. The problem of finding the potential function ϕ from Poisson's equation can be reformulated by introducing a new potential

$$\tilde{\phi} = \phi - V \tag{3.9}$$

and by using the linearity of the Laplace operator, one gets that

$$\triangle\tilde{\phi} = \triangle\phi - \underbrace{\triangle V}_{=0} = -\frac{\rho}{\epsilon_0} \tag{3.10}$$

From equation (3.4) and linearity, it holds that

$$-\nabla\tilde{\phi} = -\nabla\phi + \underbrace{\nabla V}_{=\mathbf{0}} = \boldsymbol{E} \tag{3.11}$$

hence, without knowing the value of V, the electrical field can be calculated with equations

$$\begin{cases} \triangle\tilde{\phi} = -\frac{\rho}{\epsilon_0} \\ \nabla\tilde{\phi} = -\boldsymbol{E} \end{cases}$$

where $\tilde{\phi}_{|\partial\Omega} = \phi_{|\partial\Omega} - V = 0$. These types of boundary conditions are referred to as *homogeneous Dirichlet boundary conditions.*

In Warp, Dirichlet boundary conditions are applied to the surfaces on the conducting objects automatically [8]. The hot filament, the ion collector and the anode grid of the ionization gauge are represented by conducting objects in this work.

3.3 The Particle in Cell (PIC) method

3.3.1 Motivation for using the PIC method

In this work, the domain Ω is filled with neutral gas molecules and electrons for ionizing the gas molecules, so that positively charged ions will be generated. Lets assume that Ω contains N charged particles in total. In [22] it is described that an interaction force will occur between each pair of the charged particles, where the force from the charged particle p' that affect the charged particle p is given by *Coulomb's law on vector form* as

$$\boldsymbol{F}_{pp'} = \frac{q_p q_{p'}}{4\pi\epsilon_0|\boldsymbol{x}_p - \boldsymbol{x}_{p'}|^2} \cdot \frac{\boldsymbol{x}_p - \boldsymbol{x}_{p'}}{|\boldsymbol{x}_p - \boldsymbol{x}_{p'}|} \tag{3.12}$$

In equation (3.12), q_p and $q_{p'}$ are the charges and $\boldsymbol{x}_p$ and $\boldsymbol{x}_{p'}$ are the positions of particle p and p' respectively. The term

$$\frac{\boldsymbol{x}_p - \boldsymbol{x}_{p'}}{|\boldsymbol{x}_p - \boldsymbol{x}_{p'}|}$$

is a unit vector that indicates the direction of the force coming from particle p' and affects particle p. According to *the principle of superpositions of forces*, the total force acting on particle p from the rest of the charged particles is given by the vector sum of the forces from those particles (cf. [5]). Lapenta [22] expresses this vector sum as

$$\boldsymbol{F}_p = \sum_{k=1}^{N-1} \boldsymbol{F}_{pp'_k} \tag{3.13}$$

where the force $\boldsymbol{F}_{pp'_k}$ is calculated from equation (3.12) and origins from particle p'_k. According to [23], each particle in Ω change their position by solving the ordinary differential equations (3.2) and (3.3). A numerical scheme for the advance of particle p is stated in [23] as

$$\boldsymbol{v}_p^{\text{new}} = \boldsymbol{v}_p^{\text{old}} + \frac{\Delta t}{m_p} \boldsymbol{F}_p \tag{3.14a}$$

$$\boldsymbol{x}_p^{\text{new}} = \boldsymbol{x}_p^{\text{old}} + \Delta t \, \boldsymbol{v}_p^{\text{new}} \tag{3.14b}$$

where Δt is a suitable time step. This method of calculating the forces acting upon the particles and their motions is referred to as *the particle-particle method* [23].

To calculate the force $\boldsymbol{F}_{p_k}$ for all the N charged particles, one would have to evaluate equation (3.12)

$$\sum_{k=1}^{N-1} (N-k) = \frac{N(N-1)}{2} \tag{3.15}$$

times in total. N is generally large in plasma simulations [16]. In this work particularly, the gas that is to be ionized in the simulations has a number density of size 10^{16}, and due to the size of the domain Ω in this work, about 10^{12} neutral gas molecules would have to be initialized at the beginning of the simulations. The number of created ions is an output of the simulations and according to Section 1.2, not all of the neutral gas should be ionized. But even if a small fraction of the neutral gas molecules turn into ions, N might be large. From equation (3.15), one can conclude that the number of times Coulomb's law has to be applied is of size $\mathcal{O}(N^2)$. These calculations becomes heavy for a computer to make if N is large, and an algorithm with a better scaling in N is to prefer for the simulations in this work. An example of such an algorithm is the *Particle in Cell (PIC)* method. The complexity of the PIC method is $\mathcal{O}(N + Mlog(M))$, where M is the number of grid points on the mesh [24]. M is generally much smaller than N. In this work, the great benefit with using the PIC method is that the number of computations that the computer needs to perform becomes relatively small compared to for example the particle-particle method.

According to Verboncoeur [19], the most complete description of PIC is given in [25].

3.3.2 The basics of PIC

In [16] it is written that the PIC method is commonly used for simulation of plasma. In a PIC code many *physical particles*, such as neutral gas molecules, positive ions and electrons, are represented by *macro particles.* Other names for macro particles are for example *finite-sized particles, clouds, computational particles* or *super particles*, but since the name "macro particles" is used in [26] that name will be used in this work as well.

The ratio of number of physical particles per macro particles is called the *specific weight*, cf. [16]. This means that the relationship between the macro particles and the physical particles is expressed as

$$N_{ph} = (\text{specific weight}) \cdot N_{macro} \tag{3.16}$$

where N_{ph} is the number of physical particles and N_{macro} is the number of macro particles. If the motion of N_{ph} physical particles would have been simulated as explained in Section 3.3.1, then from equation (3.12) it is clear that singularities would appear if the position of particle p and particle p' coincides, so that

$$|\boldsymbol{x}_p - \boldsymbol{x}_{p'}| \rightarrow 0$$

This issue does not need to be handled if macro particles are used instead of point charges [25]. It is explained in [22] that macro particles have a finite size and if they are far away from each other, they interact via Coulombs law in the same way as point charges would do. But if the macro particles get closer to each other, they will eventually start to overlap. In that case, the region where the particles overlap gets neutralized, meaning that the forces on the overlapping parts cancel. The bigger the overlapping part of the two particles get, the weaker the interacting force between the particles get and if they would overlap each other completely, the interacting force would be zero. This way, the singularities that occurs from Coulomb's law in equation (3.12) if point particles are used, disappear if macro particles are used instead.

The positions and the velocities of the macro particles are defined in a continuous phase space [25]. This means that the macro particles can be placed anywhere in Ω and be given any velocity. The values of the electric and the magnetic field on the other hand, are not given at every spatial position in Ω. Instead, those field values are defined only at discrete points in Ω and the field values are then interpolated from those discrete points onto the positions of the macro particles. When the interpolation is done, the movement of the macro particles can be calculated from equations (3.2) and (3.3) and after the particles have moved, particle boundary conditions are checked. Source terms needed for updating the electric or the magnetic field are then interpolated from the updated positions of the particles onto the discrete points in Ω [19]. This procedure is referred as the PIC loop, which is shown and explained in more detail in the following sections in this chapter.

In the PIC method, the field values, the particles positions and the particles velocities are defined at discrete times. The particles positions and velocities are often updated with the *leapfrog method*, which require few operations and little computational storage [19]. In the leapfrog method the quantities that are to be updated are defined on a *staggered grid* in time, which means that some quantities are defined at different times than others [21]. A standard way to implement the leapfrog method in plasma simulation codes is to use *Boris method* [16]. In Warp, a version of Boris method is implemented for moving the particles [27]. These methods and how they are implemented in Warp will be explained in Section 3.5.

3.3.3 Electromagnetic PIC

It is explained in [28] that in the electromagnetic PIC, equation (3.1c)

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t}$$

and equation (3.1d)

$$\nabla \times \boldsymbol{B} = \mu_0(\boldsymbol{J} + \epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t})$$

are solved in order to calculate the particles positions, velocities and the electromagnetic fields. The two remaining equations of Maxwell's equations, equation (3.1a) and equation (3.1b), does not have to be solved in the electromagnetic case since they are satisfied anyway, given that certain initial conditions at time $t = 0$ are satisfied. Below, those initial conditions will be given and the proofs of that equations (3.1a) and (3.1b) are satisfied will be carried through.

Theorem 1. *If Faraday's law*

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times \boldsymbol{E}$$

is satisfied all the time and provided that

$$\nabla \boldsymbol{\cdot} \boldsymbol{B}(t = 0) = 0$$

is satisfied, then

$$\nabla \boldsymbol{\cdot} \boldsymbol{B} = 0$$

is satisfied all the time.

Proof. The time derivative of the divergence of the magnetic field is

$$\frac{\partial(\nabla \boldsymbol{\cdot} \boldsymbol{B})}{\partial t} = \nabla \boldsymbol{\cdot} \frac{\partial \boldsymbol{B}}{\partial t} = \nabla \boldsymbol{\cdot} (-\nabla \times \boldsymbol{E})$$

and since the divergence of the curl of any vector field is equal to zero, it holds that

$$\frac{\partial(\nabla \boldsymbol{\cdot} \boldsymbol{B})}{\partial t} = 0 \quad \Rightarrow \quad \nabla \boldsymbol{\cdot} \boldsymbol{B} = C$$

where C is a constant in time. Since it was assumed that

$$\nabla \boldsymbol{\cdot} \boldsymbol{B}(t = 0) = 0$$

the constant C must be zero and it holds that

$$\nabla \boldsymbol{\cdot} \boldsymbol{B} = 0$$

is satisfied all the time. □

Theorem 2. *If the continuity equation* (2.22c) *and Ampère's circuital law, stated respectively as*

$$\begin{aligned}\frac{\partial\rho}{\partial t}+\nabla\cdot\boldsymbol{J} &= 0\\ \nabla\times\boldsymbol{B} &= \mu_0(\boldsymbol{J}+\epsilon_0\frac{\partial\boldsymbol{E}}{\partial t})\end{aligned}$$

are satisfied all the time, and provided that

$$\nabla\cdot\boldsymbol{E}(t=0)=\frac{\rho}{\epsilon_0}$$

is satisfied, then

$$\nabla\cdot\boldsymbol{E}=\frac{\rho}{\epsilon_0}$$

is satisfied all the time.

Proof. By taking the time derivative of the terms in Gauss law, one gets

$$\frac{\partial}{\partial t}(\nabla\cdot\boldsymbol{E}-\frac{\rho}{\epsilon_0})=\nabla\cdot(\frac{1}{\mu_0\epsilon_0}\nabla\times\boldsymbol{B}-\frac{1}{\epsilon_0}\boldsymbol{J})-\frac{1}{\epsilon_0}\frac{\partial\rho}{\partial t}=-\frac{1}{\epsilon_0}(\nabla\cdot\boldsymbol{J}+\frac{\partial\rho}{\partial t})$$

where Ampère's circuital law and that fact that the divergence of the curl of any vector is zero, was used. The right hand side of the equation above contains the continuity equation and since it is provided that the continuity equation is satisfied at all times, it holds that

$$\frac{\partial}{\partial t}(\nabla\cdot\boldsymbol{E}-\frac{\rho}{\epsilon_0})=0\quad\Rightarrow\quad\nabla\cdot\boldsymbol{E}-\frac{\rho}{\epsilon_0}=C$$

where C is a constant in time. Since it was given that

$$\nabla\cdot\boldsymbol{E}(t=0)-\frac{\rho}{\epsilon_0}=0$$

the constant C must be zero and

$$\nabla\cdot\boldsymbol{E}=\frac{\rho}{\epsilon_0}$$

is satisfied all the time. □

Due to Theorem 1 and Theorem 2, equation (3.1a) and equation (3.1b) are not solved explicitly in the electromagnetic PIC algorithm.

In Warp, an electromagnetic field solver called `EM3D` and an electrostatic field solver called `MultiGrid3D` are implemented. At an early state of this work, ionization of the gas molecules was observed if `EM3D` was used and no ionization of the gas molecules was observed with the field solver `MultiGrid3D`. Therefore, the author of this thesis first intention was to use `EM3D` for the simulations.

The problem with `EM3D` was that the electric fields did not make physical sense. The field values became extremely high at the boundaries of the domain and the electric field looked very chaotic. The author of this thesis also discovered that if the time step was set explicitly when `MultiGrid3D` was used, motion of particles and ionization of the gas molecules was observed. Therefore, the field solver `EM3D` was replaced by the field solver `MultiGrid3D`.

3.3.4 Electrostatic PIC

According to [26], the electrostatic PIC scheme goes through the following steps at each iteration in time:

1. The charge q_i of the macro particles is deposited onto the nodes of the mesh generated on Ω.

2. The electrostatic potential ϕ is computed on the nodes of the mesh, through equation (3.5).

3. The electric field $\boldsymbol{E}$ is computed on the nodes of the mesh, through equation (3.4).

4. The electric field is interpolated from the nodes of the mesh to the macro particles by a gather operation.

5. The velocities and the positions of the macro particles are updated by solving equations (3.2) and (3.3).

In other literature additional steps to the PIC loop are added, cf. [19] or [16]. Magnetic fields generated by the plasma are neglected in the electrostatic case and only externally generated magnetic fields are taken in to account. This means that the magnetic field does not have to be recalculated at each time step. If the boundary conditions for the magnetic field are defined, the magnetic field can be calculated from the field solver of the PIC code [29]. Since the electrical field changes every time the particles move, the electrical field gets recalculated at every time step [28]. Figure 3.1 illustrates how these steps keep repeating in an electrostatic PIC loop.

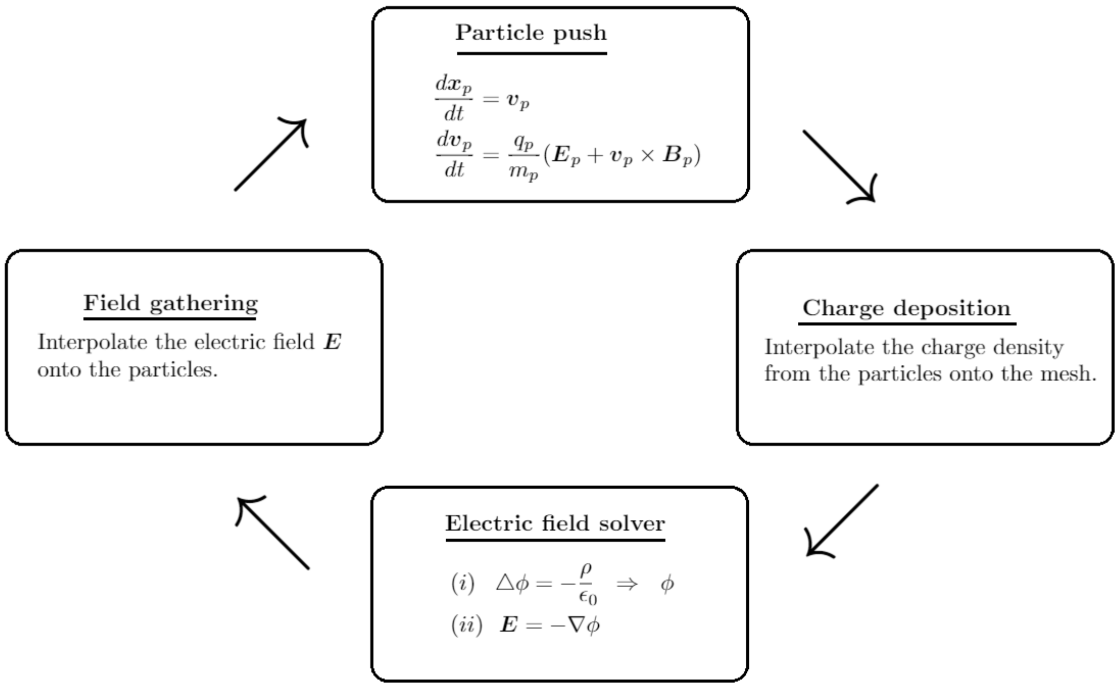

Particle push
$\frac{d\boldsymbol{x}_p}{dt} = \boldsymbol{v}_p$
$\frac{d\boldsymbol{v}_p}{dt} = \frac{q_p}{m_p}(\boldsymbol{E}_p + \boldsymbol{v}_p \times \boldsymbol{B}_p)$
Charge deposition
Interpolate the charge density from the particles onto the mesh.
Electric field solver
$(i)\ \triangle\phi = -\frac{\rho}{\epsilon_0} \Rightarrow \phi$
$(ii)\ \boldsymbol{E} = -\nabla\phi$
Field gathering
Interpolate the electric field $\boldsymbol{E}$ onto the particles.

Figure 3.1: The electrostatic PIC loop used in Warp.

Charge deposition (1.)

The charge deposition process will first be explained with the case with just one macro particle p inside of a cell, as shown in Figure 3.2. Then the case with several macro particles will be explained.

The corners of the cell in Figure 3.2 are nodes of the mesh generated on Ω and the particle p have the charge q_p. According to [26], the charge q_p is interpolated *from* the position particle p has in the cell, *onto* the nodes of the cell. The interpolation is in Warp done by *trilinear interpolation* and a description of that interpolation technique will be carried through here. The information of how this technique works comes from [30].

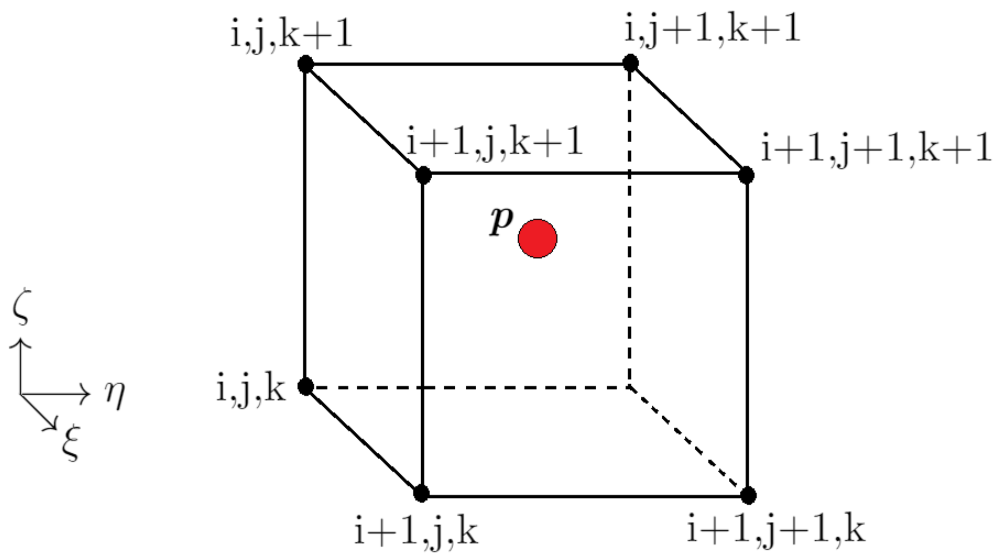


Figure 3.2: A particle in a cell.

The particle p in Figure 3.2 has a *global* position in Ω and also a *local* position in the cell it is in. Let that local position inside the cell be given by the local coordinates ξ, η and ζ, where

$$(\xi, \eta, \zeta) \in [0,1]^3 \subset \Omega$$

and where the origin of that local coordinate system coincides with node indicated by (i,j,k). The charge at node (i,j,k) that is interpolated from the charge of particle p, is denoted $q_{i,j,k}$ and given by

$$q_{i,j,k} = q_p \cdot (1-\xi)(1-\eta)(1-\zeta) \tag{3.18}$$

Equation (3.18) shows that the closer particle p gets to the node (i,j,k), the bigger part of its charge is distributed to that node. If the position of particle p would coincide with the position of node (i,j,k), all of the charge q_p would be distributed to node (i,j,k). On the contrary, if particle p would be located at any other of the nodes that belongs to the cell in Figure 3.2, the node (i,j,k) would not get any of the charge q_p.

The charge of particle p is distributed to each corner of the cell in Figure 3.2 the exact same way as the charge is distributed from particle p to node (i,j,k). This means that the charges on the remaining seven nodes are calculated by

$$\begin{aligned}
q_{i+1,j,k} &= q_p \cdot \xi\,(1-\eta)\,(1-\zeta) \\
q_{i,j+1,k} &= q_p \cdot (1-\xi)\,\eta\,(1-\zeta) \\
q_{i,j,k+1} &= q_p \cdot (1-\xi)\,(1-\eta)\,\zeta \\
q_{i+1,j+1,k} &= q_p \cdot \xi\,\eta\,(1-\zeta) \\
q_{i+1,j,k+1} &= q_p \cdot \xi\,(1-\eta)\,\zeta \\
q_{i,j+1,k+1} &= q_p \cdot (1-\xi)\,\eta\,\zeta \\
q_{i+1,j+1,k+1} &= q_p \cdot \xi\,\eta\,\zeta
\end{aligned} \tag{3.19}$$

If the cell in Figure 3.2 is not located at the boundary of the computational domain, the node (i, j, k) will be the corner of 8 different cells and these cells may contain several macro particles each. According to [25], the charges of all the macro particles that are within these 8 cells are interpolated onto node i, j, k, with the same interpolation technique used for every charged particle. As mentioned earlier, this interpolation technique is trilinear interpolation in Warp. The charges are then summed up for every mesh node in the domain and the total charge at node (i, j, k) is here denoted $q^{\text{tot}}_{i,j,k}$. This means that $q^{\text{tot}}_{i,j,k}$ is the sum of the interpolated charges from the macro particles in the 8 cells that surrounds node (i, j, k). The charge density at node (i, j, k) is calculated as

$$\rho_{i,j,k} = \frac{q^{\text{tot}}_{i,j,k}}{V_{cell}} \tag{3.20}$$

where V_{cell} is the volume of the cells in the domain. This way the charge densities are saved at all the nodes of the mesh.

Computation of electric potential (2.)

When the charge density is distributed to all the nodes of the mesh, the electrostatic potential is calculated by solving the electrostatic Poisson equation on the mesh [26]. The electrostatic Poisson equation is given by equation (3.5) and in Warp, Poisson's equation for the electrostatic potential can be solved by either a *multigrid* based solver or a *fast Fourier transform* solver [8]. In this work the multigrid solver `MultiGrid3D` is used to solve Poisson's equation. More details on how Poisson's equation is solved are given in Section 3.4.

Computation of electric fields (3.)

The electric field is calculated from equation (3.4) [16]. How the calculation is done is explained in Section 3.4.

Gather operation (4.)

The electric field is interpolated from the nodes of the mesh onto the macro particles [26]. This interpolation should be done via trilinear interpolation since that interpolation technique was used for the charge deposition [25].

Particle push (5.)

The particles positions and velocities are updated from solving the ordinary differential equations (3.2) and (3.3) [16]. This step is described in more detail in Section 3.5.

3.4 Field solvers

3.4.1 The electrostatic MultiGrid3D solver

It is mentioned in [8] that the *standard second-order finite-difference stencil* is used by the multigrid solvers for solving Poisson's equation for the electrostatic potential. In 2D the five point stencil is used and in 3D the seven point stencil is used. In Warp the user can decide how fine the mesh applied to the domain Ω will be by setting the parameters `w3d.nx`, `w3d.ny` and `w3d.nz`, which stands for the number of grid points in the x-,y- and z-direction respectively. If the number of grid points is defined in Warp, the distance between each grid point is then the same in each direction. A grid of this type is referred to as a uniform grid [31].

Discretization

Here is an explanation of what the standard second-order finite-difference stencil is and where it comes from. The electrostatic potential ϕ is the quantity that is to be computed from Poisson's equation, given by equation (3.5). The Laplacian is defined as

$$\triangle\phi \overset{def}{=} \sum_n \frac{\partial^2\phi}{\partial x_n^2} \tag{3.21}$$

and the discretization of the terms in equation (3.21) is made as

$$\frac{\partial^2\phi}{\partial x_n^2} \approx \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{h^2} \tag{3.22}$$

where h is the distance between the grid points. In two dimensions, Poisson's equation is stated as

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = -\frac{\rho}{\epsilon_0} \tag{3.23}$$

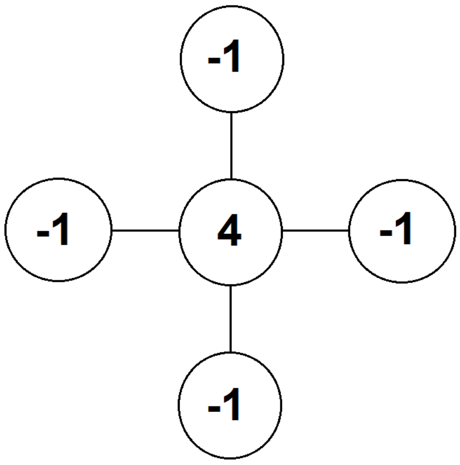


Figure 3.3: 5 point stencil.

and the finite difference approximation of equation (3.23) becomes

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{h_x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{h_y^2} = -\frac{\rho_{i,j}}{\epsilon_0} \tag{3.24}$$

where

$$\begin{aligned} i &= 0, 1, \ldots, N, N+1 \\ j &= 0, 1, \ldots, N, N+1 \end{aligned}$$

and N is the number of inner grid points on the mesh. In this work, the distances between the grid points in Ω are equal in all direction, so that

$$h_x = h_y = h_z \overset{def}{=} h.$$

In the two dimensional case, this means that equation (3.24) can be re formulated as

$$4\phi_{i,j} - \phi_{i-1,j} - \phi_{i+1,j} - \phi_{i,j-1} - \phi_{i,j+1} = h^2 \frac{\rho_{i,j}}{\epsilon_0} \tag{3.25}$$

and from equation (3.25) it is clear that information from four neighbours to node (i, j) is needed to calculate the potential on node (i, j). The name "stencil" refers to the set of nodes used it the computations (cf. [32]), and in equation (3.25) are five points involved. A visual expression of this five point stencil is given in Figure 3.3. To compute a solution to equation (3.24) the *finite difference method* can be used. A description of the finite difference method applied to a problem with homogeneous Dirichlet boundary conditions is given below. The information behind the description comes from [21].

The electrostatic potential in this work is given by a constant value on the boundaries of domain Ω. As explained in Section 3.2.1, this is a situation where Poisson's equation can be reformulated so that homogeneous Dirichlet boundary conditions are applied. If homogeneous Dirichlet boundary conditions are applied, the electrostatic potential is zero on all grid points on the boundaries of the domain, such that

$$\begin{aligned} \tilde{\phi}_{i,0} &= 0, \qquad \forall\, i = 0, 1 \ldots N+1 \\ \tilde{\phi}_{0,j} &= 0, \qquad \forall\, j = 0, 1 \ldots N+1 \end{aligned} \tag{3.26}$$

and the N^2 linear equations

$$\begin{aligned} 4\tilde{\phi}_{1,1} - \tilde{\phi}_{2,1} - \tilde{\phi}_{1,2} &= \frac{h^2}{\epsilon_0}\rho_{1,1} \\ 4\tilde{\phi}_{2,1} - \tilde{\phi}_{1,1} - \tilde{\phi}_{3,1} - \tilde{\phi}_{2,2} &= \frac{h^2}{\epsilon_0}\rho_{2,1} \\ &\vdots \\ 4\tilde{\phi}_{N,1} - \tilde{\phi}_{N-1,1} - \tilde{\phi}_{N,2} &= \frac{h^2}{\epsilon_0}\rho_{N,1} \\ 4\tilde{\phi}_{1,2} - \tilde{\phi}_{2,2} - \tilde{\phi}_{1,1} - \tilde{\phi}_{1,3} &= \frac{h^2}{\epsilon_0}\rho_{1,2} \\ 4\tilde{\phi}_{2,2} - \tilde{\phi}_{1,2} - \tilde{\phi}_{3,2} - \tilde{\phi}_{2,3} &= \frac{h^2}{\epsilon_0}\rho_{2,2} \\ &\vdots \\ &\vdots \\ 4\tilde{\phi}_{i,j} - \tilde{\phi}_{i-1,j} - \tilde{\phi}_{i+1,j} - \tilde{\phi}_{i,j-1} - \tilde{\phi}_{i,j+1} &= \frac{h^2}{\epsilon_0}\rho_{i,j} \\ &\vdots \\ &\vdots \\ 4\tilde{\phi}_{N,N} - \tilde{\phi}_{N-1,N} - \tilde{\phi}_{N,N-1} &= \frac{h^2}{\epsilon_0}\rho_{N,N} \end{aligned} \tag{3.27}$$

have to be solved. The linear equations in (3.27) can be expressed as

$$A\tilde{\boldsymbol{\phi}} = \frac{h^2}{\epsilon_0}\boldsymbol{\rho} \tag{3.28}$$

where $\tilde{\boldsymbol{\phi}}$ is a vector that contains all the sought values of $\phi_{i,j}$, $\boldsymbol{\rho}$ is a vector that contains all the solutions $\rho_{i,j}$ and A is an $N^2 \times N^2$ matrix on block tridiagonal form.

From the equations in (3.27) one can conclude that

$$A = \begin{bmatrix} 4 & -1 & & & \ddots & & & & \\ -1 & 4 & -1 & & & -1 & & & \\ 0 & -1 & 4 & -1 & & & -1 & & \\ & & \ddots & \ddots & \ddots & & & \ddots & \\ \ddots & & & -1 & 4 & 0 & & & \\ & -1 & & & 0 & 4 & -1 & & \\ & & -1 & & & \ddots & \ddots & \ddots & \\ & & & \ddots & & & & -1 & 4 \end{bmatrix}$$

The extension to discretize Poisson's equation in three dimension is straightforward. The finite difference approximation becomes

$$\begin{aligned} \frac{\phi_{i+1,j,k} - 2\phi_{i,j,k} + \phi_{i-1,j,k}}{h_x^2} + \frac{\phi_{i,j+1,k} - 2\phi_{i,j,k} + \phi_{i,j-1,k}}{h_y^2} + \\ + \frac{\phi_{i,j,k+1} - 2\phi_{i,j,k} + \phi_{i,j,k-1}}{h_z^2} = -\frac{\rho_{i,j,k}}{\epsilon_0} \end{aligned} \tag{3.29}$$

and with equal step size in each dimension one get

$$\begin{aligned} 6\phi_{i,j,k} - \phi_{i-1,j,k} - \phi_{i+1,j,k} - \phi_{i,j-1,k} \\ -\phi_{i,j+1,k} - \phi_{i,j,k-1} - \phi_{i,j,k+1} = h^2 \frac{\rho_{i,j,k}}{\epsilon_0} \end{aligned} \tag{3.30}$$

Equation (3.30) show a seven point stencil.

In the 3 dimensional case, the matrix A is of size $N^3 \times N^3$ and the time it takes more time to solve equation (3.28) in the 3 dimensional case compared to the 2 dimensional case [21]. If Poisson's equation is solved on a fine grid, then N is large and there is a large number of equations to solve. On a coarser grid, the number of equations is less. The class of methods called *multigrid* methods use the approach to switch between this finer and coarser grids. `MultiGrid3D` is a multigrid, Poisson solver in 3 dimensions [7].

3.5 Particle push

3.5.1 What the particle push is

The particle push in a PIC code is the algorithm that moves the macro particles in the domain. Governing equations for the motion of the particles are the ordinary differential equations (3.2) and (3.3), which are solved by a push algorithm. For a reminder, equations (3.2) and (3.3) are stated as

$$\frac{d\boldsymbol{x}_p}{dt} = \boldsymbol{v}_p$$

$$\frac{d\boldsymbol{v}_p}{dt} = \frac{q_p}{m_p}(\boldsymbol{E}_p + \boldsymbol{v}_p \times \boldsymbol{B}_p)$$

respectively, where q_p is the charge of the particle p and m_p is the mass of the particle p. The term

$$q_p(\boldsymbol{E}_p + \boldsymbol{v}_p \times \boldsymbol{B}_p)$$

is referred to as the Lorentz force. There may be additional forces affecting the particles, such as gravitational force and centripetal force. It depends on the problem if such forces should be included in the simulations, but generally the Lorentz force is dominating and other forces are neglected [16].

Since the electric and the magnetic fields are know only at the grid points, the field values are interpolated from the grid points to the spatial positions of the macro particles [25]. This is called a *gather operation* [16]. According to [25], it is desirable to use the same interpolation technique as used for interpolating the source terms from the particles onto the grid points. Otherwise a self force may affect the particles, which means that the particles accelerates itself. The interpolation used in Warp is described in **Charge deposition (1.)**, Section 3.3.4.

3.5.2 Considerations regarding the push algorithm

The number of macro particles in a plasma simulation may be large and a simulation may also require many time steps. The equations of motion are solved for each macro particle in the domain, so if N is the number of macro particles and n_{steps} is the number of time steps, each differential equation that governs the motion of the particles has to be solved

$$n_{steps} \cdot N$$

times. Since the governing equations of motion normally are solved many times during a simulation, it is desirable to choose an algorithm that solves the differential equations fast and at the same time do not require much computational storage while solving them. Solving equations (3.2) and (3.3) with Euler's explicit method for instance, would require that $2N$ values of the particles positions and velocities would have to be saved in each time step. The updated positions and velocities of the particles would with Euler's explicit method be calculated as

$$\boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t \cdot f_1(t^n, \boldsymbol{x}_p^n) \tag{3.31a}$$

$$\boldsymbol{v}_p^{n+1} = \boldsymbol{v}_p^n + \Delta t \cdot f_2(t^n, \boldsymbol{v}_p^n) \tag{3.31b}$$

where t^n is the time at time step n, Δt is the time difference between t^n and t^{n+1} and

$$f_1(t^n, \boldsymbol{x}_p^n) = \boldsymbol{v}_p^n \tag{3.32a}$$
$$f_2(t^n, \boldsymbol{v}_p^n) = \frac{q_p}{m_p}(\boldsymbol{E}_p^n + \boldsymbol{v}_p^n \times \boldsymbol{B}_p^n). \tag{3.32b}$$

If a higher order method would be used, more floating points operations for solving the equations of motion may be required. With the RK4 (Runge-Kutta) method for example, the velocities of the macro particles would be calculated as

$$\begin{aligned}
k_1 &= f_2(t^n, \boldsymbol{v}_p^n) \\
k_2 &= f_2(t^n + \frac{\Delta t}{2}, \boldsymbol{v}_p^n + \frac{\Delta t \cdot k_1}{2}) \\
k_3 &= f_2(t^n + \frac{\Delta t}{2}, \boldsymbol{v}_p^n + \frac{\Delta t \cdot k_2}{2}) \\
k_4 &= f_2(t^n + \Delta t, \boldsymbol{v}_p^n + \Delta t \cdot k_3) \\
\boldsymbol{v}_p^{n+1} &= \boldsymbol{v}_p^n + \frac{\Delta t}{6} \cdot \big(k_1 + 2k_2 + 2k_3 + k_4\big)
\end{aligned}$$

which is more costly compared to Euler's explicit method, with respect to the number of floating point operations. Moreover, calculating the coefficients k_1, k_2, k_3 and k_4 that are needed for updating the position and the velocities of the particles requires the vectors

$$\begin{aligned}
&\boldsymbol{x}_p^n \\
&\boldsymbol{v}_p^n, \boldsymbol{v}_p^{n+\frac{1}{2}}, \boldsymbol{v}_p^{n+1} \\
&\boldsymbol{E}_p^n, \boldsymbol{E}_p^{n+\frac{1}{2}}, \boldsymbol{E}_p^{n+1} \\
&\boldsymbol{B}_p^n, \boldsymbol{B}_p^{n+\frac{1}{2}}, \boldsymbol{B}_p^{n+1}
\end{aligned}$$

while Euler's explicit method only requires those vectors evaluated at time t^n. Hence, by using the RK4 method more information needs to be stored in the computer and since vectors evaluated at time $t^{n+\frac{1}{2}}$ and t^{n+1} are needed at time t^n, the RK4 method seems difficult to use for solving the governing equations of motion.

A commonly used method for calculating the motion of the particles in PIC codes is the *leapfrog method*, which has some similarities with Euler's explicit method. In Warp, the leapfrog method is implemented the *Boris method* is used for updating the velocity vectors of the particles [8]. These methods will be explained in the following sections and in the end some modifications of the particle push implemented in Warp will be described.

3.5.3 The leapfrog method

It is mentioned in Section 3.5.2 that the algorithm that moves the macro particles should be fast and require little computer memory at each time step. According to

[25], the leapfrog method has these properties and it has an acceptable accuracy. The method is based on the discretizations

$$\frac{dv}{dt}(t^n) \approx \frac{v^{n+\frac{1}{2}} - v^{n-\frac{1}{2}}}{\Delta t} = \frac{F_p^n}{m_p}$$
$$\frac{dx}{dt}(t^n) \approx \frac{x^{n+1} - x^n}{\Delta t} = v^{n+\frac{1}{2}} \tag{3.33}$$

of the governing equations of motion, and the numerical scheme of the leapfrog method becomes

$$\begin{cases} \boldsymbol{v}_p^{n+\frac{1}{2}} = \boldsymbol{v}_p^{n-\frac{1}{2}} + \frac{\Delta t}{m_p}\boldsymbol{F}_p^n \\ \\ \boldsymbol{x}_p^{n+1} = \boldsymbol{x}_p^n + \Delta t\, \boldsymbol{v}_p^{n+\frac{1}{2}} \end{cases} \tag{3.34}$$

The leapfrog method is of second order accuracy, while Euler's explicit method only is of first order [21]. From the equations in (3.34) it can be concluded that not many floating points operations are needed and not much data has to be saved at each time step in the leapfrog method. Figure 3.4 illustrates the staggered grid in the leapfrog scheme.

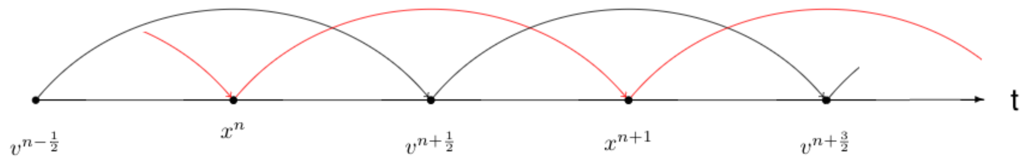


Figure 3.4: Illustration of a leapfrog scheme. The particles positions are defined at times $n\Delta t$ and the velocities of the particles are defined on times $(n + \frac{1}{2})\Delta t$.

The implementation of the leapfrog method is straightforward if the magnetic field is zero, but otherwise some additional mathematics must be used to take care of the term

$$\boldsymbol{v} \times \boldsymbol{B}$$

in the Lorentz force. The *Boris method* is the standard method for doing this in plasma simulation codes [16].

3.5.4 The Boris pusher

Motivation for using Boris method

The leapfrog scheme is shown by the equations in (3.34). To make the equations in this section tidier, the subscript "p" will be left out. If the Lorentz force $\boldsymbol{F}^n$ is expressed in terms of the electric and magnetic field, the equation for updating the velocity vector becomes

$$\boldsymbol{v}^{n+\frac{1}{2}} = \boldsymbol{v}^{n-\frac{1}{2}} + \Delta t \left(\frac{q}{m} (\boldsymbol{E}^n + \boldsymbol{v}^n \times \boldsymbol{B}^n) \right) \tag{3.35}$$

where the right hand side of equation (3.35) contains the unknown velocity vector $\boldsymbol{v}^n$. One suggestion of how to modify equation (3.35) so it can be solved by the leapfrog algorithm, is given in [33]. The approach is to first replace $\boldsymbol{v}^n$ by the average of the velocity vectors at time steps $n + \frac{1}{2}$ and $n - \frac{1}{2}$, such that

$$\boldsymbol{v}^n = \frac{\boldsymbol{v}^{n-\frac{1}{2}} + \boldsymbol{v}^{n+\frac{1}{2}}}{2} \tag{3.36}$$

By replacing the unknown velocity vector in equation (3.35) with the right hand side of equation (3.36) and making the substitution

$$\alpha \overset{def}{=} \Delta t \frac{q}{m}$$

the equation

$$\boldsymbol{v}^{n+\frac{1}{2}} = \boldsymbol{v}^{n-\frac{1}{2}} + \alpha \boldsymbol{E}^n + \frac{\alpha}{2} (\boldsymbol{v}^{n-\frac{1}{2}} \times \boldsymbol{B}^n) + \frac{\alpha}{2} (\boldsymbol{v}^{n+\frac{1}{2}} \times \boldsymbol{B}^n) \tag{3.37}$$

is obtained. In Appendix B it is stated that the cross product of two vectors can be expressed by the product of a skew-symmetric matrix and a vector, which means that one can make the substitutions

$$\begin{aligned} \boldsymbol{v}^{n+\frac{1}{2}} \times \boldsymbol{B}^n &= R_{[\times]} \boldsymbol{v}^{n+\frac{1}{2}} \\ \boldsymbol{v}^{n-\frac{1}{2}} \times \boldsymbol{B}^n &= R_{[\times]} \boldsymbol{v}^{n-\frac{1}{2}} \end{aligned} \tag{3.38}$$

where $R_{[\times]}$ is a skew-symmetric matrix as explained in Appendix B. By inserting the substitutions made in (3.38) in equation (3.37) one gets

$$\boldsymbol{v}^{n+\frac{1}{2}} = \alpha \boldsymbol{E}^n + \left(\boldsymbol{I} + \frac{\alpha}{2} R_{[\times]}\right) \boldsymbol{v}^{n-\frac{1}{2}} + \frac{\alpha}{2} R_{[\times]} \, \boldsymbol{v}^{n+\frac{1}{2}} \tag{3.39}$$

where $\boldsymbol{I}$ is the identity matrix. An updating formula for the velocity vector can then be expressed as

$$\boldsymbol{v}^{n+\frac{1}{2}} = \left(\boldsymbol{I} - \frac{\alpha}{2} R_{[\times]}\right)^{-1} \left[\alpha \boldsymbol{E}^n + \left(\boldsymbol{I} + \frac{\alpha}{2} R_{[\times]}\right) \boldsymbol{v}^{n-\frac{1}{2}} \right] \tag{3.40}$$

According to [16], implementing equation (3.40) in the leapfrog scheme will simulate the motion of the particles in a physically correct way. The energy of the macro particles will be conserved and the Larmor radius, given by equation (2.38), will be correct. This is however an implicit method that require quite a lot of calculation for each macro particle at each time step. It is mentioned in this thesis that the number of macro particles in plasma simulations may be large, and in that case an implementation of equation (3.40) in the leapfrog method will make the computations of the macro particle motion slow [16]. An alternative approach to solve equation (3.3) is to use the Boris method.

Derivation of Boris method

Boris method is a way of calculating $\boldsymbol{v}^{n+\frac{1}{2}}$ in equation (3.35). The procedure of doing that can be divided into three steps:

- Update the velocity vector $\boldsymbol{v}^{n-\frac{1}{2}}$ by adding half of the electric impulse from the electric field to it.
- Rotate the updated velocity vector in the plane perpendicular to the magnetic field.
- Add the rest of the impulse from the electric field to the updated velocity vector.

The magnetic field is interpolated from the mesh points onto the particles in the same way as described in Section 3.3.4. In this section, the material that describes the Boris method comes from Birdsall and Langdon [25] and some additional mathematics to clarify the derivation of Boris method have been added by the author of this thesis.

In the Boris method, the substitution (3.36) is made for the unknown velocity vector $\boldsymbol{v}^n$ in equation (3.35) and equation (3.3) is discretized as

$$\frac{\boldsymbol{v}^{n+\frac{1}{2}} - \boldsymbol{v}^{n-\frac{1}{2}}}{\Delta t} = \frac{q}{m}\left(\boldsymbol{E}^n + \frac{\boldsymbol{v}^{n+\frac{1}{2}} + \boldsymbol{v}^{n-\frac{1}{2}}}{2} \times \boldsymbol{B}^n\right) \tag{3.41}$$

The two new vectors

$$\boldsymbol{v}^- \overset{def}{=} \boldsymbol{v}^{n-\frac{1}{2}} + \frac{\Delta t}{2}\frac{q}{m}\boldsymbol{E}^n \tag{3.42a}$$

$$\boldsymbol{v}^+ \overset{def}{=} \boldsymbol{v}^{n+\frac{1}{2}} - \frac{\Delta t}{2}\frac{q}{m}\boldsymbol{E}^n \tag{3.42b}$$

are introduced and by expressing the vectors $\boldsymbol{v}^{n+\frac{1}{2}}$ and $\boldsymbol{v}^{n-\frac{1}{2}}$ in terms of $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$, the electrical field term $\boldsymbol{E}^n$ in equation (3.41) cancels out and one ends up with the equation

$$\frac{\boldsymbol{v}^+ - \boldsymbol{v}^-}{\Delta t} = \frac{q}{m}\frac{\boldsymbol{v}^- + \boldsymbol{v}^+}{2} \times \boldsymbol{B}^n \tag{3.43}$$

To continue the derivation, Theorem 3 is used.

Theorem 3. *The vectors* $\boldsymbol{v}^-$ *and* $\boldsymbol{v}^+$*, defined in equations* (3.42a) *and* (3.42b)*, have the same norm.*

Proof. The proof is done by taking the scalar product of (3.43) with the vector $(\boldsymbol{v}^+ + \boldsymbol{v}^-)$. The left hand side of equation (3.43) becomes

$$\begin{aligned}(\boldsymbol{v}^+ + \boldsymbol{v}^-) \cdot \frac{(\boldsymbol{v}^+ - \boldsymbol{v}^-)}{\Delta t} &= \frac{1}{\Delta t}\Big(\boldsymbol{v}^+ \cdot \boldsymbol{v}^+ + \boldsymbol{v}^- \cdot \boldsymbol{v}^+ - \boldsymbol{v}^+ \cdot \boldsymbol{v}^- - \boldsymbol{v}^- \cdot \boldsymbol{v}^-\Big) = \\ &= \frac{1}{\Delta t}\Big(\boldsymbol{v}^+ \cdot \boldsymbol{v}^+ - \boldsymbol{v}^- \cdot \boldsymbol{v}^-\Big)\end{aligned} \tag{3.44}$$

According to the properties (B.13) and (B.14) in Appendix B, the right hand side of equation (3.43) can be rewritten as

$$\frac{q}{m}\frac{\boldsymbol{v}^- + \boldsymbol{v}^+}{2} \times \boldsymbol{B}^n = -\frac{q}{2m}\boldsymbol{B}^n \times \boldsymbol{v}^+ - \frac{q}{2m}\boldsymbol{B}^n \times \boldsymbol{v}^- \tag{3.45}$$

By introducing

$$\alpha \overset{def}{=} -\frac{q}{2m} \tag{3.46}$$

and taking the scalar product of the right hand side of equation (3.43) with the vector $(\boldsymbol{v}^+ + \boldsymbol{v}^-)$, one gets

$$\begin{aligned} \alpha(\boldsymbol{v}^+ + \boldsymbol{v}^-) \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^+ + \boldsymbol{B}^n \times \boldsymbol{v}^-) = \alpha\big(&\boldsymbol{v}^+ \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^+) + \\ &+ \boldsymbol{v}^- \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^+) + \boldsymbol{v}^+ \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^-) + \boldsymbol{v}^- \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^-)\big) \end{aligned} \tag{3.47}$$

From the definition of the cross product, it holds that

$$\boldsymbol{v}^+ \perp (\boldsymbol{B}^n \times \boldsymbol{v}^+) \;\;\Rightarrow\;\; \boldsymbol{v}^+ \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^+) = 0 \tag{3.48a}$$

$$\boldsymbol{v}^- \perp (\boldsymbol{B}^n \times \boldsymbol{v}^-) \;\;\Rightarrow\;\; \boldsymbol{v}^- \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^-) = 0 \tag{3.48b}$$

and the right hand side of equation (3.47) reduces to

$$\alpha\big(\boldsymbol{v}^- \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^+) + \boldsymbol{v}^+ \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^-)\big) \tag{3.49}$$

By using the properties (B.12) and (B.13) in Appendix B, one gets

$$\boldsymbol{v}^+ \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^-) = -\boldsymbol{v}^- \cdot (\boldsymbol{B}^n \times \boldsymbol{v}^+) \tag{3.50}$$

which shows that the term (3.49) is zero. Hence, by taking the scalar product of $(\boldsymbol{v}^+ + \boldsymbol{v}^-)$ with the terms in equation (3.43), one ends up with

$$\frac{1}{\Delta t}\Big(\boldsymbol{v}^+ \cdot \boldsymbol{v}^+ - \boldsymbol{v}^- \cdot \boldsymbol{v}^-\Big) = 0 \;\;\Rightarrow\;\; \boldsymbol{v}^+ \cdot \boldsymbol{v}^+ = \boldsymbol{v}^- \cdot \boldsymbol{v}^- \tag{3.51}$$

which means that $|\boldsymbol{v}^+| = |\boldsymbol{v}^-|$. $\square$

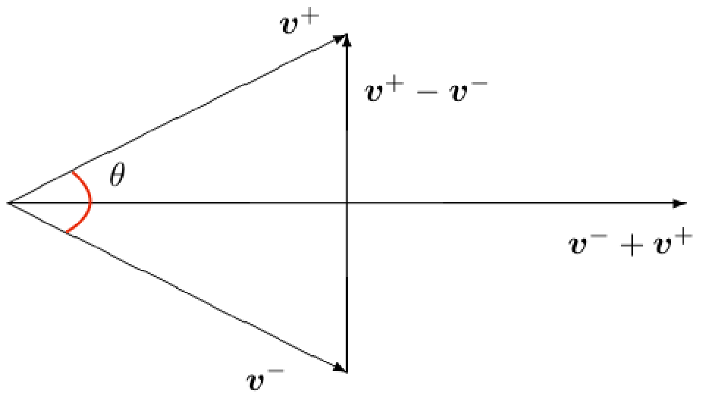

$\boldsymbol{v}^+$
$\boldsymbol{v}^+ - \boldsymbol{v}^-$
θ
$\boldsymbol{v}^- + \boldsymbol{v}^+$
$\boldsymbol{v}^-$

Figure 3.5: The vectors $\boldsymbol{v}^-$, $\boldsymbol{v}^+$ and the angle θ between them.

From Theorem 3 it is concluded that $\boldsymbol{v}^+$ is just a rotation of the vector $\boldsymbol{v}^-$ in the plane spanned by $\boldsymbol{v}^+$ and $\boldsymbol{v}^-$. Let the angle between the vectors $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$ be θ, as shown in Figure 3.5. Equation (3.43) implies that

$$\frac{|\boldsymbol{v}^+ - \boldsymbol{v}^-|}{\Delta t} = \frac{q}{2m}|(\boldsymbol{v}^- + \boldsymbol{v}^+) \times \boldsymbol{B}^n| \tag{3.52}$$

and from Figure 3.5 and equation (3.52) it can be concluded that

$$\tan\left(\frac{\theta}{2}\right) = \frac{|\boldsymbol{v}^+ - \boldsymbol{v}^-|}{|\boldsymbol{v}^+ + \boldsymbol{v}^-|} = \frac{q\Delta t}{2m}\frac{|(\boldsymbol{v}^- + \boldsymbol{v}^+) \times \boldsymbol{B}^n|}{|\boldsymbol{v}^- + \boldsymbol{v}^+|} \tag{3.53}$$

From how a cross product is defined, it holds that

$$|(\boldsymbol{v}^- + \boldsymbol{v}^+) \times \boldsymbol{B}^n| = |(\boldsymbol{v}^- + \boldsymbol{v}^+)||\boldsymbol{B}^n|\sin(\varphi) \tag{3.54}$$

where φ is the angle between the vectors $(\boldsymbol{v}^- + \boldsymbol{v}^+)$ and $\boldsymbol{B}^n$ in the plane spanned by the vectors $(\boldsymbol{v}^- + \boldsymbol{v}^+)$ and $\boldsymbol{B}^n$. Since v^+ is just a rotation of v^- in the plane perpendicular to the magnetic field $\boldsymbol{B}^n$, $\sin(\varphi) = 1$ in equation (3.54) and equation (3.53) becomes

$$\tan\left(\frac{\theta}{2}\right) = \frac{q\Delta t}{2m}|\boldsymbol{B}^n| = \frac{q\Delta t B}{2m} \tag{3.55}$$

where $B = |\boldsymbol{B}^n|$.

Next step in Boris algorithm is to add a new vector to the vector $\boldsymbol{v}^-$. This new vector shall be perpendicular to both $\boldsymbol{v}^-$ and $\boldsymbol{B}^n$, which means that it can be expressed as

$$\boldsymbol{v}^- \times \boldsymbol{t} \tag{3.56}$$

where $\boldsymbol{t}$ is a vector parallel to $\boldsymbol{B}^n$. The length of the cross product in (3.56) shall be such that it just reaches the line that bisects the angle θ, if the cross product in

(3.56) starts from the end of the vector $\boldsymbol{v}^-$. Figure 3.6 show how the cross product in (3.56) is placed.

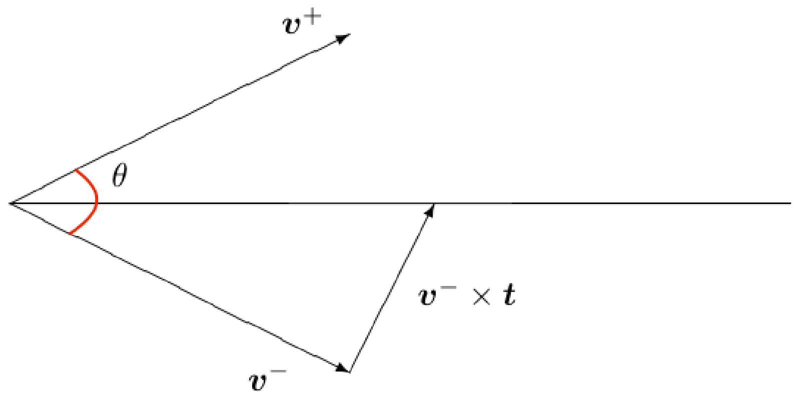


Figure 3.6: The vector $\boldsymbol{v}^- \times \boldsymbol{t}$ reaches the line that intersect the angle θ.

The length of the vector $\boldsymbol{t}$ can be calculated geometrically from Figure 3.6, as

$$\tan\left(\frac{\theta}{2}\right) = \frac{|\boldsymbol{t}|}{|\boldsymbol{v}^-|} = \frac{q\Delta t B}{2m} \tag{3.57}$$

where equation (3.55) is used in the last step. This gives that

$$|\boldsymbol{t}| = |\boldsymbol{v}^-|\frac{q\Delta t B}{2m} \tag{3.58}$$

and since $\boldsymbol{v}^-$ and $\boldsymbol{t}$ are perpendicular,

$$|\boldsymbol{v}^- \times \boldsymbol{t}| = |\boldsymbol{v}^-| \cdot \frac{q\Delta t B}{2m} \tag{3.59}$$

Equation (3.59) holds if $\boldsymbol{t}$ is chosen as

$$\boldsymbol{t} = \frac{q}{m}\frac{\Delta t}{2}\boldsymbol{B} \tag{3.60}$$

The vector that is obtained after adding the cross product in (3.56) is defined as

$$\boldsymbol{v}' \overset{def}{=} \boldsymbol{v}^- + \boldsymbol{v}^- \times \boldsymbol{t} \tag{3.61}$$

and $\boldsymbol{v}'$ follows the line that bisects θ in Figure 3.6. Next step is to find the vector $\boldsymbol{v}' \times \boldsymbol{s}$ that is added to $\boldsymbol{v}^-$ so that the rotated vector $\boldsymbol{v}^+$ is obtained. The vector $\boldsymbol{v}' \times \boldsymbol{s}$ is shown in Figure 3.7 and since this cross product lies in the plane spanned by $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$, the vector $\boldsymbol{s}$ has to be parallel to $\boldsymbol{B}$. If $\boldsymbol{s}$ is chosen as

$$\boldsymbol{s} = \frac{2\boldsymbol{t}}{1 + |\boldsymbol{t}|^2} \tag{3.62}$$

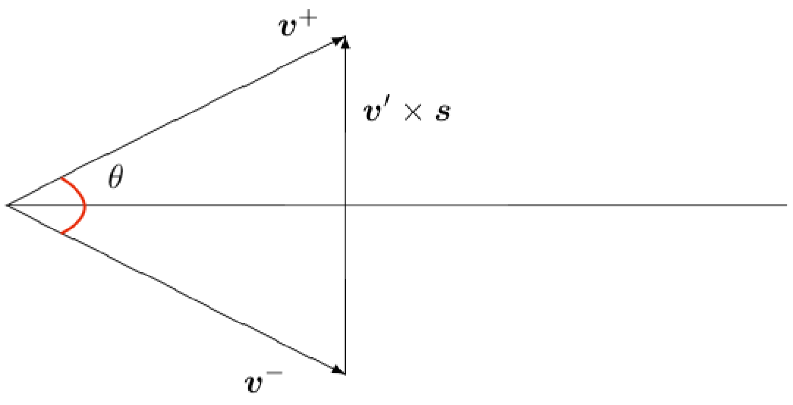


Figure 3.7: The vectors $\boldsymbol{v}^-$, $\boldsymbol{v}^+$ and the angle θ between them.

the vector $\boldsymbol{v}^+$ can be computed as

$$\boldsymbol{v}^+ = \boldsymbol{v}^- + \boldsymbol{v}' \times \boldsymbol{s} \tag{3.63}$$

The last step is to compute the velocity vector at time step $n + \frac{1}{2}$, according to equation (3.42b):

$$\boldsymbol{v}^{n+\frac{1}{2}} = \boldsymbol{v}^+ + \frac{\Delta t}{2}\frac{q}{m}\boldsymbol{E}^n \tag{3.64}$$

A summary of Boris method is expressed in the pseudo code below.

```
for $n = 1, 2, 3...$ do
    Data: $\boldsymbol{E}^n, \boldsymbol{B}^n, \boldsymbol{v}^{n-\frac{1}{2}}$

    $\boldsymbol{t} = \frac{q\Delta t}{2m}\boldsymbol{B}^n$
    $\boldsymbol{s} = \frac{2\boldsymbol{t}}{1+|\boldsymbol{t}|^2}$

    $\boldsymbol{v}^- = \boldsymbol{v}^{n-\frac{1}{2}} + \frac{q\Delta t}{2m}\boldsymbol{E}^n$
    $\boldsymbol{v}' = \boldsymbol{v}^- + \boldsymbol{v}^- \times \boldsymbol{t}$
    $\boldsymbol{v}^+ = \boldsymbol{v}^- + \boldsymbol{v}' \times \boldsymbol{s}$
    $\boldsymbol{v}^{n+\frac{1}{2}} = \boldsymbol{v}^+ + \frac{\Delta t}{2}\frac{q}{m}\boldsymbol{E}^n$
end
```

3.5.5 Particle push in Warp

The modified Boris pusher used in Warp

According to [8], the default particle push in Warp is built on Boris method. If the time step Δt is large compared to the gyroperiod τ_g given by equation (2.40), the gyration radius (equation (2.38)) computed by Boris method will be too large [34].

In Warp, the problem with the spuriously large gyration radius is solved by an updated version of the Boris algorithm. The velocity of a particle is computed as

$$\boldsymbol{v}_{eff} = \boldsymbol{b}(\boldsymbol{b} \cdot \boldsymbol{v}_B) + \alpha \boldsymbol{v}_{B,\perp} + (1 - \alpha)\boldsymbol{v}_d \tag{3.65}$$

where $\boldsymbol{v}_B$ is the velocity calculated by Boris method, $\boldsymbol{b}$ is a unit vector that points in the same direction as the magnetic field and $\boldsymbol{v}_{B,\perp}$ is the velocity of the particle in the plane perpendicular to the magnetic field, denoted as $\boldsymbol{v}_\perp$ in Section 2.3. α is an interpolation coefficient and if α is set to

$$\alpha = \frac{1}{\sqrt{1 + (\frac{\omega_g \Delta t}{2})^2}} \tag{3.66}$$

both the correct drift velocity and the correct gyration radius is calculated [27]. In equation (3.66) Δt is the time step and ω_g is the gyro frequency, explained by equation (2.39).

It can be concluded from equations (2.39) and (2.40) that in regions where the magnetic field is strong, the gyro period gets small. In regions where the magnetic field gets strong enough, the gyro period gets much shorter than other time scales of interest and a very small time step is needed for the simulation of the electron movement. According to [27], in such regions it is sufficient to approximate the particle movement in the plane perpendicular to the magnetic field with the $E \times B$ drift. The modified particle mover implemented in Warp can be used if such simplifications are needed [27].

Output diagnostics

In [27] it is written that since the velocities in (3.34) are evaluated at a staggered grid with respect to the positions, it is not possible to directly retrieve particle data, inject new particles or change the time step dynamically. To solve this problem, an algorithm that combines leapfrog steps with periodic special steps has been implemented in Warp. Most of the time during a simulation the leapfrog steps are given by equation (3.34), but to get synchronized data the velocity is updated a half time step every now and then. To start the leapfrog sequence given in equation (3.34), or to restart it after the positions and the velocities have been given at the same time step, the velocities are updated with a half time step in a first iteration.

3.6 Calculation of ionization

A script called `lpa_script.py` is one of several example scripts available in Warp. The script `lpa_script.py` was studied and used for learning how to set up simulations in Warp. This script was changed little by little so that in the end it would simulate the ionization gauge described in Section 1.4. `lpa_script.py` was chosen as a starting point because it simulated a mixture of neutral molecules, positive ions and electrons. These kinds of particles also had to be included in the script for simulating the ionization gauge.

After making some changes in `lpa_script.py`, ionization of the hydrogen molecules was observed. However, `lpa_script.py` is a script for simulating laser-wakefield acceleration and if atoms or molecules interact with near infrared strong laser pulses, *tunnel ionization* can occur. Through a Python debugger it was found that the observed ionization was computed from the script `tunnel_ionization.py`. The result of the debugging is seen in Appendix D. When tunnel ionization happened, the specific weight for the electrons was very high. After analytically having calculated the specific weight so that 10 mA would be emitted from the hot cathode, ionization was no longer observed.

In the simulations of the ionization gauge, *impact ionization* is expected to happen, not tunnel ionization. Impact ionization is described in Section 2.4.4. There is a Python script called `ionization.py` which contains a class called `Ionization`. This class is made for generating particles from impact ionization and seem suited for calculating the ionization of the hydrogen gas.

To use the class `Ionization`, first an object of the class `Ionization` has to be created. Then the function `add_ionization(...)` which is implemented in the class `Ionization` needs to be called once when the simulation is set up. The user has to give the function `add_ionization(...)` the arguments *target species*, *emitted species* and *incident species*. The target species are the neutral molecules that can become positive ions, the emitted species are the positive ions and the electrons that are added to the simulation if ionization occur. The incident species are the particles that can collide with the target species so that positive ions are created. In this work, the hydrogen molecules are a target specie, the protons and electrons are the emitted species and electrons is the incident specie.

During the simulation, the function `generate(...)` in the class `Ionization` is called at each iteration. In that function a one dimensional array called `ncol` is generated, which have the same size as the number of macro particles representing the incident species. Every element n in `ncol` is a floating point number, where

$$\texttt{ncol[n]} \propto \mathrm{dp[n]} \cdot \sigma \cdot v_i[\mathrm{n}] \cdot \Delta t \tag{3.67}$$

In (3.67), dp[n] is the partial pressure at the position of incident specie number n, σ is the ionization cross section that can be provided by the user, v_i[n] is the velocity of incident specie number n and Δt is the time step. The partial pressure dp[n] is defined at the grid points of the mesh and depend on how many target particles that are in the vicinity of the grid points. dp[n] is interpolated from the grid points of the mesh onto the position of incident specie number n.

A higher the value of `ncol[n]` gives a higher probability that incident specie number n generates an ion. If that happens, the emitted species are generated at approximately the same position the incident specie had when the ionization was generated. Unfortunately, the class `Ionization` was not completely implemented when the work with this thesis began. The problem with the class `Ionization` is if a target specie turns in to a positive ion, that target specie is not deleted. So by using the class `Ionization` to simulate ionization would increase the number of macro particles every time a positive ion is created and therefore also increase the pressure in the domain. The developers of Warp have now been informed about this problem and the author of this thesis added some code to the class `Ionization` that deletes the right number of target species if ionization occur.

Here, the code added to `Ionization` to delete the target species will be described. The positions of the incident species that generate the ions are saved in arrays in the class `Ionization`. Every particle in Warp has a unique identification number and the already implemented function `getssn(...)` return those identification numbers. The user can give x-, y- and z-values to `getssn(...)`, in which case the function returns only the identification numbers of the macro particles that are within those x-, y,- and z-values given by the user. It is also important to give an argument that states for which particle group the user wants the identification numbers.

In the code added in `Ionization`, `getssn(...)` returns the identification numbers of the macro particles within a region with the size of 8 mesh cells around the position where the incident specie generated the ion. If there are several target species within that region, one of those target species are removed randomly. If there are no target species in that region, a target specie is removed randomly from the computational domain.

In Warp, the particle data for all the macro particles is kept in a one dimensional array, here denoted `particleData[]`. To find out which position the target specie that is to be removed has in `particleData[]`, the function `selectparticles(...)` that already is implemented in Warp, is used. `selectparticles(...)` takes the identification numbers of the macro particles as input arguments and return the positions those macro particles has in the `particleData[]`. When those positions in `particleData[]` are known, the program executes the code

```
put(ipg.gaminv,array_placement,0)                (3.68)
```

where `array_placement` is an array with the particles placements in `particleData[]` and `gaminv` is a parameter that every macro particle have. If `gaminv` is set to zero, that particle will be removed from the simulation. The line of code in (3.68) puts `gaminv` to zero for the particles with indices in `particleData[]` given by `array_placement`.

An other way to simulate ionization is to increase the specific weight of the electrons to a value that is several orders of magnitudes greater than the value of the specific weight computed originally. The number of generated ions in this case is also dependent of the specific weight of the hydrogen molecules.

Chapter 4

Simulation

4.1 Simulation setup

4.1.1 Conducting objects

Warp contains the script `generateconductors.py`, where different classes are implemented from which geometrical objects can be created. Those objects are *conducing objects* and in this work they were used to model the solid parts of the ionization gauge that had a potential applied to them. There are several geometries of the conducting objects available in Warp and the user can define some characteristic parameters such as their length, height and potential. In this work, the ion collector and modulators were created by `ZCylinder`, the hot cathode was made of two conducting objects of type `Cylinder` and the anode grid was made by several conducting objects of type `ZTorus`.

Conducting objects should be generated in the Python script after a field solver have been registered with the command `registersolver(...)` and before the call `generate(...)` is made. To make the conducting objects absorb particles, a *particle scraper* has to be created. A particle scraper was created for every conducting object in this work.

As mentioned in Section 3.2.1, for the electric field Dirichlet boundary conditions are applied to the boundaries of the conducting objects. More information about conducting objects can be found in [7].

4.1.2 Classes created for this work

To be able to make the simulations and retrieve results from the simulations, some classes were written and used in this work. Here is a summary of those classes and some description of what they do.

`CreateConductingObjects`

The class `CreateConductingObjects` contains all the conducting objects needed for the simulation of the ionization gauge. Some parameters can be passed through from the main script, for example their placement in the domain, the voltage, the number of grid toruses, length and radius. A consistency check is implemented in the function that generates the anode grid, such that no toruses are overlapping.

The particle scraper needed for absorbing particles on the conducting objects, is initialized in the constructor of `CreateConductingObjects`. This scraper is used by all the conducting objects of the class `CreateConductingObjects`.

`WriteToFileClass`

`WriteToFileClass` is a separate class for handling all the writing of data to separate files and for printing relevant information to the screen during a simulation. For example, functions for writing the positions of the particles, the velocities of the particles and the strength of the electric and the magnetic fields at the positions of the particles, are implemented in `WriteToFileClass`. Those functions build on the in Warp already implemented functions

```
getx(), gety(), getz()
getvx(), getvy(), getvz()
getex(), getey(), getez()
```

which returns the the particles positions, velocities and electric and the elctric field at the positions of the particles, respectively. Information about those functions can be found in [26].

An alternative to retrieve the strength of the electric field at the positions of the particles, is to use the function `getselfe()`. `getselfe()` returns the electric field values at the grid points in the form of an array that looks like

$$[E, n_x, n_y, n_z] \tag{4.1}$$

The parameter E in (4.1) is an array with three elements. The first element contains all the values of the electric field in the x-direction and the second and the third element contain the values of the electric field in the y- and z-direction respectively. The arrays n_x, n_y and n_z are then used to specify at which grid points the user wants to look at the electric field values. For example would

$$[E, n_x, n_y, n_z] = [0, 10, 5, 19] \tag{4.2}$$

return the x-compontent of the electric field at the 11th grid point in the x-direction, the 6th grid point in the y-direction and the 20th grid point in the z-direction (the first grid point corresponds to the value 0).

Functions to visualize the electric field values on the xy-, xz- and yz-planes were implemented in `WriteToFileClass`. The function for plotting the electric field on the xy-plane for example, the user defines at which height in the z-direction the xy-plane shall be. Then two for-loops are used to loop over all the grid points in that xy-plane and write the spatial positions of the grid points and the values of the electric field in the x-, y- and z-direction to a file. In Figure 4.1 those uniformly distributed grid point are visualized in the xy-plane and in the xz-plane.

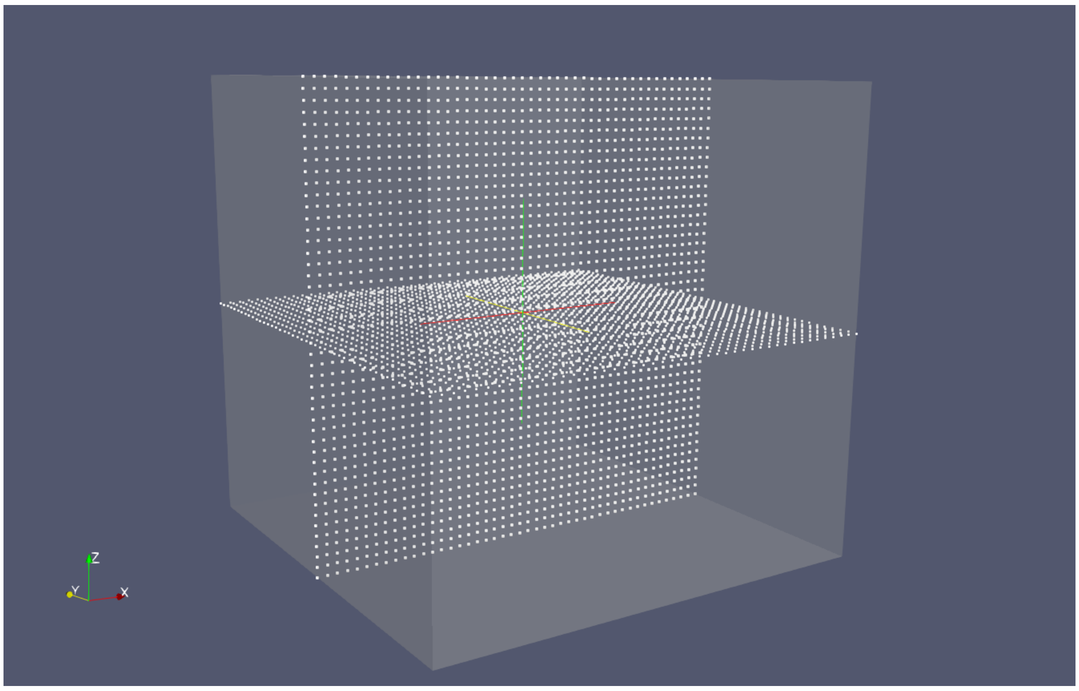


Figure 4.1: Grid points placed uniformly over the domain. The electrical field can be plotted on those points.

`PlasmaInjectorModified`

In the script `lpa_script.py` a class `PlasmaInjector` is used to inject particles. The positions or velocities of the particles can not be given as arguments to `PlasmaInjector`. Instead, particles are placed with equal distance from each other in a subsection of the domain defined by the user.

An updated version of `PlasmaInjector` was created for injecting electrons from the hot cathode. This new class was called `PlasmaInjectorPosition` and the positions and the velocities of the particles had to be given as input arguments from the user. How those positions and velocities are calculated is described in Section 4.3.

When the simulations were run in parallel, every processor began to inject electrons. If for example 30 processors were used for the computations, which was usually the case, then 30 electrons would be injected into the domain at every iteration instead of 1. First an attempt to make only one processor inject the electrons was carried through. Then the variable `lallindomain` was found, which took care of the problems with the parallelization. `lallindomain` was by default set to `True`. When `lallindomain` was set to `False` instead, the processors would not add electrons outside of their sub-domain. This way, only the processor or processors that made the calculations of the part of the domain Ω where the hot cathode was placed, would also add the electrons. The domain Ω was distributed among the processors so that the hot cathode did not intersect any boarders of the sub-domains.

Further investigation showed that the class `PlasmaInjector` used the function

```
addparticles(...)
```

for injecting the plasma into the domain. `addparticles` is a function that belongs to the class `Species`, implemented in Warp. In contrast to `PlasmaInjector`, `addparticles` can take the boolean `lallindomain` a an argument. Hence, the problem with the parallelization was already taken care of if the function `addparticles` was used with the argument `lallindomain=False`.

Finally, a new version of `PlasmaInjector` was made. That class was called `PlasmaInjectorModified` and it was used only for injecting the hydrogen molecules at the very first iteration of the simulation. The only difference from the original `PlasmaInjector` was that `lallindomain=False`. This class is used to fill up the domain Ω with hydrogen molecules distributed with an equal distance away from each other. This class works in the parallel version.

4.2 Tests of different parts of Warp

To evaluate the functionality of Warp, different parts of the software were tested individually. Tested parts and the outcomes of the tests are given in the subsections below.

4.2.1 Absorption of particles

By default, the particles that get absorbed by the conductive objects are deleted by the Warp software. But by setting the variable `lsavelostpart=True`, the absorbed particles are saved and this was used for controlling that the particles actually got absorbed by the conductive objects. By filling the domain Ω with electrons and installing the hot cathodes, the positions where those electrons were absorbed on the cathodes could be plotted. Those positions are shown in Figure 4.2, where the shape of the cathodes appear. This indicates that the particles scraper is working properly.

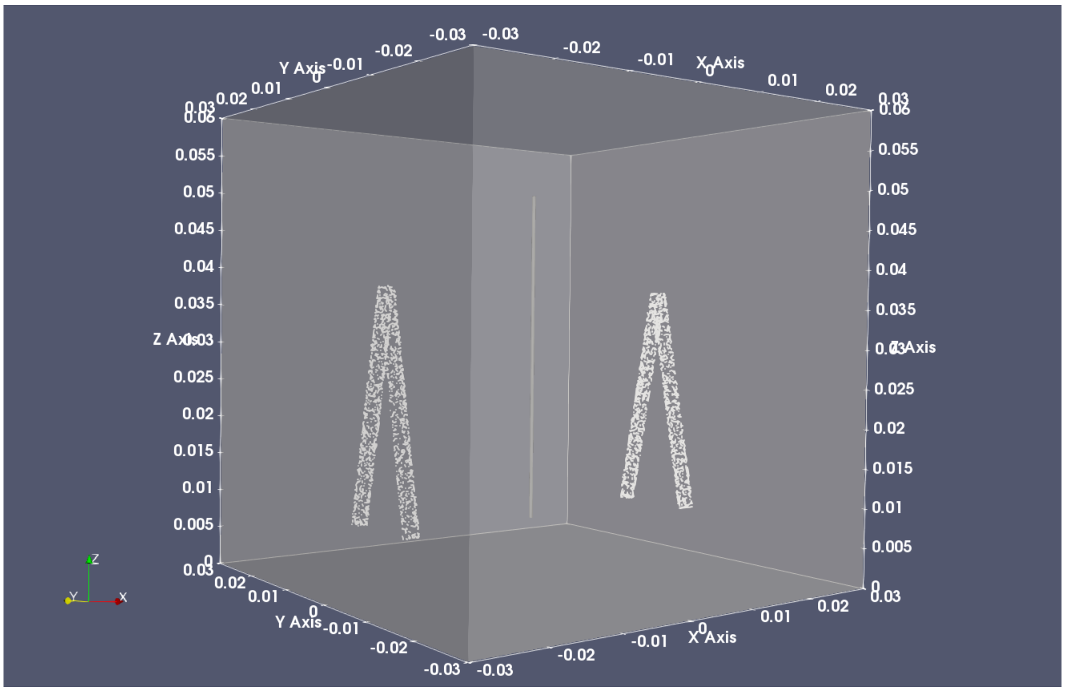


Figure 4.2: The small white dots show where the electrons are absorbed. The radiuses of the both cathodes were temporary set to 1 millimeter in this experiment.

4.2.2 Field solvers

Electrical fields from different parts of the ionization gauge were generated. Figure 4.3 and Figure 4.4 show the electric field from the anode grid only. The field is plotted on the equidistant mesh grid described in Section 4.1.2, under `WriteToFileClass`. Dirichlet boundary conditions are applied on every boundary of the domain. The reason why the field is perpendicular to the boundaries is explained in Section 3.2.1 and by equation (2.27).

The electrical field from the hot, emitting cathode and the ion collector was also computed. The result is shown in Figure 4.5.

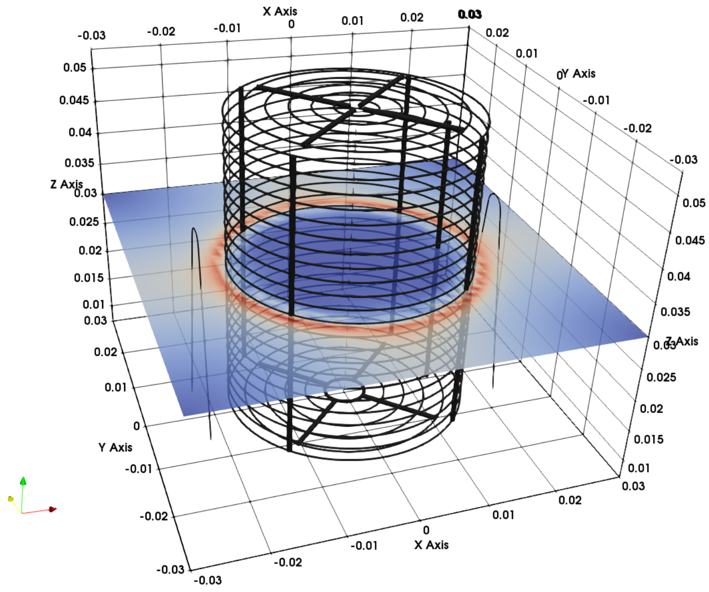


Figure 4.3: The magnitude of the electric field generated from the anode grid.

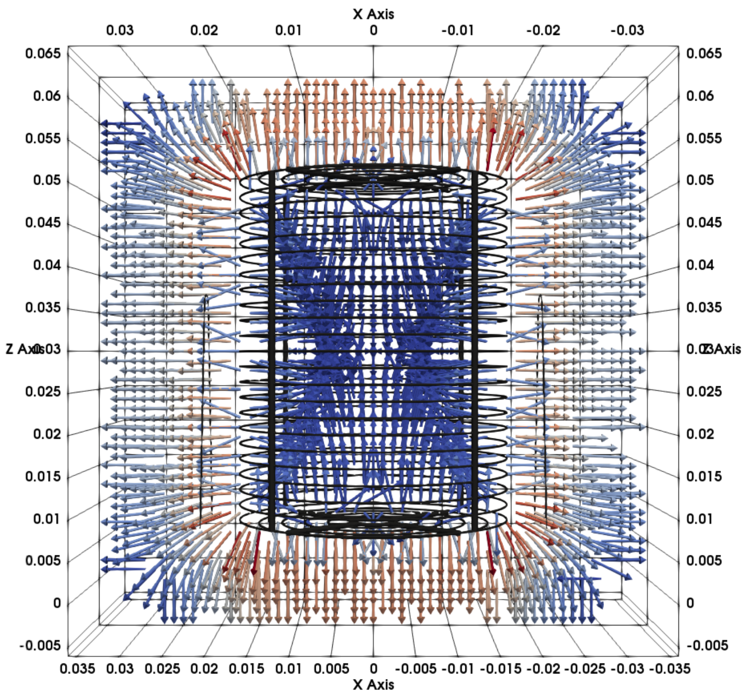


Figure 4.4: The electric field represented by vectors in the xz-plane. The color of the vectors is the magnitude of the electric field.

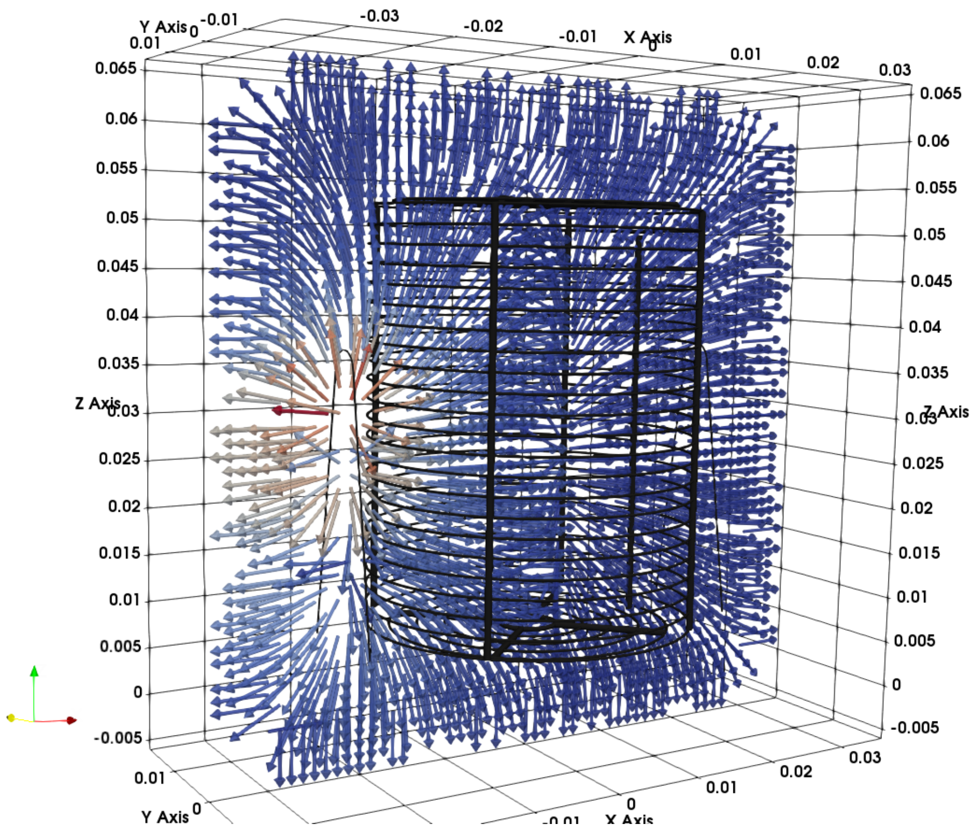


Figure 4.5: Electric field generated by the cathode and the ion collector.

4.2.3 Particle movement and the ion collector

Four positively charged hydrogen ions were placed inside of the anode grid with zero velocity initially. Their starting positions are given in Table 4.1 and Figure 4.6 and Figure 4.7 show how the ions move as time evolve. The ions are represented by the red spheres and the arrows placed on each ion show their speed and their velocity projected on the xy-plane and xz-plane respectively. The color of the arrows shows the speed of the ions. In this simulation the electric field comes from the anode grid and the ion collector. The radius of the ion collector in the figures is drawn bigger compared to its radius in the simulation.

The ions did not get absorbed by the ion collector in this simulation. Other simulations were made where the ion collector had a bigger radius and in those simulations the ions got absorbed by the ion collector.

Table 4.1: Starting positions of the ions

Ion / coordinate	x	y	z
1	-0.005	-0.005	0.01
2	0.005	0.005	0.02
3	-0.005	0.005	0.03
4	0.005	-0.005	0.04

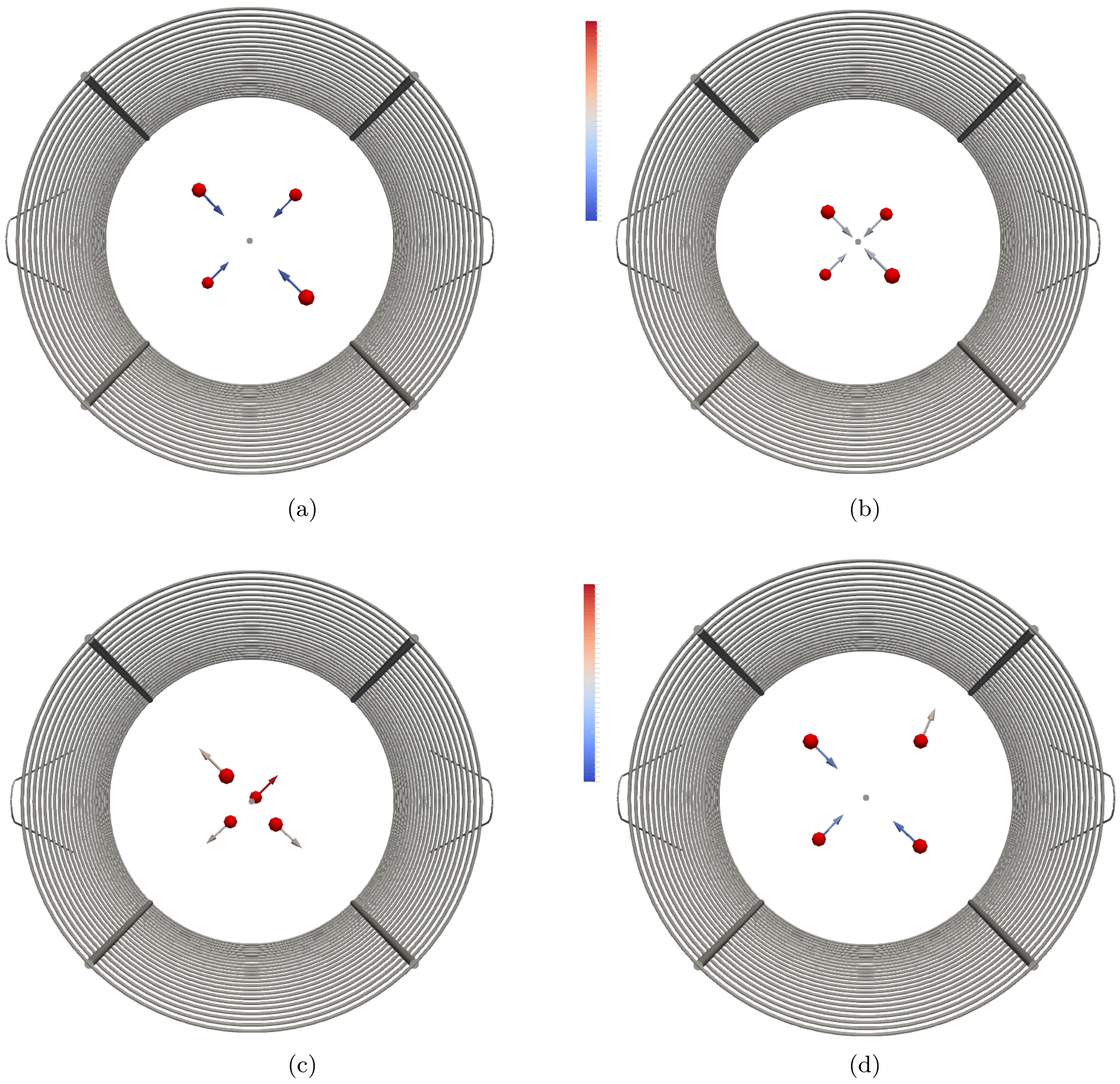


Figure 4.6: The ionization gauge from above. The color scale indicates the speed of the ions. (a) The starting positions of the ions. (b) All ions have moved closer to the ion collector. (c) The ions missed the collector and are moving away from it. (d) The ions start moving towards the collector again.

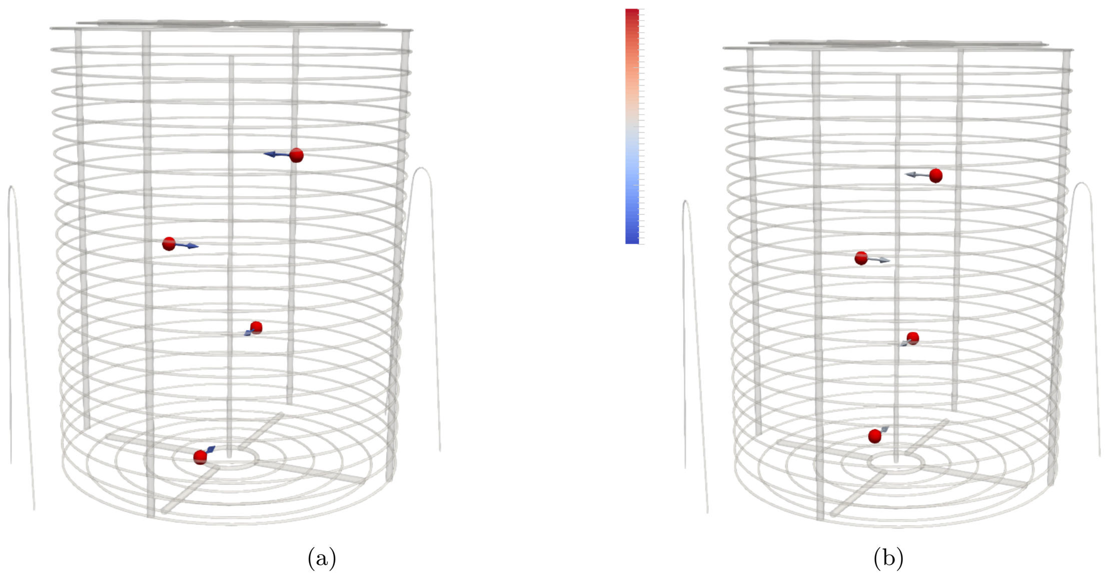
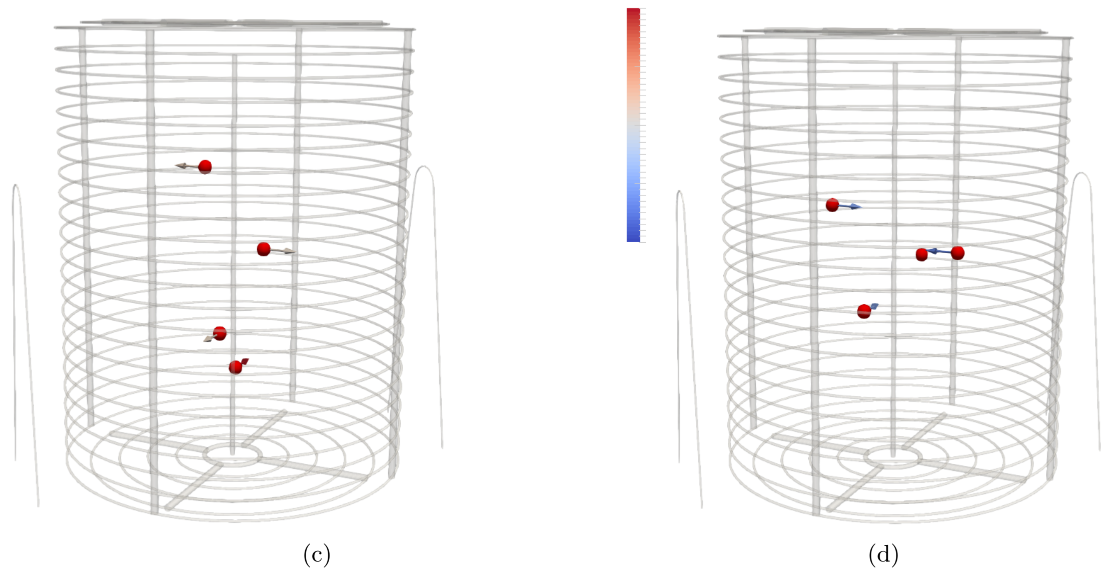
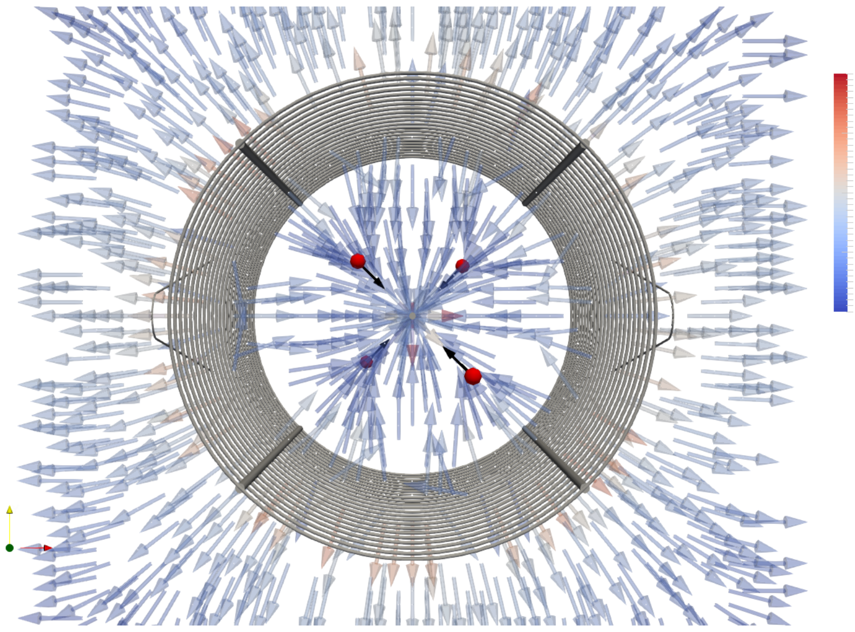
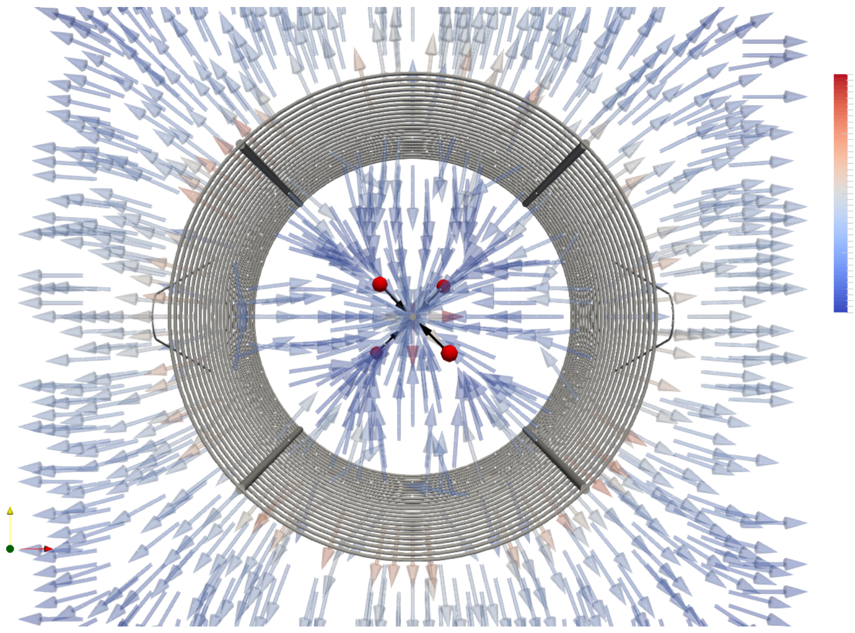



Figure 4.7: The same results as in Figure 4.6, but seen from the side of the ion collector.

Figure 4.8 and Figure 4.9 show that the ions seem to follow the electric field lines. The ions move towards the ion collector, which is the purpose of the construction of the ionization gauge.

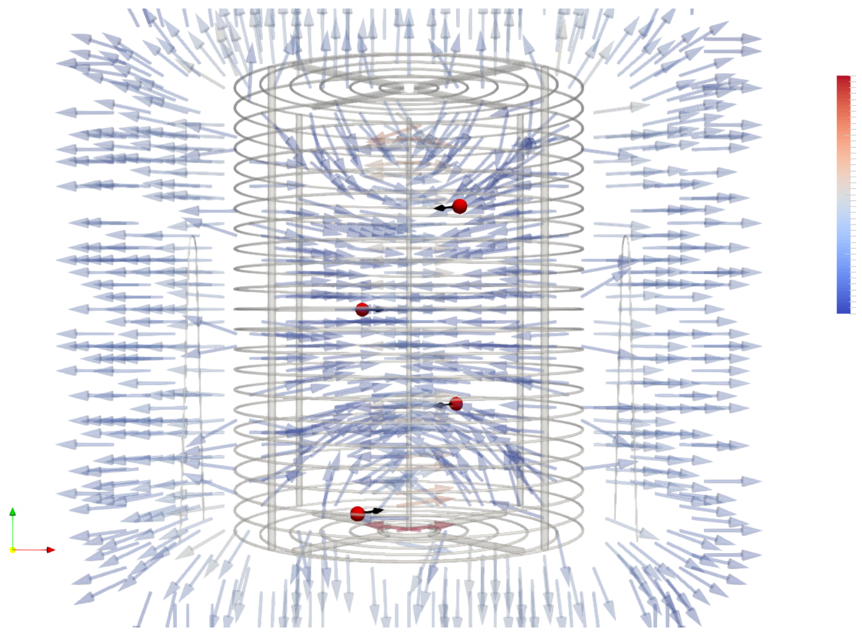


(a)

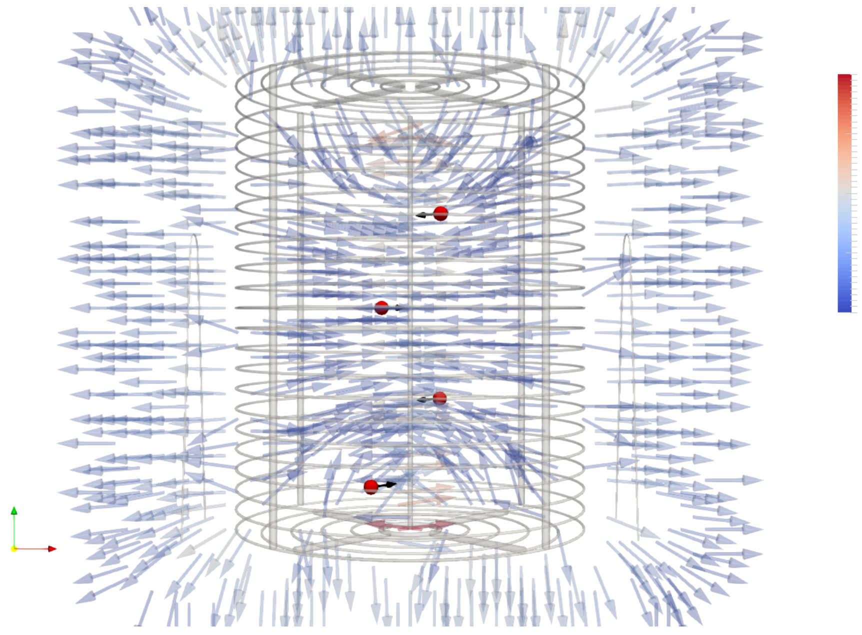


(b)

Figure 4.8: The red spheres represent the ions moving in the electric field. Picture (a) is from a time step right before picture (b).



(a)



(b)

Figure 4.9: The same result as visualized in Figure 4.8, but in the x-z-plane. Picture (a) comes before picture (b) in time.

4.3 Approach to simulate the ionization gauge

Specific weight of the macro particles

In the simulations carried through in this work, the electrons are represented by macro particles. The number of real electrons N_{elec} that had to be emitted from the hot cathode is calculated as

$$N_{elec} = \frac{\Delta t \cdot I_e}{e} \tag{4.3}$$

and the specific weight for the macro particles is calculated as

$$\text{specific weight} = \frac{N_{elec}}{N_{MP,elec}} \tag{4.4}$$

In (4.3), I_e is the electron current and e is the elementary charge. In (4.4), $N_{MP,elec}$ is the number of macro particles representing the electrons that are emitted from the cathode at every time step. $N_{MP,elec}$ is an input parameter in the simulations and in this work, only one macro particle representing an electron is emitted at each time step. From the equations (4.3) and (4.4), the specific weight for the macro particles representing electrons becomes approximately $6.2 \cdot 10^6$.

The macro particles representing hydrogen are injected into the domain by the function `PlasmaInjectorModified`. Two hydrogen macro particles are placed in each cell and the specific weight of the hydrogen gas is calculated as

$$\text{weight}_H = \frac{n \cdot V_{cell}}{N_{MP,H}} \tag{4.5}$$

where V_{cell} is the volume of a mesh cell and $N_{MP,H}$ is the number of macro particles representing hydrogen. n is the number of hydrogen particles per m^3 and this value is calculated through the ideal gas law, given by equation (1.4).

Starting positions of the electrons

The cathodes in this work consist of two cylinders merged together, shaped as two "V" tuned upside down. Figure 4.2 indicates the shapes and positions of those cathodes and only the cathode to the left in Figure 4.2 emits electrons. Compared to the cathodes in [9], the geometry of the cathodes in this work was more easy to implement in the code.

The starting positions of the electrons are randomized on the surface of the emitting cathode. The random variables Y, Z and θ are used to generate a random starting position. To describe this randomization algorithm, the two end points towards the ground of the emitting cathode need to be defined. The left end point is denoted

$$[x_{\text{end}}, y_{\text{left}}, z_{\text{end}}]$$

and the right point is denoted

$$[x_{\text{end}}, y_{\text{right}}, z_{\text{end}}]$$

The x- and z-position of those points are the same and have therefore the same notation.

The random variable Y takes the two values y_{right} or y_{left} with equal probability. Z and θ are uniformly distributed such that

$$Z \in \text{Uniform(0,H)}$$
$$\theta \in \text{Uniform}(0,2\pi)$$

where H stands for the height of the hot cathode. The starting position of the macro particles are then given by

$$x_{\text{start}} = r_c \cdot \cos(\theta) + x_{\text{end}} \tag{4.7a}$$
$$y_{\text{start}} = r_c \cdot \sin(\theta) + Y \cdot \frac{H-Z}{H} \tag{4.7b}$$
$$z_{\text{start}} = Z + z_{\text{end}} \tag{4.7c}$$

where r_c is the radius of the cylinders that makes up the emitting cathode. An illustration of the randomized starting positions of the macro particles is given in Figure 4.10.

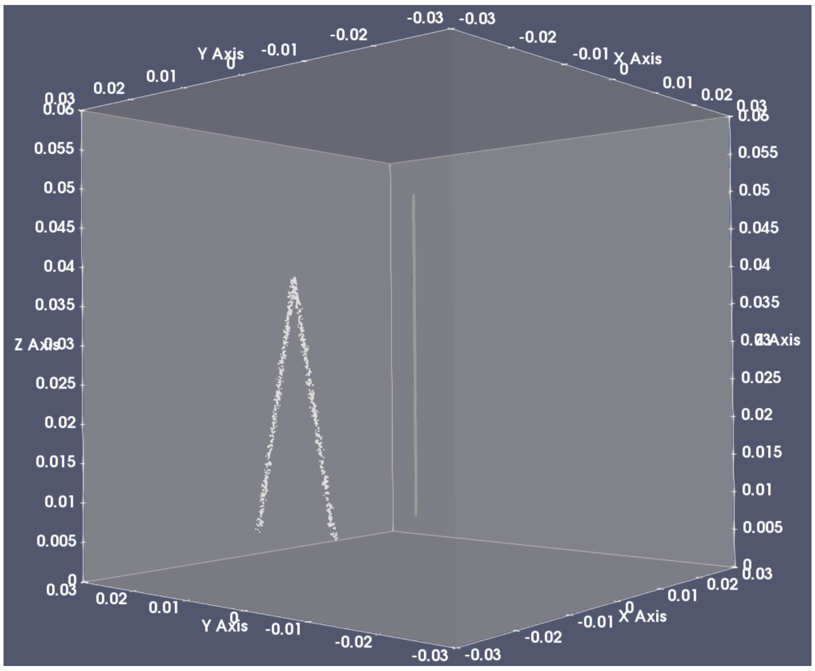


Figure 4.10: The starting positions of the electrons are shown by the small, white dots inside of the cube.

The initial speed of the electrons v_0 is set to $4.42 \cdot 10^6$ m/s. This speed is calculated from the work function given that the cathode had a voltage of +50 volt. The components of the initial velocity vector $\boldsymbol{v}_{\text{start}}$ of the electrons, are given by random, spherical coordinates. Those coordinates are computed as

$$\begin{aligned} v_{x,\text{start}} &= v_0 \cdot \cos(\theta)\sin(\varphi) && (4.8a)\\ v_{y,\text{start}} &= v_0 \cdot \sin(\theta)\sin(\varphi) && (4.8b)\\ v_{z,\text{start}} &= v_0 \cdot \cos(\varphi) && (4.8c) \end{aligned}$$

and it holds that $|\boldsymbol{v}_{\text{start}}| = v_0$. Here is

$$\varphi \in \text{Uniform}(0,\pi)$$

If more than one macro particle per iteration would be generated, the random variables Z, θ and φ would be random vectors. In that case, all the macro particles would be generated either on the cylinder to the left or on cylinder to the right at every iteration.

The Python function `random.random()` was used to generate all the random variables. `random.random()` generates a uniformly random number in the interval $[0, 1)$. More information about random numbers generated in Python can be found in [35].

Collecting the ions

Macro particles that are scraped on particle scrapers or absorbed on the walls of the computational domain, are deleted in the source code of Warp by default. Since the objective in this work is to simulate an ion current in the ion collector, it is necessary to somehow count how many macro particles representing ions that get absorbed on the ion collector. A macro particle representing several ions will be denoted MP_{ion}.

To save particle data that has to do with macro particles that disappear from the domain, one can set the variable

```
lsavelostpart = True
```

This way, it will be possible to retrieve the number of macro particles of a certain specie that have disappeared. To return the number of disappeared MP_{ion}, the function call

```
ions.getn(lost=1)
```

is made, where `ions` is an object that contains data about the hydrogen ions. This call will return the total number of lost MP_{ion} and not only the number of MP_{ion} that have been absorbed by the ion collector.

To sort out the MP_{ion} that have been absorbed on the ion collector from the MP_{ion} that have been absorbed or lost elsewhere in the domain, a for-loop on the following form was implemented:

```
Data:
$W_{ion}$: the specific weight of $MP_{ion}$
$I_i$: the ion current
t: elapsed time
for $n = 1$ to Total number of lost $MP_{ion}$ do
    if The position of lost $MP_{ion,n}$ is on the ion collector then
        Add the number of collected $MP_{ion}$ by 1;
    end
end
$I_i = \frac{W_{ion} \cdot (\text{collected MP}_{ion})}{t}$;
```

To verify the if-statement in the for-loop, built in functions that gives the coordinates of the lost particles are used. A problem with this approach is that the arrays that store the data of the lost particles, keep growing and growing as ions disappear throughout the simulation. For simulations that run for a long time, those arrays may be completely filled or it may take a very long time to execute the for-loop. One solution could be to continuously count the particles absorbed by the ion collector and then delete all the particles throughout the simulation, so that not so much memory is needed to store all data. Unfortunately, no way to delete lost particle data during simulation time was found. In the script that was used for simulating the ionization gauge, all the data of the lost particles was saved.

Other attempts to improve the simulation

In Warp it is possible to create functions that removes particles on positions defined by the user. To remove the particles, an array called `gaminv` is used. Every element in `gaminv` corresponds to a particle and by setting an element in `gaminv` to -1, that corresponding particle is removed.

This approach was used to create a function that removes particles close to the boundaries of the domain. This way, the only lost ions that are saved are the ones that disappear on the ion collector, provided that none of the other conducting objects absorb the ions. This function worked well if the code was executed sequentially, but in parallel the vector `gaminv` was divided to the different processors so that an index error appeared. Warp uses the MPI library for message passing for the parallelization [8]. It should also be mentioned that during a simulation, the sum of the neutral particles and the positive ions should remain constant in order to keep a constant pressure inside of the domain. So whenever the sum of neutral particles and ions decreases, new neutral gas molecules are generated at random positions in the domain so that the pressure remains the same.

Attempts to switch the variable `lsavelostpart` between `True` to `False` during simulation time were also made. The thought was that if `lsavelostpart` was switch from `False` to `True`, the saved lost particles would be deleted. By setting `lsavelostpart=True` again during simulation time, hopefully the counting of the lost particles would be reset from zero. It turned out that if `lsavelostpart` was switched from `True` to `False` during simulation time, the counting of the lost particles only paused until `lsavelostpart` was set to `True` again.

In order to speed up the computations, a mesh refinement can be applied on parts of the computational domain where extra fine resolution is needed. This way, a fine mesh don't have to be applied on parts that are not important for the results and computational time can be saved. The field solver that can handle mesh refinement is called `MRBlock3D`, which is a class derived from the field solver `MultiGrid3D`. Where in the domain a mesh refinement will be applied and how many times finer the resolution in the x-,y- and z-direction will be is decided by the user.

An attempt to make a finer mesh of the domain where the anode grid is placed was carried through. The original mesh was divided into 30 grid points in each direction, but in the region where

$$\begin{aligned} x &\in [-21, 21] \text{ millimeter} \\ y &\in [-21, 12] \text{ millimeter} \\ z &\in [15, 21] \text{ millimeter} \end{aligned}$$

the mesh was 4 times smaller in each direction. The mesh refinement did not seem to have any impact on the absorption of the electrons, which is shown in Figure 4.11. This result is contrary to the results in Section 4.5.1, where the absorbed electrons were collected more uniformly over the anode grid if the mesh was finer. It is not clear why there is a difference, but one thought is that it has to do with the size of the time step. In the case where the mesh was refined over the whole domain, the time step also decreased. In the case where the mesh refinement was applied only to a small part of the domain, the time step was still adapted to the coarser part of the mesh.

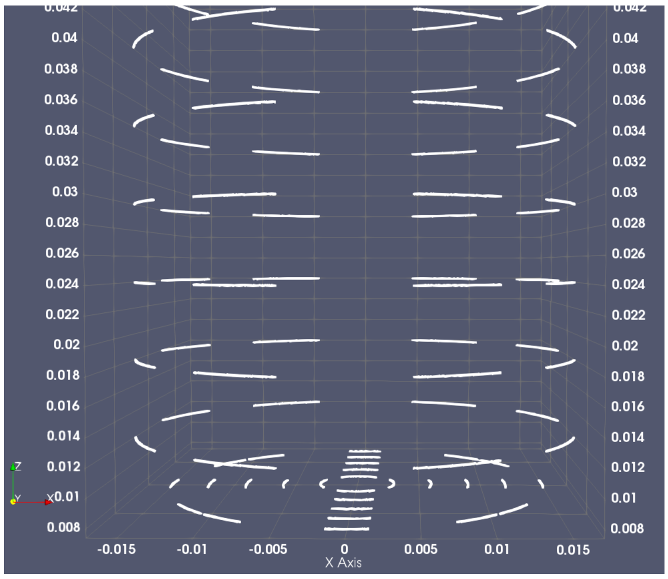


Figure 4.11: A mesh refinement was applied around the torus placed 18 millimeter in the z-direction.

4.4 CFL-condition and stability

It is mentioned in [16] that a necessary constraint in all PIC codes is that the particles should not travel further than to a neighbouring cell in one iteration. It is also mentioned that it is generally a good idea to let the particles travel a shorter distance than a length of a cell in every iteration. Moreover, it is described in Section 3.3.4 that the charge of the particles are interpolated onto the grid points of the mesh. To avoid a situation where some grid points don't get any charge from a certain macro particle just because it is traveling more than one cell at each iteration, the time step had to be adjusted.

Let a particle p have the velocity $\boldsymbol{v} = [v_x, v_y, v_z]$ at an arbitrary time during a simulation. If particle p should not travel more than to the neighbouring cell at each time step, then the time step has to fulfill

$$\begin{aligned} \Delta t \cdot v_x &< C \cdot \Delta x \\ \Delta t \cdot v_y &< C \cdot \Delta y \\ \Delta t \cdot v_z &< C \cdot \Delta z \end{aligned}$$

where C is a constant smaller or equal to 1, Δt is the time step and Δx, Δy and Δz are the length of a grid cell in the x-,y- and z-direction respectively. To calculate an appropriate value of Δt, the maximum values of the velocity components from all the electrons in the domain are selected. Those components are here denoted $v_{x,max}$, $v_{y,max}$ and $v_{z,max}$ and are not necessary selected from the same electron.

The velocity components of the ions and the neutral particles are not taken into account, since they move so much slower than the electrons. The time step is then calculated as

$$\Delta t = C \cdot \min(\frac{\Delta x}{v_{x,max}}, \frac{\Delta y}{v_{y,max}}, \frac{\Delta z}{v_{z,max}}) \tag{4.9}$$

To avoid having to gather the maximum values of the velocity vectors of the electrons at each time step, a pre-simulation was of 20000 iterations was carried through. After 10000 iteration, the minimum values of the time steps was saved in a data file every 100th iteration. Then the minimum value of those times steps was selected, which was $\Delta t = 1.15 \cdot 10^{-10}$ seconds. The time step in the long simulations that will be described in Section 4.5.3, was set to $\Delta t = 1 \cdot 10^{-10}$ seconds to be sure that the time step would be small enough. The constant C was set to 0.7.

4.5 Results of the simulations

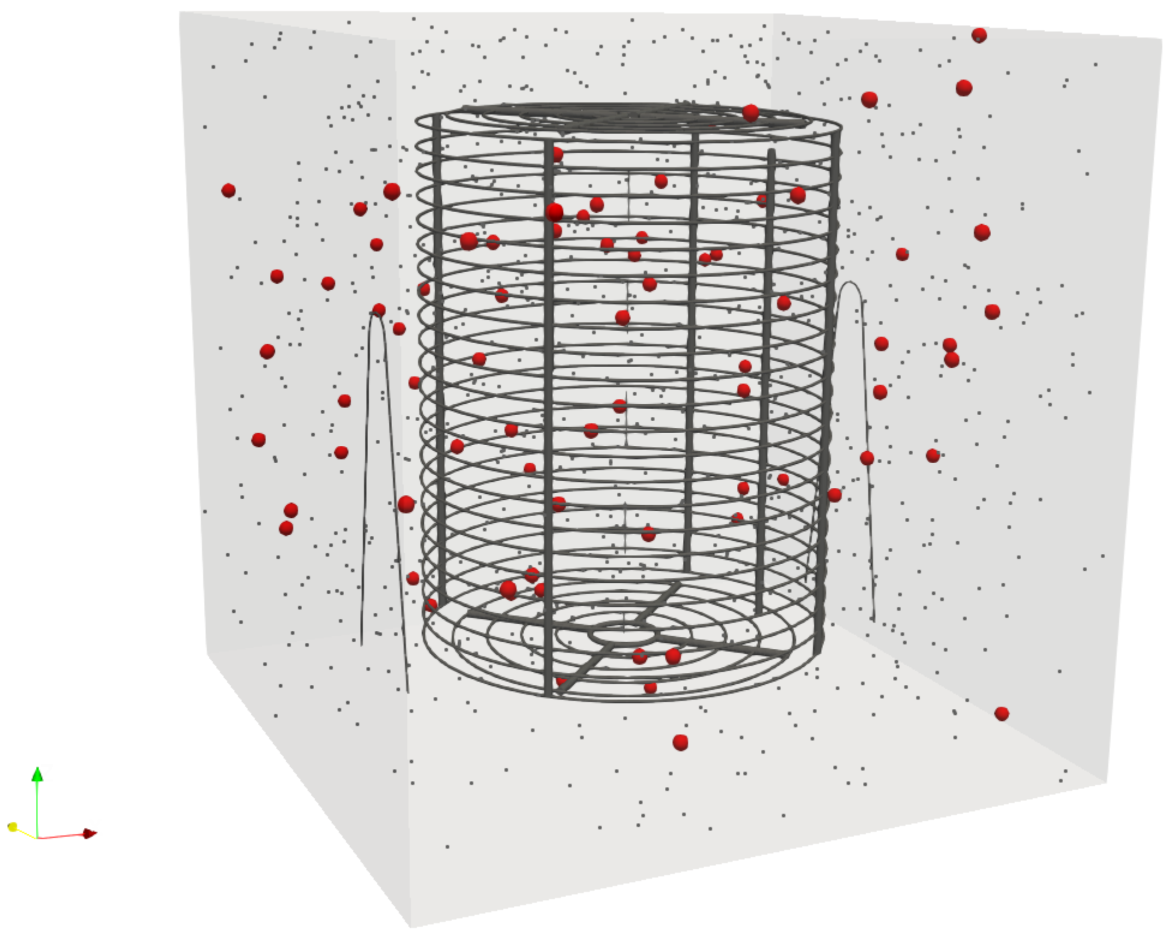


Figure 4.12: A simulation in action.

Figure 4.12 shows the result from a simulation where the domain Ω is filled with hydrogen molecules and electrons are emitted from the hot cathode. The small dots represent the electrons, the bigger spheres represent the ions that have been generated from the hydrogen during the simulation and the cube that surrounds the ionization gauge and the particles represent the computational domain. The simulation is running on 30 processors, so the domain is divided into 30 parts. Each processor takes care of the computations of one of these parts each. The parts are divided as $6 \times 5 \times 1$ in the x-, y- and z-direction respectively.

4.5.1 Issue with the absorption of electrons

If the anode grid is very thin, only certain parts of it will absorb electrons. If the mesh where the anode grid is placed gets finer, the electrons will be absorbed more uniformly on the grid. An example of this can be seen in figure 4.13.

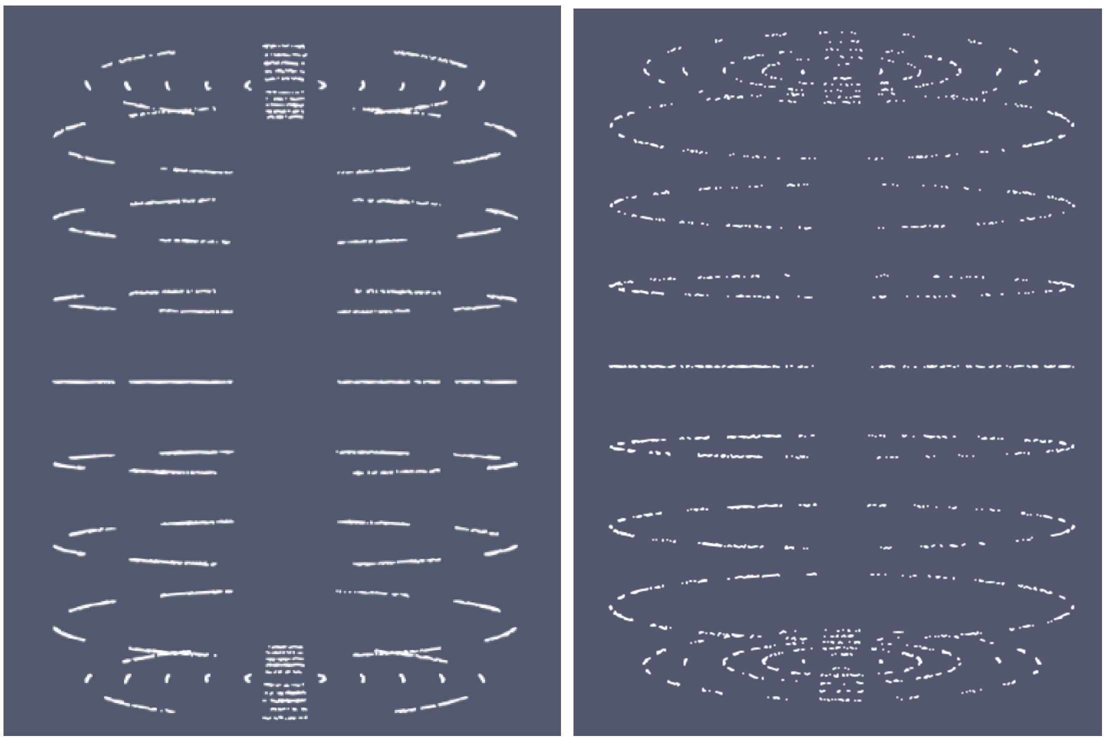

(a) (b)

Figure 4.13: The white dots show the positions where the electrons are absorbed. In (a) the domain is divided into 40×40×40 cells and in (b) the domain is divided into 80×80×80 cells.

As an experiment, the anode grid was made thicker to see if it would affect the absorption of the electrons. The anode rings radius was 0.8 millimeter in this simulation. In Figure 4.14 it can be seen that the electrons are absorbed more uniformly.

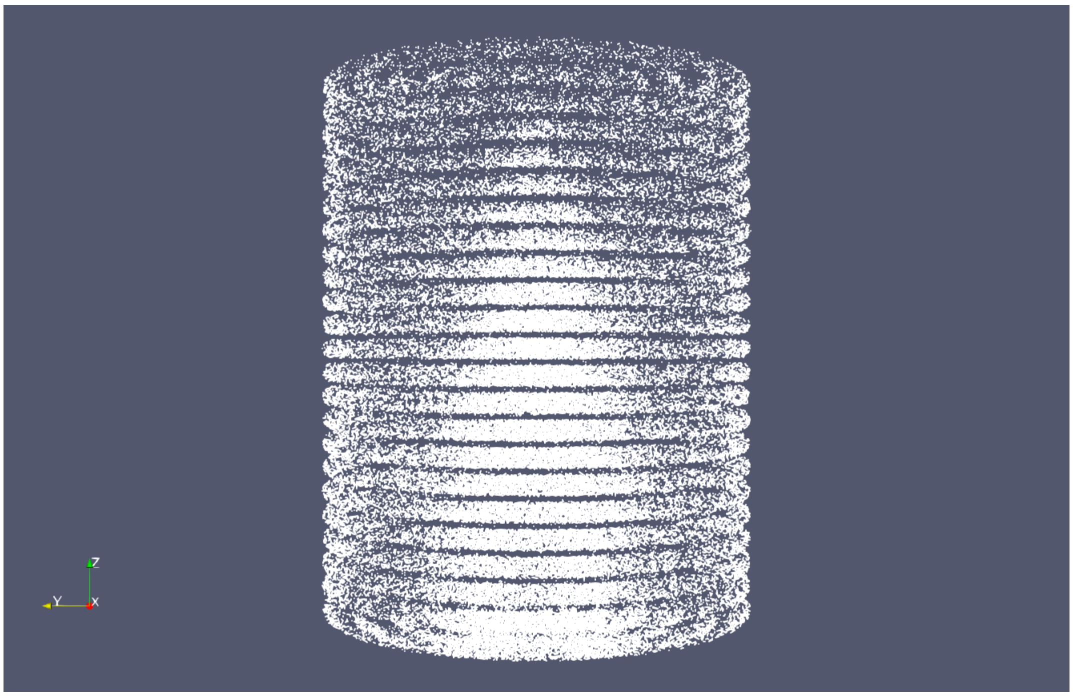


Figure 4.14: The small dots represent the final destination of the electrons. The emitting cathode is placed between the viewer and the anode grid in the picture.

4.5.2 The ionization

When some code was added to the class `Ionization`, the right number of neutral hydrogen molecules was deleted if positive ions were generated. Before that code was added, the number of neutral hydrogen molecules remained the same throughout the simulation, even though positive ions were generated. To test if the new code in the class `Ionization` worked, a simulation with a simple set up was made. The domain was filled with neutral hydrogen molecules and electrons were emitted from the centre of the domain. Figure 4.15 show the positions where the neutral hydrogen molecules are deleted from the simulation and Figure 4.16 show those positions again, together with the positions where the positive ions are added to the simulation. It can be concluded that the ions are added at positions close to where the neutral hydrogen molecules are deleted, which was the purpose of the new code. It can also be seen that some of the gas molecules are deleted far away from the cluster of ions. This may have to do with that if there are no neutral hydrogen molecules nearby where an electron generate an ion, a neutral hydrogen molecule is deleted randomly in the domain. A description of the added code is found in Section 3.6.

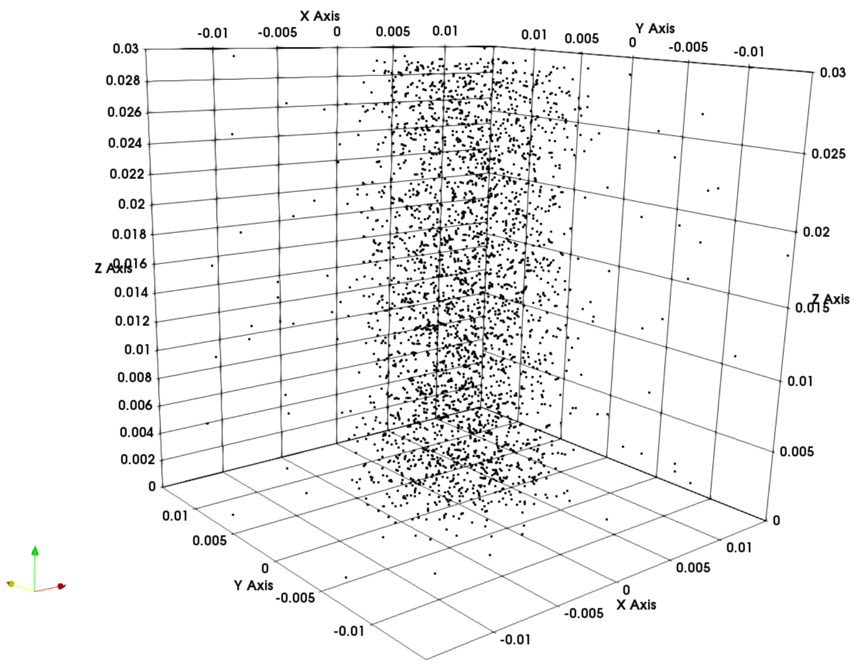


Figure 4.15: The small dots represent the positions where the neutral hydrogen molecules are deleted from the simulation.

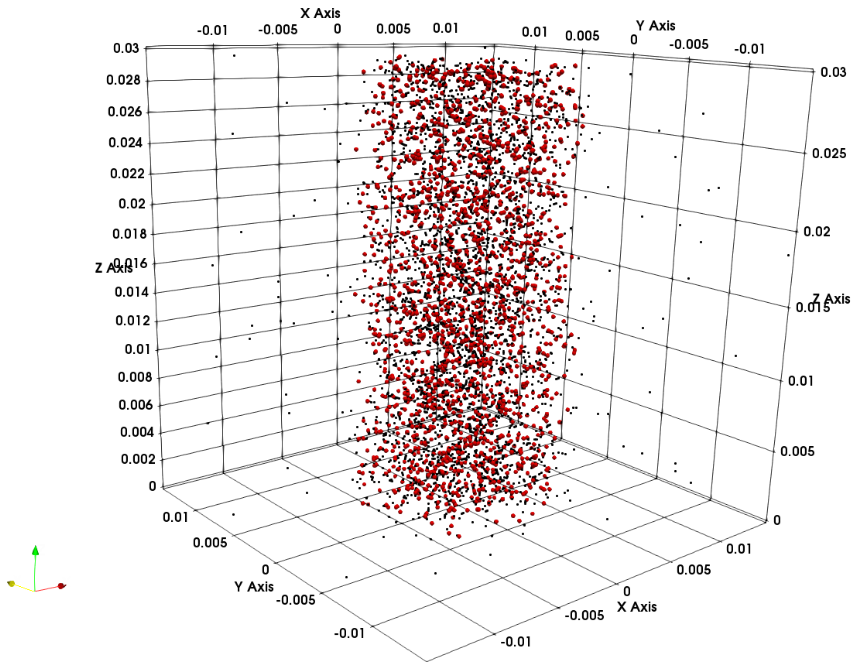
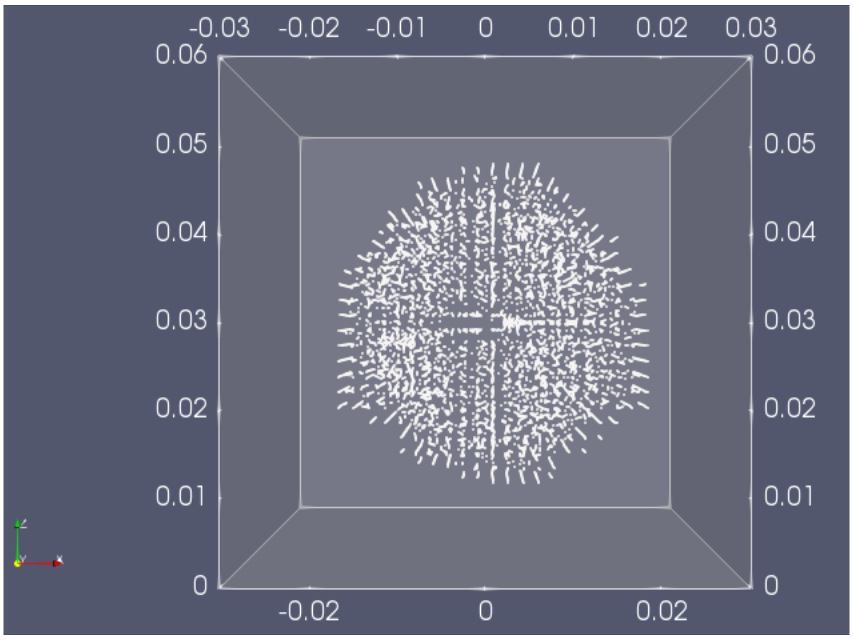
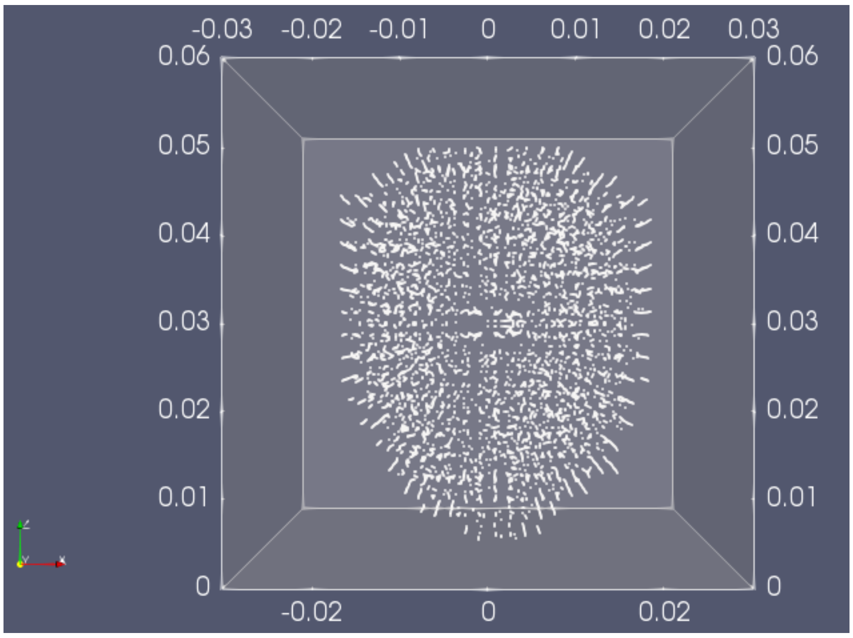


Figure 4.16: The bigger red spheres represent the positions where the ions are added to the simulation.

As described in Section 3.6, tunnel ionization occur if the specific weight of the electrons is high enough. Figure 4.17 shows the results from two simulations where this type of ionization is generated. The ions are generated because a macro particle representing an electron is emitted from the centre of the domain and the only difference between the two simulations is the grid resolution in the z-direction (upwards in the figure). Compared to Figure 4.17 (a), the mesh resolution in the z-direction in Figure 4.17 (b) is coarser and the ionization also seems to propagate faster in the z-direction in Figure 4.17 (b). The reason might be that at the beginning of the simulation, one macro particle representing a gas molecule is placed at the centre of each cell. This means that the distance between the macro particles increases in the direction where the mesh get coarser. To be sure to avoid nonphysical results due to the geometry of the mesh, cubic cells were used in the simulations.



(a)



(b)

Figure 4.17: In picture (a), the number of grid points is $30 \times 30 \times 30$. Picture (b) has $30 \times 30 \times 20$ grid points.

4.5.3 Long simulations

Simulations where the electrons were represented by macro particles were carried through. These simulations were made before the new code in the class `Ionization` was added by the author of this thesis. One electron macro particle was emitted at every time step and the specific weight was $6.2415 \cdot 10^6$, a value calculated from equations (4.3) and (4.4). Since the electrons move so fast compared to the ions, the simulations were planed to run for about two weeks. After two weeks time it was assumed that enough ions would have been collected so an estimation of the sensitivity of the gauge could be obtained.

First, a simulation with absorbing boundary conditions for the particles was made. Two macro particles representing gas molecules were placed in each cell. The number of electrons in the domain did not seem to change significantly after the simulation had run a couple of 100 000 iterations. This can be seen in figure 4.18.

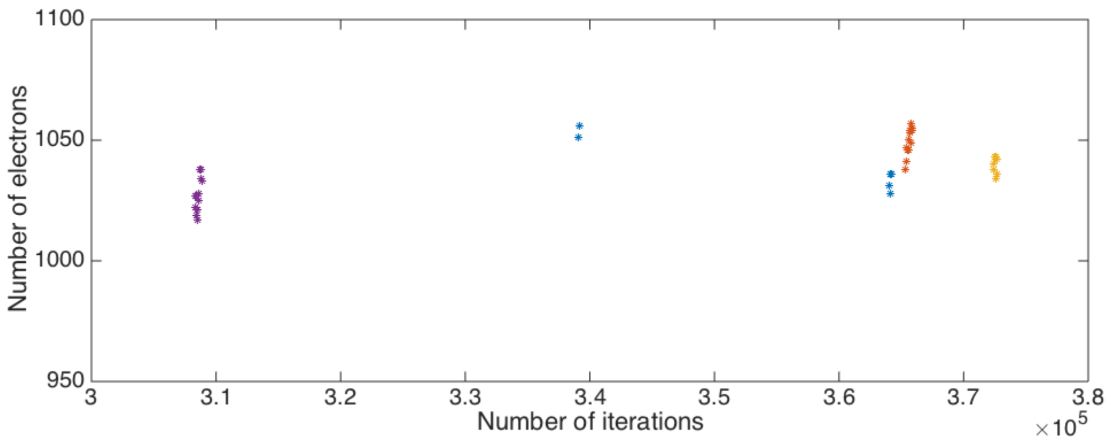


Figure 4.18: Total number of electrons in the domain at different iterations. After 372 700 iterations the simulation was interrupted.

There is a boundary condition called `prwall` implemented in Warp, which is a cylindrical wall that absorbs the particles that collide with it. `prwall` was used in the simulation, with a radius of 30 millimeter. The cylindrical form was chosen to shrink the domain and consequently save computational power. No ionization of the hydrogen occurred and after about 372700 iteration a segmentation error appeared, which interrupted the simulation.

Then a second simulation with reflective boundary conditions for the particles was done. It was assumed that the reflective boundary conditions would generate more ions, since more electrons that can generate ions would be in the domain. In a real situation with an ionization gauge surrounded by a wall and electrons absorbed by the wall, the wall would after a while be negatively charged due to the negative charge of the electrons. That would make the electrons repel from the wall. This reasoning was used as a justification to use reflective boundary conditions for the particles.

No boundary with shape of a cylinder that could reflect particles was found, so the domain had the shape of a cube in this second simulation. Since no ionization occurred in this simulation, it was interrupted manually.

In both of the simulations, two macro particles representing hydrogen molecules were placed initially in each cell in the domain. For the field, Dirichlet boundary conditions was applied to the boundaries of the domain.

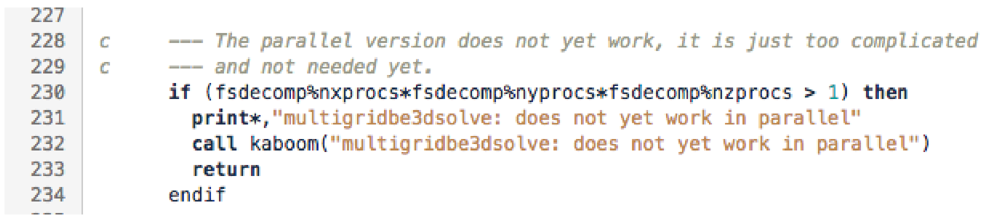
4.6 Discussion and future work

A problem with the absorption of the macro particles on the conductive objects have been observed. The macro particles that represents electrons are absorbed only on certain parts, which is shown in Section 4.2.1. Furthermore, in the simulation with the four ions in Section 4.2.3 the ions did not get absorbed by the ion collector. This problem might be caused by the small dimensions of the parts of the ionization gauge. Simulations with a bigger radius on the ion collector and bigger toruses of the anode grid were carried through and then no problems with the absorption of the particles were detected.

To carry through the simulations of interest successfully, it is essential that the absorption of the particles works correctly. It is specially important that the ion collector absorbs the macro particles that represents the ions. One way to make the ion collector absorb more ions is to increase the radius of the ion collector. In that case the model of the ionization gauge in this work would no longer have the same measurements as the ionization gauge in [9]. Then, comparing the results in this work with the results in [9] may no longer make sense.

In Section 2.4 it is explained that if the electrons are represented as macro particles, the time step in the simulations needs to be very small. Since both electrons and ions have to be included in the simulation of the ionization gauge in this work, many iterations are needed because the ions move so slowly compared to the electrons.

To reduce the number of computations in the simulations, Boltzmann distributed electrons that are available in Warp can be used instead of macro particles that represent many electrons. There is a field solver called `multigridbe3dsolve` implemented in Warp, which solves Poisson's equation with a Boltzmann electron term added. Attempts to use this field solver were carried through in this work, but a segmentation error occurred every time. It was found that `multigridbe3dsolve` was not completely implemented in Warp and the developers were informed about this. After they fixed that field solver, a segmentation error did not longer appear. In this work, it was not accomplished to simulate ionization with `multigridbe3dsolve` and in the source code where that field solver is implemented, the comment "The parallel version does not yet work, it is just too complicated and not needed yet" was found. Therefore, no more attempts to use `multigridbe3dsolve` were carried through. But a future task is to make a simulation with such Boltzmann distributed electrons. It can be tested if the class `Ionization` can be used together with Boltzmann distributed electrons as well.



```
c     --- The parallel version does not yet work, it is just too complicated
c     --- and not needed yet.
      if (fsdecomp%nxprocs*fsdecomp%nyprocs*fsdecomp%nzprocs > 1) then
        print*,"multigridbe3dsolve: does not yet work in parallel"
        call kaboom("multigridbe3dsolve: does not yet work in parallel")
        return
      endif
```

Figure 4.19: Comments found in the source code.

The results of numerical simulations become more believable if they agree with mathematical theory and with experiments [36]. In Section 4.2, electric fields generated by different conductive objects are visualized. At the boundary surfaces of the domain and at the surfaces of the conductive objects, homogeneous Dirichlet boundary conditions are applied and the field vectors are perpendicular to those surfaces. This result agrees with the theory in Section 2.2.2. The electric fields get stronger at spatial positions close to the solid objects that generate the fields, which makes physical sense. It was also shown in Section 4.2 that the positively charged ions were affected by the electric field according to the theory in Section 2.3. However, it is necessary to further investigate if the particle motion generated by the magnetic field also works according to theory.

At the beginning of the work with this thesis, the article [9] was intended to be used as a benchmark for the simulations in this work. For example, the sensitivity of the ionization gauge simulated by Warp could be compared to the sensitivity of the gauge simulated in the article [9]. If the results in this work would agree with the results in [9], the results in this work would be more believable. Due to problems with setting up a working simulation in Warp, there was not time to generate such results with Warp that could be compared to the results in [9].

It should be tested if the class `Ionization` can be used to simulate the physics of an ionization gauge of the type that is described in this thesis. At the beginning of the work with this thesis, it was not understood how the class `Ionization` should be used properly and after a while, the issues with that class were encountered. Now when more knowledge is obtained and some problem with the code is fixed, it may be possible to simulate the physics of an ionization gauge successfully in Warp. The results obtained after such simulations should be compared to already known results, for example the results in [9].

Appendix A

Definitions and notation

• **The computational domain** $\Omega \in \mathbb{R}^3$ is a simplified representation of the physical domain that is to be simulated.

• **Phase space** is a space containing all the possible states of the particles positions and their velocities (cf. *Encyclopedia of Mathematics* [20]).

• A variable that is representing a vector quantity is written in **bold front**. Scalar quantities are written in *normal front.* For example, $\boldsymbol{v}$ denotes a vector and ρ denotes a scalar.

• Variables used in numerical schemes can have indexes both at the top and at the bottom, for example $v^n_{i,j,k}$. The superscript n at the top indicates the time step and the subscripts i, j and k at the bottom indicate the spatial position.

• The spatial position of a particle p at a time step n, is given by a vector denoted $\boldsymbol{x}^n_p$. The electrical field in the computational domain is changing in both time and space and to simplify the notation, the electrical field where particle p is located at a time step n, is denoted

$$\boldsymbol{E}(\boldsymbol{x}^n_p) \overset{not.}{=} \boldsymbol{E}^n_p$$

In the same way, a magnetic field at particle p:s position at time step n is denoted

$$\boldsymbol{B}(\boldsymbol{x}^n_p) \overset{not.}{=} \boldsymbol{B}^n_p$$

• Line integrals along a curve C between the two points $\boldsymbol{a}$ and $\boldsymbol{b}$ are denoted

$$\int_{\boldsymbol{a}}^{\boldsymbol{b}} \mathbf{F} \cdot \boldsymbol{\tau} \, dx \equiv \int_C \boldsymbol{F} \cdot \boldsymbol{\tau} \, dx$$

where $\boldsymbol{F}$ is a vector field and $\boldsymbol{\tau}$ is the tangential vector of the curve C. If the curve C is *closed*, meaning that $\boldsymbol{a} = \boldsymbol{b}$, the curve integral will be denoted

$$\oint_C \boldsymbol{F} \cdot \boldsymbol{\tau} \, dx$$

• Double integrals over a surface S will be denoted

$$\int_S \boldsymbol{F} \cdot \mathbf{n} \, d\sigma(x)$$

where $d\sigma(x)$ is an infinitesimal area element of S and $\boldsymbol{n}$ is an outer normal unit vector to $\sigma(x)$. If the surface S is a *closed surface*, the double integral will be denoted

$$\oint_S \boldsymbol{F} \cdot \mathbf{n} \, d\sigma(x)$$

• Triple integrals over a volume V will be denoted

$$\int_V f(x) \, dx.$$

where f is a scalar function.

Appendix B

Used mathematical identities

The divergence theorem and Stokes theorem

Let $\boldsymbol{F}$ be a vector field and $\partial\nu$ a closed surface that bounds a volume ν. Then the *divergence theorem* states that

$$\oint_{\partial\nu} \boldsymbol{F}\cdot\boldsymbol{n}\,d\sigma(x) = \int_{\nu} (\nabla\cdot\boldsymbol{F})\,dx \tag{B.1}$$

where $\boldsymbol{n}$ is the unit outward normal vector.

Let S be a piecewise smooth surface that is bounded by the closed curve ∂S. Then *Stokes theorem* states that

$$\oint_{\partial S} \boldsymbol{F}\cdot\boldsymbol{\tau}\,dx = \int_{S} (\nabla\times\boldsymbol{F})\cdot\boldsymbol{n}\,d\sigma(x) \tag{B.2}$$

where $\boldsymbol{n}$ is the unit outward normal vector. The divergence theorem and Stokes theorem are explained further in Griffiths [10] for example.

The Helmholtz theorem

Let $\boldsymbol{F}(\boldsymbol{r})$ be a vector field with its divergence given by

$$D = \nabla\cdot\boldsymbol{F} \tag{B.3}$$

and its curl given by

$$\boldsymbol{C} = \nabla\times\boldsymbol{F} \tag{B.4}$$

If D and $\boldsymbol{C}$ goes to zero sufficiently fast as $r\to\infty$ and $\boldsymbol{F}$ goes to zero as $r\to\infty$, then $\boldsymbol{F}$ is uniquely given by

$$\boldsymbol{F} = -\nabla U + \nabla\times\boldsymbol{W} \tag{B.5}$$

where

$$U(\boldsymbol{r}) = \frac{1}{4\pi}\int_{\text{all space}} \frac{D(\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|}d\sigma' \tag{B.6}$$

and

$$\boldsymbol{W}(\boldsymbol{r}) = \frac{1}{4\pi}\int_{\text{all space}} \frac{\boldsymbol{C}(\boldsymbol{r'})}{|\boldsymbol{r}-\boldsymbol{r'}|} d\sigma' \tag{B.7}$$

Here, $\boldsymbol{r}$ and $\boldsymbol{r'}$ are vectors from the origin of some coordinate system in the domain, where $\boldsymbol{r'}$ typically is a vector to the source of the vector field $\boldsymbol{F}$ and $\boldsymbol{r} \neq \boldsymbol{r'}$. r is the magnitude of $\boldsymbol{r}$. The Helmholtz theorem is proven and further explained in Griffiths [10].

Potentials

In Griffiths [10], the statements below are made:

If the curl of a vector field $\boldsymbol{F}$ vanishes everywhere, then that vector field can be written as the gradient of a scalar potential. The converse relationship also holds.

$$\nabla \times \boldsymbol{F} = 0 \quad \Longleftrightarrow \quad \boldsymbol{F} = -\nabla\phi \tag{B.8}$$

If the divergence of a vector field $\boldsymbol{F}$ vanishes everywhere, then that vector field can be expressed as the curl of a vector potential $\boldsymbol{A}$. The converse relationship also holds.

$$\nabla \cdot \boldsymbol{F} = 0 \quad \Longleftrightarrow \quad \boldsymbol{F} = \nabla \times \boldsymbol{A} \tag{B.9}$$

Second derivatives

It holds that the curl of a gradient is equal to zero

$$\nabla \times (\nabla \cdot \boldsymbol{F}) = 0 \tag{B.10}$$

and that the divergence of a curl also is equal to zero

$$\nabla \cdot (\nabla \times \boldsymbol{F}) = 0 \tag{B.11}$$

See for instance Griffiths [10].

Cross products and scalar products

Let

$$\begin{aligned}\boldsymbol{a} &= [a_1, a_2, a_3]^T \in \mathbb{R}^3\\ \boldsymbol{b} &= [b_1, b_2, b_3]^T \in \mathbb{R}^3\\ \boldsymbol{c} &= [c_1, c_2, c_3]^T \in \mathbb{R}^3\end{aligned}$$

From the definition of the vector $\boldsymbol{a} \times \boldsymbol{b}$ it holds that

1. $|\boldsymbol{a} \times \boldsymbol{b}| = |\boldsymbol{a}||\boldsymbol{b}|\sin(\theta)$

2. $\boldsymbol{a} \times \boldsymbol{b}$ is orthogonal to $\boldsymbol{a}$ and $\boldsymbol{b}$

Some properties of the cross product are

$$\boldsymbol{a} \bullet (\boldsymbol{b} \times \boldsymbol{c}) = \boldsymbol{b} \bullet (\boldsymbol{c} \times \boldsymbol{a}) = \boldsymbol{c} \bullet (\boldsymbol{a} \times \boldsymbol{b}) \tag{B.12}$$
$$\boldsymbol{a} \times \boldsymbol{b} = -\boldsymbol{b} \times \boldsymbol{a} \tag{B.13}$$
$$\boldsymbol{a} \times (\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a} \times \boldsymbol{b} + \boldsymbol{a} \times \boldsymbol{c} \tag{B.14}$$

cf. Bertil Westergren [37]. The cross product of $\boldsymbol{a}$ and $\boldsymbol{b}$ can be expressed as

$$\boldsymbol{a} \times \boldsymbol{b} = R_{[\times]} \boldsymbol{a} \tag{B.15}$$

where $R_{[\times]}$ is a skew-symmetric matrix such that

$$R_{[\times]} = \begin{bmatrix} 0 & b_3 & -b_2 \\ -b_3 & 0 & b_1 \\ b_2 & -b_1 & 0 \end{bmatrix}$$

Cf. Bernstein [38], **Fact 3.10.1**.

Some properties of the scalar product are

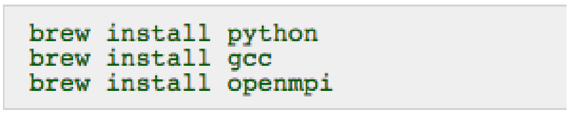
$$\boldsymbol{a} \bullet \boldsymbol{b} = \boldsymbol{b} \bullet \boldsymbol{a} \tag{B.16}$$
$$\boldsymbol{a} \bullet (\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a} \bullet \boldsymbol{b} + \boldsymbol{a} \bullet \boldsymbol{c} \tag{B.17}$$

cf. Bertil Westergren [37].

Appendix C

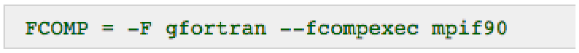
Installation of Warp

The simulations for this thesis were done in Warp version 4.5. To install Warp 4.5 the steps described by Grote [39] were followed. However, some steps in the installation process had to be done differently from what is described by Grote [39]. The following changes were made:



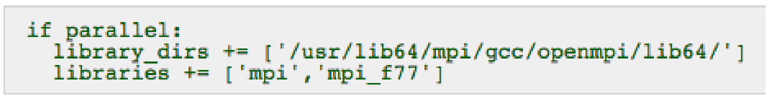
```
brew install python
brew install gcc
brew install openmpi
```

The key word `brew` was replaced with `zypper`.



```
FCOMP = -F gfortran --fcompexec mpif90
```

The key word `gfortran` was replaced with `gcc-fortran`.



```
if parallel:
  library_dirs += ['/usr/lib64/mpi/gcc/openmpi/lib64/']
  libraries += ['mpi','mpi_f77']
```

The bracket `['mpi','mpi_f77']` was replaced with `['mpi']`.

The operation system *openSUSE Leap 42.2* was used in this work.

Appendix D

Debugging of the Warp code

Ionization

To find out from where in the Warp code the ionization was computed, a Python debugger was used. It was found that the script `tunnel_ionization` was used for that. The outcome of the debugging is shown below.

```
/home/olle/runs/
171024_Alexanders_script_long_simulation.py(389)inject_electrons()
->electrons_from_cathode.addparticles(x=x_, y=y_, z=z_, vx=vx_,
vy=vy_, vz=vz_, lallindomain=False)

> /usr/lib64/python2.7/site-packages/warp/particles/
tunnel_ionization.py(146)generate()
-> def generate(self,dt=None):

> /usr/lib64/python2.7/site-packages/warp/particles/
tunnel_ionization.py(304)generate()
-> for js in self.x[pg]:
(Pdb)
> /usr/lib64/python2.7/site-packages/warp/particles/
tunnel_ionization.py(305)generate()
-> self.flushpart(pg,js)
(Pdb)
-Call-
> /usr/lib64/python2.7/site-packages/warp/particles/
ionization.py(361)flushpart()
-> def flushpart(self,pg,js):
(Pdb)
> /usr/lib64/python2.7/site-packages/warp/particles/
ionization.py(363)flushpart()
-> if self.nps[pg][js]>0:
```

```
(Pdb)
-Return-
> /usr/lib64/python2.7/site-packages/warp/particles/
ionization.py(363)flushpart()->None
-> if self.nps[pg][js]>0:
(Pdb)
> /usr/lib64/python2.7/site-packages/warp/particles/
tunnel_ionization.py(306)generate()
-> processlostpart(pg,js+1,top.clearlostpart,top.time,top.zbeam)
```

Multigrid solver that handles Boltzmann electrons

Attempts to use Boltzmann distributed electrons were made in this work and a mismatch of the arguments of the field solver that handles Boltzmann distributed electrons was found. Form the Python class MultiGrid3D, the function call

```
multigridbe3dsolve(iwhich, self.nx, self.ny, self.nz,
                   self.dx, self.dy, self.dz*zfact,
                   self.potential, self.source,
                   rstar, self.linbend, self.bounds,
                   self.xmmin, self.ymmin, self.zmmin*zfact,
                   self.mgparam, mgiters, self.mgmaxiters,
                   self.mgmaxlevels, mgerror, self.mgtol, mgverbose,
                   self.downpasses, self.uppasses,
                   self.lcndbndy, self.laddconductor, self.icndbndy,
                   f3d.gridmode, conductorobject,
                   iondensitygrid3d,
                   self.fsdecomp)
```

is made to a Fortran script f3d_mgrid_be.F. The call was received at

```
subroutine multigridbe3dsolve(iwhich,nx,ny,nz,
     &                    nxguardphi,nyguardphi,nzguardphi,
     &                    nxguardrho,nyguardrho,nzguardrho,
     &                    dx,dy,dz,phi,rho,
     &                    rstar,linbend,bounds,
     &                    xmmin,ymmin,zmmin,
     &                    mgparam,mgiters,mgmaxiters,
     &                    mgmaxlevels,mgerror,mgtol,mgverbose,
     &                    downpasses,uppasses,
     &                    lcndbndy,laddconductor,icndbndy,
     &                    gridmode,conductors,iondensitygrid3d,
     &                    fsdecomp)
```

It was found that the argument nxguardphi, nyguardphi, nzguardphi, nxguardrho, nyguardrho and nzguardrho were missing in the function call. The developers of Warp were informed about this and this problem is now solved.

Bibliography

[1] D. Li and K. Jousten, "Comparison of some metrological characteristics of hot and cold cathode ionisation gauges", Physikalisch-Technische Bundesanstalt, Tech. Rep., 2003.

[2] K. Jousten, "Pressure measurements with ionisation gauges", Physikalisch-Technische Bundesanstalt, Tech. Rep., 2000.

[3] S. R. Systems. (). Bayard-alpert ionization gauges, [Online]. Available: `http://www.thinksrs.com/downloads/PDFs/ApplicationNotes/IG1BAGapp.pdf`.

[4] A. Berman, *Total pressure measurements in vacuum technology.* Academic Press, 1985.

[5] H. D.Young and R. A. Freedman, *University physics.* Pearson, 2014.

[6] J. H. Leck, *Total and partial pressure measurement in vacuum systems.* Springer US, 1989.

[7] D. Grote, J. Vay, A. Friedman, and S. Lund. (). Warp wiki, [Online]. Available: `http://warp.lbl.gov/`.

[8] J. Vay, D. Grote, R. Cohen, and A. Friedman, "Novel methods in the particle in cell accelerator code framework warp", Lawrence Berkeley National Laboratory, Tech. Rep., 2012.

[9] P. Juda, B. Jenninger, P. Chiggiato, and T. Richard, "3d-simulation of ionisation gauges and comparison with measurements", *Vacuum*, vol. 138, 2016.

[10] D. J. Griffiths, *Introduction to electrodynamics.* Prentice Hall, 1999.

[11] W. Möller, *Fundamentals of plasma physics*, Summer 2006.

[12] K. Nishikawa and M. Wakatani, *Plasma physics.* Springer, 2000.

[13] K. Wiesemann, "A short introduction to plasma physics", 2014.

[14] J. R. Roth, *Industrial plasma engineering.* Institute of physics publishing, 1995.

[15] N. W. Ashcroft and N. D. Mermin, *Solid state physics.* Brooks/Cole, Cengage learning, 1976.

[16] L. Particle In Cell Consulting. (). Particleincell.com, [Online]. Available: `https://www.particleincell.com/`.

[17] F. F. Chen, *Introduction to plasma physics and controlled fusion.* Springer, 2016.

[18] J. D. Jackson, *Classical electrodynamics.* John Wiley & sons, 1962.

[19] J. P. Verboncoeur, "Particle simulation of plasmas: Review and advances", Department of nuclear engineering, University of California, Berkely, Tech. Rep., 2004.

[20] (). Encyclopedia of mathematics, [Online]. Available: `www.encyclopediaofmath.org`.

[21] L. Edsberg, *Introduction to computation and modeling for differential equations.* John Wiley & sons, 2016.

[22] G. Lapenta, "Particle in cell method- a brief description of the pic method", Centrum voor Plasma Astrofysica, Katholieke Universiteit Leuven, Tech. Rep.

[23] R. W. Hockney and J. W. Eastwood, *Computer simulations using particles.* McGraw-Hill Inc., 1981.

[24] B. Wang, S. Ethier, W. Tang, K. Z. Ibrahim, K. Madduri, S. Williams, and L. Oliker, "Modern gyrokinetic particle-in-cell simulation of fusion plasmas on top supercomputers", 2015.

[25] C. K. Birdsall and A. B. Langdon, *Plasma physics via computer simulation.* IOP Publishing Ltd, 1991.

[26] D. P. Grote. (). Warp manual, [Online]. Available: `http://hifweb.lbl.gov/Warp/manual/manual.html`.

[27] J. Vay, D. Grote, R. Cohen, A. Friedman, S. Lund, W. Sharp, and I. H. R. Kishek, "Computational methods in the warp code framework for kinetic simulations of particle beams and plasmas", Tech. Rep., 2014.

[28] S. M. Lund and R. L. Jean-Luc Vay. (). Electromagnetic particle-in-cell codes, [Online]. Available: `https://people.nscl.msu.edu/~lund/uspas/scs_2016/lec_adv/A1a_EM_PIC.pdf`.

[29] S. Lund, *Intro. lecture 04: Numerical methods for particle and distribution methods: Introduction to the pic method*, 2016.

[30] P. Bourke. (). Http://paulbourke.net/, [Online]. Available: `http://paulbourke.net/miscellaneous/interpolation/`.

[31] M. Hanke, *Structured grids*, 2016.

[32] M. Schatzman, *Numerical analysis a mathematical introduction.* Oxford University Press, 1991.

[33] T. Tajima, *Computational plasma physics: With applications to fusion and astrophysics.* Westview press, 2004.

[34] S. E. Parker, C. K. Birdsall, “Numerical Error in Electron Orbits with Large $\omega_{ce}\Delta t$”, Tech. Rep., 1991.

[35] (). Generate pseudo-random numbers, [Online]. Available: `https://docs.python.org/2/library/random.html`.

[36] S. Lund, *Intro. lecture 01: Overview*, 2016.

[37] L. Bertil Westergren, *Mathematics handbook for science and engineering*, fifth. Studentlitteratur, 2004.

[38] D. S. Bernstein, *Matrix mathematics*, second. Princeton University Press, 2009.

[39] D. Grote. (). Warp installation manual online, [Online]. Available: `http://warp.lbl.gov/home/how-to-s/installation`.

TRITA -SCI-GRU 2018:001