NTNU

Norwegian University of
Science and Technology

A Three-Dimensional Finite Element Poisson Solver for Monte Carlo Particle Simulators

Siri Narvestad Fatnes

Master of Science in Physics and Mathematics
Submission date: July 2018
Supervisor: Helge Holden, IMF
Co-supervisor: Trond Brudevoll, FFI

Norwegian University of Science and Technology
Department of Mathematical Sciences

In memory of my beloved mother,
who always offered support and encouragement

Sammendrag

En tredimensjonal ligningsløser for Poissons ligning, basert på en endelig element-metode, er implementert i Fortran for beregning av det elektriske feltet i en eksisterende Monte Carlo-simulator for partikkeltransport i transistorer. Feltberegningen er avhengig av partiklenes posisjon i det gitte tidssteget, og robuste og effektive algoritmer for punktlokasjon i ustrukturerte to- og tredimensjonale gitter er implementert. Disse benyttes i forbindelse med partikkelinjeksjon ved Ohmske kontakter, og for å regne ut partiklenes bidrag til systemets lastvektor. Det lineære ligningssystemet som oppstår ved å diskretisere Poissons ligning med endelige elementer løses med en metode basert på konjugerte gradienter, forbehandlet med en ufullstendig LU-faktorisering med null innfylling. Denne ligningsløseren har overlegen konvergenstid sammenlignet med to andre ligningsløsere testet på det samme systemet. Egne klasser for lagring av glisne matriser er implementert og sømløst integrert med de lineære løserne. Arbeidet har resultert i en ny programstruktur kalt MCFEM. Programvaren blir testet ved å utføre korte simuleringer av en avalanche fotodiode (APD) med negativ forspenning på ulike gittertettheter. Både lineære og kvadratiske basisfunksjoner blir benyttet.

Abstract

A finite element Poisson solver for the calculation of the three-dimensional electric field in self-consistent particle simulations has been implemented in Fortran and integrated into an existing Monte Carlo simulator for particle transport. Robust and efficient algorithms for point location in unstructured two and three-dimensional grids is implemented, used in a new injection routine adapted to unstructured meshes and for assembling the load vector of the finite element system. The linear system arising from the finite element approximation of the Poisson equation is solved using the conjugate gradient method preconditioned with an incomplete LU-factorization, which outperforms two other tested solvers. Special storage schemes are implemented to construct and store the matrices of the linear system and are seamlessly combined with the linear system solvers. The work has resulted in a new program structure called Monte Carlo software with finite element Poisson solver (MCFEM), which has been tested by performing short bias simulations of avalanche photodiodes on different grid refinements with the use of both linear and quadratic polynomial basis functions.

Preface

This Master's Thesis is submitted to the Norwegian University of Science and Technology (NTNU), and completes my Master's Degree in Industrial Mathematics at the Department of Mathematical Sciences (IMF). The thesis is written in cooperation with the Norwegian Defence Research Establishment (FFI), under the supervision of Trond Brudevoll and Asta Storebø. The work has been carried out during the spring semester 2018, and has official subject title *TMA4900 Industrial Mathematics, Master's Thesis*.

My supervisors Trond and Asta at FFI deserves sincere acknowledgment, always welcoming me to Kjeller and answering my long emails. I hope they will see value in my contribution. Professor Helge Holden has been my supervisor at IMF, and I am grateful for the help and encouragement he has provided during the last year.

Siri Narvestad Fatnes,
July 2nd, 2018

Table of Contents

List of Abbreviations

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
APD	avalanche photodiode
BiCG-Stab	bi-conjugate gradient stabilized method
CBF	Chordá's, Blasco's and Fueyo's algorithm
CG	conjugate gradient method
CPU	central processing unit
CRS	compressed row storage
DOK	dictonary of keys
FEM	finite element method
FFI	the Norwegian Defence Research Establishment
FP	face-to-point algorithm
FTI	face-trajectory-intersection test
GS	Guiba's and Stolfi's algorithm
GUI	graphical user interface
HgCdTe	mercury cadmium telluride
ILU0	incomplete LU-factorization with zero fill-in
MC	Monte Carlo
MCFEM	Monte Carlo software with finite element Poisson solver
MCS	Monte Carlo software
OC	Ohmic contact
PCG	preconditioned conjugate gradient method
PDE	partial differential equation
PLA	point location algorithm
PTI	particle-to-the-inside test
PTL	particle-to-the-left test
TTL	trajectory-to-the-left test

List of Symbols

A	Stiffness matrix of discretized Poisson equation
$A_{\mathbf{BC}}$	The stiffness matrix A altered with an identity block for imposing Dirichlet conditions
A_{ij}	Entry of matrix A at ith row and jth column
A^{LU}	The LU decomposition of the matrix A, $A^{LU} = L + U$

$\boldsymbol{c}_{\mathbf{CRS}}$ Array containing the column position of non-zero values in a sparse matrix. One of three arrays for saving matrices in the CRS-format

$\boldsymbol{E}$ Electric vector field

E Constant scale for the electric field

$\boldsymbol{E}^*$ Unscaled electric field with physical unit Volt per meter

$F(\cdot)$ The linear functional of the weak formulation of the Poisson equation

$\mathcal{F}_K$ Mapping from an element K to the reference element $\hat{K}$

$\mathcal{F}_K^{-1}$ Mapping from the reference element $\hat{K}$ to an element K

F_i Face of element opposite to node $\boldsymbol{x}_i$

$H^1(\Omega)$ First order Sobolev space on the domain Ω

$H^1_\Gamma(\Omega)$ Subspace of $H^1(\Omega)$, $v \in H^1_\Gamma(\Omega) \implies v \in H^1(\Omega)$ and $v = 0$ on Γ

K Arbitrary element of the triangulation $\mathcal{T}_h$

$\hat{K}$ Reference element

$K_{\mathbf{init}}$ Initial search element for point location

K_p Element containing the point $\boldsymbol{x}_p$

$\{K_j\}_{j=1}^M$ Set of elements in a triangulation $\mathcal{T}_h$

$|K|$ Volume of an element K

L Constant scale for length

L Lower triangular matrix

$L^2(\Omega)$ The space of Lebesgue square integrable functions on the bounded domain Ω

M Number of elements K in the triangulation $\mathcal{T}_h$

N_d Degrees of freedom = Number of nodes = Number of basis functions on each element

N_h Dimension of approximation space, degrees of freedom, total number of basis functions

$\boldsymbol{N}_i$ Node on the reference element

N_i^K Global node number of local node $\boldsymbol{x}_i \in K$

$N_{\mathbf{insert}}$ Number of particles that need to be inserted for neutrality at the Ohmic contact

N_p Total number of particles

N_q Number of quadrature points

N_z Number of non-zero elements in a sparse matrix

Q Constant scale for charge

Q A quadrature rule consisting of a set of weights and points, $\{w_i, \boldsymbol{x}_i^q\}_{i=1}^{N_q}$

$\boldsymbol{r}_{\mathbf{CRS}}$ Array containing the indices into V of the first non-zero entry in each row. One of three arrays for saving matrices in the CRS-format

$\mathbb{R}$ The space of real numbers

$\mathbb{R}^d$ The space of real numbers in d-dimensional space

R_g Lifting function for the inhomogenous Dirichlet boundary condition

$\boldsymbol{S}$ Search trajectory from initial point $\boldsymbol{x}_{\text{init}}$ to $\boldsymbol{x}_p$ in point location

$\mathcal{T}_h$ Triangulation

U Constant scale for potential

U	Upper triangular matrix
$\boldsymbol{v}_{\mathbf{CRS}}$	Array containing the non-zero values of a sparse matrix in sequential order. One of three arrays for saving matrices in the CRS-format
V	Test function space, short notation for $H^1_{\partial\Omega_\mathrm{D}}(\Omega)$
V_h	Finite dimensional space approximating V
$a(\cdot,\cdot)$	The bilinear form of the weak formulation of the Poisson equation
$\boldsymbol{a}_i$	Local vector from initial search point to local edge node for computation of the TTL decision parameter α_i
d	Number of spatial dimensions
f	Right-hand side of Poisson equation
f	Arbitrary scalar function
$\boldsymbol{f}$	Load vector for the discretized Poisson equation, $\boldsymbol{f} = [f_1, \ldots, f_{N_h}]^T$
$\boldsymbol{f}_{\mathbf{BC}}$	Complete right-hand side vector for the discretized Poisson equation altered with zeros in Dirichlet node entries
g	Function imposed on Dirichlet boundary
g_i	Coefficient for the linear combination of basis functions expanding the approximation to the lifting function R_g
$\boldsymbol{n}$	Outwardpointing normal vector of a surface
p	Index of Particle; Particle
q	Dimensionless charge
q_p	Charge of particle p
q^*	Charge with unit Coulomb
u	Electric potential; Scaled, dimensionless potential
u_h	Approximation of the solution u to the Poisson equation
u_i	Coefficient for the linear combination of basis functions expanding the approximation to the inhomogenous Dirichlet problem, u_h
$\mathring{u}_i$	Coefficient for the linear combination of basis functions expanding the approximation to the homogeneous Dirichlet problem, $\mathring{u}_h$
$\mathring{u}$	Potential for homogeneous boundary conditions
$\mathring{u}_h$	Approximation of potential for homogeneous boundary conditions
$\mathring{\boldsymbol{u}}$	Vector containing the coefficients $\mathring{u}_i$ of the approximation $\mathring{u}_h$
u^*	Unscaled electric potential with physical unit Volt
$\boldsymbol{u}$	The vector of coefficients u_i for the appriximation u_h
v	Test function in V
v_h	Test function in approximation space V_h
v_j	Coefficient for the linear combination of basis functions expanding the test function v_h
w_i	Quadrature weights
$\boldsymbol{x}$	Spacial variable in Cartesian coordinates, $\boldsymbol{x} = (x, y, z)$
$\boldsymbol{x}_i$	Node in triangulation $\mathcal{T}_h$

$\boldsymbol{x}_{\mathbf{init}}$	Initial search point for point location, defined as the center of K_{init}
$\{\boldsymbol{x}_i\}_{i=1}^{N_h}$	Set of nodes in a triangulation $\mathcal{T}_h$
$\boldsymbol{x}_p$	Position of charged particle (free or fixed) with index p
$\boldsymbol{x}_i^q$	Quadrature point
$\boldsymbol{x}^*$	Spacial variable with unit meter
Δy	The depth of the neutral region adjacent to an Ohmic contact
Γ	Subset of a boundary $\partial\Omega$ to a domain Ω
Ω	A bounded domain in Euclidian space
$\partial\Omega$	The boundary of Ω
$\partial\Omega_{\text{D}}$	The boundary of Ω with imposed Dirichlet conditions; Ohmic contact surface
$\partial\Omega_{\text{N}}$	The boundary of Ω with imposed Neumann conditions; Free surface
α	Dimensionless scaling parameter for the Poisson equation
α_i	TTL decision parameter
β	Dimensionless scaling parameter for the electric field equation
β_e	PTL decision parameter
δ	The Dirac distribution (the measure giving unit mass to its argument)
δ_{ij}	The Kronecker-δ
ε	Material dependent permittivity, $\varepsilon = \varepsilon_{\text{r}}\varepsilon_{\text{r}}$
ε_0	Permittivity of vacuum
ε_{r}	Material dependent relative permittivity
γ_F	PTI decision parameter
$\hat{\lambda}_i$	Barycentric coordinates on the 3D simplex, $i \in \{0, 1, 2, 3\}$
ω_e	FTI decision parameter
$\hat{\varphi}_i$	Basis function on the reference element $\hat{K}$
φ_i	Basis function corresponding to a global node $\boldsymbol{x}_i \in \mathcal{T}_h$
φ_i^K	Basis function corresponding to a local node $\boldsymbol{x}_i \in K$ for $K \in \mathcal{T}_h$
$\{\varphi_i\}_{i=1}^{N_h}$	Set of basis functions for approximation space V_h
ρ	Total charge density
$\rho_{\mathbf{da}}$	Charge density contribution from doping atoms (background charge)
$\rho_{\mathbf{fc}}$	Charge density contribution from free carriers
$\boldsymbol{\xi}$	Coordinates $\boldsymbol{\xi} = (\xi, \eta.\zeta)$ in a reference coordinate system, i.e. on the reference element $\hat{K}$
$\boldsymbol{\xi}_i^q$	Quadrature points on the reference element
∇^2	The Laplacian operator, $\nabla^2 f = \nabla \cdot (\nabla f)$

List of Figures

1 Introduction

Monte Carlo (MC) particle-based simulations for transport in semiconductor devices has long been a state-of-the-art tool for modeling transistors. As device dimensions decrease, more sophisticated methods for considering quantum effects and particle behavior under influence of high frequencies are included in the simulation programs. Self-consistent simulations require accurate approximations to the electric field taking into account time- and space-dependent variables for each simulated particle. In this thesis, a finite element method (FEM) is implemented and integrated into an existing Monte Carlo software (MCS) for approximating the electric field through the solution of the Poisson equation. The MCS under consideration is developed for research purposes at the Norwegian Defence Research Establishment (FFI), in cooperation with NTNU.

Availability of open source or commercial MCS for device research is limited, and FFI has been developing a Full Band MC simulator in order to gain insight into device physics for development and production, with the necessary framework for a program with long lifetime and extensive possibilities for modifications to comply with present and future requirements. The FFI-program, also referred to as the FFI-MCS, or simply the MCS, has during this work been expanded with robust and exportable blocks containing numerical schemes for equation solving, linear algebra routines and unstructured domain discretization handling.

A short background on Monte Carlo simulation is given in the next section, before some aspects of the electric field update in self-consistent particle simulations are introduced in Section 1.2. Section 1.3 will give a brief overview of the approach and content of this thesis.

1.1 Monte Carlo Simulation

From a mathematical point of view, MC methods are tools to obtain solutions of many different mathematical problems through random number generation. Possible applications range from computational statistics and finance to simulation modeling in industrial engineering and physical processes [44]. The first application of MC methods to the simulation of semiconductor materials was presented in 1966 and has since become a popular and sophisticated tool for the simulation of semiconductor devices [37, 49, 68]. Particle behavior in solid state physics is of statistical nature, and good models can be achieved with the use of MC-methods.

To a certain degree of accuracy, the motion of carriers can be explained by transport equations. An example is particles in electric and magnetic fields that are described through the

Boltzmann transport equation coupled with the Maxwell field equations [68]. An analytic solution to this type of systems are in most cases unachievable, except in extremely simplified, and thus uninteresting, situations. Numerical solutions can be found by a variety of approaches, ranging from fluid models to models based on large ensembles of particles. The semiclassical ensemble MC method for particle transport is a powerful method directly simulating the statistical behavior of the particles. Modeling is often done by applying principles from classical mechanics for carrier displacement, adding randomly generated scattering processes and including some type of quantum corrections. Collecting statistics from such simulations can produce more insight into physical characteristics than any analytic solution to the governing transport equation [38].

Crystal dynamics, band structures, electronic states and stochastic scattering of particles are important building blocks for any MC simulation tool for solid state physics. A semiclassical theory is used to simulate the carrier dynamics, with particles being described by their current position and wave vector. Calculation of the duration of free flights, scattering events, and new particle states are done by drawing random numbers from an applicable stochastic distribution. When simulating specific device geometries, contrary to bulk simulations, the particle trajectories are space-dependent, and the forces acting on the particles from electric and magnetic fields will thus be dependent on the particle positions. Ensembles of particles need to be simulated, and self-consistent updates of the fields are necessary to achieve accurate results. Books by Hess [34], Jacoboni and Lugli [37], Moglestue [49], and Vasileska et al. [68] all provide thorough introductions to the field of device simulation of semiconductors using self-consistent ensemble Monte Carlo methods.

1.2 Electric Field Updates

The main problem of interest in this thesis is the calculation of the electric field in a simulated semiconductor device. If there is no magnetic fields present, the electric field alone determines the force acting on the particles during their free flights. If a quasi-static electric field is assumed, the field can be described by the gradient of the electric potential, approximated by solving the Poisson equation with numerical methods. The most common approach in device simulation is to apply a finite difference scheme, and use particle-mesh coupling schemes, such as cloud-in-cell or nearest-grid-point, to calculate the contribution from the discrete charges to the charge density in the grid nodes [46]. This has also been the previous approach in the FFI-MCS. However, the geometry of transistors has changed dramatically since they were first introduced in the 1950's, and today it is not uncommon with increasingly small devices, measuring only a few nanometres. This introduces new requirements to the numerical solvers used within device simulation, especially with respect to adaptiveness for complicated geometries and considerations of three-dimensional (3D) effects. In such applications, the finite difference method meets its limitations, introducing a need for investigating other methods in relation to

device simulation.

Popular methods for numerically solving partial differential equations (PDEs) on complicated domains are variations of finite element methods (FEMs), of which the main idea is to divide the computational domain into smaller elements and locally compute solutions on each element. The local solutions are then combined to resemble a global approximation on the union of all elements. The elements are often polygons, such as tetrahedra, prisms or hexahedra, and no structuredness is imposed, improving the methods adaptability to advanced domains. The method is well described in most books on numerical methods for differential equations, see e.g. Quarteroni [52, 53]. FEM was introduced in the 1950's to deal with problems of structural mechanics and was first applied to field problems in the 1960's [58]. A general introduction to FEM for electromagnetics can be found in *The Finite Element Method in Electromagnetics* by Jin [40]. Snowden [65] devotes a chapter of his book *Introduction to Semiconductor Device Modelling* to the discussion of FEM as a numerical equation solver for field equations in device simulation. Recent research in the field of FEM coupled to MCS includes a parallel, three-dimensional finite element ensemble Monte Carlo device simulation tool including quantum corrections, presented by a research group at Swansea University [2, 3, 20, 47].

1.3 Approach

In this thesis, a 3D FEM for use in particle-based simulations is developed and successfully integrated into the FFI-MCS, resulting in a new program version, hereafter called the Monte Carlo software with finite element Poisson solver (MCFEM). This work was started by Åsen [5], which implemented a two-dimensional (2D) solution for a planar *pn*-junction. One of the major difficulties when applying unstructured grids in particle simulations is to provide an efficient point location algorithm (PLA), which task is to provide current element positions of particles. The implementation of a 3D-PLA was started in a thesis-preparing specialization project [23] and has now become a robust part of the source code for the MCFEM. The particle tracking problem is a widely considered problem with the use of unstructured grids in fluid dynamics and belongs to the field of Computational Geometry [18].

The new numerical methods are coupled with the old scattering and carrier dynamics routines to form the MCFEM, and tested on bias simulations of an avalanche photodiode (APD) based on a mercury cadmium telluride (HgCdTe). HgCdTe is a common material used for making infrared detectors, and HgCdTe APDs have application in night vision, eye-safe laser systems, and other thermal imaging applications [63, 64], which are important in defense and security. For the Full Band MCS developed at FFI, any material can be simulated by loading corresponding band structures, produced by WIEN2k, into the program [25, 62]. Research on MC-simulations of large-scale devices are sparse, but with the interesting applications of APDs in combination with their performance on speed, sensitivity, and quantum efficiency, simulation of large devices is desirable for development and characterization. Thus, an APD is the model

of choice for testing MCFEM. FFI presented their first Full Band MC Simulation of an APD on the SPIE Defense + Security conference in Anaheim, California in 2017 [66]. This research can hopefully continue with the use of the implemented MCFEM.

This thesis is started with the presentation of the finite element method for the Poisson equation in Chapter 2, ranging from the weak formulation to the arising linear system, with an additional note on existence and uniqueness for the specific problem under consideration. The solution of the sparse linear system arising from the discretization is briefly discussed in Chapter 3, with focus on the preconditioned conjugate gradient method (PCG). The point location problem and the implemented point location algorithms are presented in Chapter 4. Chapter 5 introduces a new method for adapting boundary conditions for carrier dynamics to unstructured grids. The full program flow of MCFEM is summarized in Chapter 6. Simulation results are presented in Chapter 7, which includes a discussion on run-times and workload for the program. Recommended paths for further improvements are suggested in Chapter 8, before the thesis is rounded off with a small conclusion.

2 | Approximation for the Electric Field

The main improvement to the FFI-MCS during this work is the integration of a 3D Poisson solver for electric field updates. It is a new solver replacing the previous 2D solvers, extending the software's capability to simulate devices with advanced and non-planar geometries. The work with applying FEM to the MCS was initialized by Åsen [5]. A solver of Galerkin type implemented on 2D unstructured triangulations with the use of linear basis functions was introduced. An updated solver fully integrated into the MCS software now provides the solution in 3D, with the additional option of using quadratic basis functions. Working with this new solver, an emphasis has been laid on providing a solver which is mainly independent of the existing software, simplifying future development. The use of independent modules makes it possible to export the methods for use in other types of software requiring FEM for numerical approximation. Only a few routines are needed as an interface between the MCS and the new solver.

In this chapter, a summary of the finite element theory necessary to obtain the numerical schemes will be provided. The theory will be concentrated around the 3D case where it is not general. A short presentation of the Poisson equation with boundary conditions arising from device simulation is given in Section 2.1. A dimensionless equivalent obtained through scaling is presented in Section 2.2 for numerical precision. The weak formulation of the problem is presented in Section 2.3. Choices for the discretized solution space and an outline of the derivation of the numerical schemes is given in Section 2.4. The specific assembly routines are presented in Section 2.5. For completeness, a treatment of existence and uniqueness for the Poisson equation with discrete contributions to the charge density can be found in Section 2.6. The presentation of the finite element theory in this section is mainly influenced by Quarteroni's book *Numerical Models for Differential Problems* [53].

2.1 The Electric Field

In particle simulations, a quasi-static electric field can be assumed by freezing the particle positions between small time intervals. A derivation of the Poisson equation for the electric field from the full set of Maxwell equations is given in Appendix B in *Computational Electronics; Semiclassical and Quantum Device Modeling and Simulation* [68]. In this thesis, we focus on

the case where the electric field $\boldsymbol{E}$ can be described by the gradient of the electric potential u as

$$\boldsymbol{E} = -\nabla u, \tag{2.1}$$

where the electric potential solves the Poisson equation. For a known charge distribution ρ, the Poisson equation can be expressed as

$$-\nabla^2 u(\boldsymbol{x}) = \frac{\rho(\boldsymbol{x})}{\varepsilon_\mathrm{r}\varepsilon_0}, \tag{2.2}$$

where ε_r is the material dependent relative permitivity, and ε_0 the permittivity of vacuum. The Laplacian is denoted by ∇^2. The spacial variable $\boldsymbol{x}$ is given in Cartesian coordinates, $\boldsymbol{x} = (x, y, z)$. Although a simplification, the relative permittivity ε_r is assumed constant. For a better physical approximation, the material permittivity $\varepsilon = \varepsilon_\mathrm{r}\varepsilon_0$ should be considered as a function of the electric field. However, this leads to a non-linear Poisson equation, complicating things considerably. The linear approximation will meet the current requirements for physical resemblance. For self-consistent simulations, the charge density ρ must be determined by the position of the free carriers at each time step, in addition to the constant background charge due to doping atoms. As known from elementary electromagnetism, the charge density can be expressed by the δ-distribution,

$$\rho(\boldsymbol{x}) = \rho_\mathrm{fc}(\boldsymbol{x}) + \rho_\mathrm{da}(\boldsymbol{x}) = \sum_{p=1}^{N_p} q_p \delta(\boldsymbol{x} - \boldsymbol{x}_p). \tag{2.3}$$

Here, ρ_fc is the charge density contribution from the free carriers, and ρ_da is the contribution from the background charge (the doping atoms). Further, N_p is the total number of free carriers and doping contributors and q_p is the charge of particle or doping contributor p positioned in $\boldsymbol{x}_p$.

When simulating semiconductor devices, the potential will be restricted to a domain Ω, and Equation (2.2) will be equipped with additional boundary constraints. The most common choice is to model the contact boundaries with Dirichlet conditions, and the free surfaces with a zero Neumann condition [37]. This will correspond to an applied, known potential at the contacts, and no electric field perpendicular to the other artificial surface areas of the device. The mathematical formulation is

$$\begin{aligned} u &= g(\boldsymbol{x}) && \text{on } \partial\Omega_\mathrm{D}, \\ \frac{\partial u}{\partial \boldsymbol{n}} &= 0 && \text{on } \partial\Omega_\mathrm{N}, \end{aligned} \tag{2.4}$$

where $\boldsymbol{n}$ denotes the outward pointing normal vector on the boundary, and $\partial/\partial\boldsymbol{n}$ denotes the normal derivative on the boundary. The metallic contact boundary with imposed Dirichlet condition is denoted by $\partial\Omega_\mathrm{D}$ and the free surface with imposed Neumann condition by $\partial\Omega_\mathrm{N}$. Ideally, the

boundary conditions on the free surface will not influence in considerable amount what happens in the regions of interest near and between the contacts [49]. The imposed boundary conditions are an approximation of physical models, and the choice of these conditions can be justified as described by Hockney and Eastwood [35]. In addition to the Dirichlet and Neumann boundary conditions for the Poisson equation, the particles will also be subjected to boundary conditions for carrier dynamics. These boundary conditions are treated in Chapter 5.

Before moving on to the numerical approximation of the potential u, it is natural to provide a scaled, dimensionless equivalent of Equation (2.2) and the boundary conditions in Equation (2.4), for simplification and greater numerical accuracy. Such a scaling can be done by a simple change of variables, as seen in the following section.

2.2 Scaling

Using the physical dimensions of the problem can lead to numerical errors due to machine precision problems. In addition, keeping variables of an order of magnitude ~ 1 is preferable for well-conditioned systems. Dimensionless systems are the most common in numerical analysis and computations.

To clarify the necessity of the scaling, consider the following small example. When applying FEM to a semiconductor device on a micrometer scale, the device will be divided into smaller elements K, whose volume $|K|$ is an important factor in the linear system arising from the numerical approximation. An element will typically have a volume in the order of magnitude

$$\mathcal{O}(|K|) = (10^{-6}\,\mathrm{m}) \cdot (10^{-6}\,\mathrm{m}) \cdot (10^{-6}\,\mathrm{m}) = 10^{-18}\,\mathrm{m}^3,$$

or less, which is below normal machine precision of 10^{-16}. This can lead to a loss of numerical precision.

To avoid this, a change of variables is applied to obtain a dimensionless equation equivalent to the one presented in Equation (2.2). To perform the scaling, we denote the potential with physical unit Volt by u^*, the scaled, dimensionless potential by u, and a constant potential scale by U. Applying the same convention to the spacial variable $\boldsymbol{x}^*$ with unit meter and the charge q^* with unit Coulomb, we can write

$$\begin{aligned} u^* &= Uu, & [u^*] &= [U] = \mathrm{V}, \\ \boldsymbol{x}^* &= L\boldsymbol{x} = (Lx, Ly, Lz), & [\boldsymbol{x}^*] &= [L] = \mathrm{m}, \\ q^* &= Qq, & [q^*] &= [Q] = \mathrm{C}, \end{aligned} \tag{2.5}$$

where L and Q are constant length and charge scales. This leads to the dimensionless Poisson equation

$$-\nabla^2 u = \alpha \sum_{p=1}^{N_p} q_p \delta(\boldsymbol{x} - \boldsymbol{x}_\mathrm{p}), \qquad \alpha = \frac{Q}{LU\varepsilon_0\varepsilon_\mathrm{r}}, \tag{2.6}$$

obtained by applying the chain rule to the original equation, differentiating the physical variables with respect to their scaled equivalents. In addition, the fact that the 3D unit of the δ-distribution is the cubed inverse of its argument's unit is used. The Laplacian ∇^2 in Equation (2.6) now denotes operation with respect to the scaled variables (x, y, z). The parameter α is dimensionless and can be set to one by an appropriate choice of scales, and q_p simply denotes the sign of the charge for a given particle, i.e., $q_p = -1$ for negative charge and $q_p = 1$ for positive charge. N_p is still the total number of free carriers, donors and acceptors. A corresponding scaling of the boundary conditons is

$$u^*(\boldsymbol{x}^*) = g^*(\boldsymbol{x}^*) \implies u(\boldsymbol{x}) = \frac{1}{U} g^*(\boldsymbol{x}) =: g(\boldsymbol{x}) \tag{2.7}$$

We thus obtain the dimensionless scaled problem

$$\begin{cases} -\nabla^2 u = \displaystyle\sum_{p=1}^{N_p} q_p \delta(\boldsymbol{x} - \boldsymbol{x}_p) & \text{in } \Omega, \\ u = g(\boldsymbol{x}) & \text{on } \partial\Omega_{\mathrm{D}}, \\ \dfrac{\partial u}{\partial \boldsymbol{n}} = 0 & \text{on } \partial\Omega_{\mathrm{N}}. \end{cases} \tag{2.8}$$

The electric field equation given in Equation (2.1) can be scaled in the same manner, letting

$$\boldsymbol{E}^* = E\boldsymbol{E} = (EE_x, EE_y, EE_z), \quad [\boldsymbol{E}^*] = [E] = \mathrm{V\,m^{-1}}. \tag{2.9}$$

The corresponding electric field equation becomes

$$\boldsymbol{E} = -\beta \nabla u, \qquad \beta = \frac{U}{LE}, \tag{2.10}$$

where β is dimensionless and one can choose the scale E such that $\beta \sim 1$. The dimensionless system is equivalent to the physical one by the Buckingham's π theorem [30, 32]. These dimensionless systems are the ones considered for the following formulation of the numerical approximations, and all variables in this thesis are considered dimensionless if not specified otherwise.

2.3 Weak Formulation

Weak and variational formulations of PDEs are a core part of analyzing solutions and developing numerical solution schemes for many types of equations [10, 22, 60]. To derive a finite element approximation to Equation (2.8), we must first establish the weak formulation. This formulation is built partially on Sobolev space theory, and it is thus necessary to introduce two of the most important Sobolev spaces. We include only the most essential, and for a more thorough introduction to Sobolev space theory, the reader is referred to Adams and Fournier

[1].

Let $\Omega \subset \mathbb{R}^d$ be a a bounded domain in d-dimensional Euclidian space. We denote by $L^2(\Omega)$ the space of Lebesgue square integrable functions on Ω,

$$L^2(\Omega) = \left\{ f : \Omega \mapsto \mathbb{R} \text{ s.t. } \int_\Omega (f(\boldsymbol{x}))^2 \mathrm{d}\Omega < \infty \right\}.$$

In our case, the most important Sobolev space is denoted by $H^1(\Omega)$ and is the space consisting of functions belonging to $L^2(\Omega)$ which also have their distributional derivatives of first order in $L^2(\Omega)$. In addition, the subspace $H^1_\Gamma(\Omega) \subset H^1(\Omega)$ is needed, where an additional constraint imposes function values to zero at a part Γ of the boundary $\partial\Omega$. We define them as

$$\begin{aligned} H^1(\Omega) &= \left\{ f \in L^2(\Omega) \text{ s.t. } \frac{\partial f}{\partial x_i} \in L^2(\Omega), i = 1, \ldots, d \right\} \\ H^1_\Gamma(\Omega) &= \left\{ f \in H^1(\Omega) \text{ s.t. } f = 0 \text{ on } \Gamma \subseteq \partial\Omega \right\}. \end{aligned}$$

We are seeking a solution to the Poisson equation with boundary conditions as formulated in Equation (2.8), and will in the following denote by $f(\boldsymbol{x})$ the distribution on the right-hand side of this equation,

$$f(\boldsymbol{x}) := \sum_{p=1}^{N_p} q_p \delta(\boldsymbol{x} - \boldsymbol{x}_p). \tag{2.11}$$

By multiplying with a smooth test function $v(\boldsymbol{x})$, integrating over the domain Ω and using Greens formula [53], we obtain a first order integral problem of the form

$$\int_\Omega (\nabla u \cdot \nabla v)\, \mathrm{d}\Omega - \int_{\partial\Omega_\mathrm{D}} \left(\frac{\partial u}{\partial \boldsymbol{n}} v \right) \mathrm{d}\gamma - \int_{\partial\Omega_\mathrm{N}} \left(\frac{\partial u}{\partial \boldsymbol{n}} v \right) \mathrm{d}\gamma = \int_\Omega (fv)\, \mathrm{d}\Omega. \tag{2.12}$$

Since the function $f(\boldsymbol{x})$ is a distribution, the right-hand side integral should be understood as the act of f on the test function v. The third term on the left-hand side is zero by the imposed Neumann condition. To obtain the weak formulation, we impose $v \in H^1_{\partial\Omega_\mathrm{D}}$, such that the second term on the left-hand side of Equation (2.12) also evaluates to zero. We require a symmetric weak formulation, meaning that both the test function v and the sought solution u are members of the same space. For a formulation where $u \in H^1_{\partial\Omega_\mathrm{D}}$, Equation (2.8) can be reformulated to a homogeneous Dirichlet problem by imposing a *lifting of the boundary data*. This lifting is a supposed known function $R_g \in H^1(\Omega)$, with $R_g|_{\partial\Omega_\mathrm{D}} = g$ and $\partial_{\boldsymbol{n}} R_g|_{\partial\Omega_\mathrm{N}} = 0$. Here, the notation $\partial_{\boldsymbol{n}}$ denotes the derivative in the direction of the outward normal $\boldsymbol{n}$. An explicit, analytic expression for the lifting function will not be necessary, as we will see when deriving the discretized problem in Section 2.4.

If we let $\mathring{u} = u - R_g$, it can be observed that $\mathring{u}|_{\partial\Omega_\mathrm{D}} = u|_{\partial\Omega_\mathrm{D}} - R_g|_{\partial\Omega_\mathrm{D}} = 0$, and $\mathring{u}$ satisfies

$$\begin{aligned} -\nabla^2 \mathring{u} &= f(\boldsymbol{x}) + \nabla^2 R_g \qquad \text{in } \Omega, \\ \mathring{u} &= 0 \qquad\qquad\qquad\quad \text{on } \partial\Omega_\mathrm{D}, \\ \frac{\partial \mathring{u}}{\partial \boldsymbol{n}} &= 0 \qquad\qquad\qquad\quad \text{on } \partial\Omega_\mathrm{N}. \end{aligned} \tag{2.13}$$

Equation (2.13) can be reformulated to the first order integral problem

$$\int_\Omega (\nabla \mathring{u} \cdot \nabla v)\,\mathrm{d}\Omega = \int_\Omega (f(\boldsymbol{x}) v(\boldsymbol{x}))\,\mathrm{d}\Omega - \int_\Omega \Big(\nabla R_g \cdot \nabla v\Big)\,\mathrm{d}\Omega \tag{2.14}$$

Denoting by V the Sobolev space $H^1_{\partial\Omega_\mathrm{D}}$, we can define the form $a(\cdot,\cdot)$

$$a : V \times V \mapsto \mathbb{R}, \qquad\qquad a(u,v) = \int_\Omega (\nabla u \cdot \nabla v)\,\mathrm{d}\Omega,$$

and the functional $F(\cdot)$

$$F : V \mapsto \mathbb{R}, \qquad F(v) = \int_\Omega (f(\boldsymbol{x}) v(\boldsymbol{x}))\,\mathrm{d}\Omega - \int_\Omega \Big(\nabla R_g \cdot \nabla v\Big)\,\mathrm{d}\Omega.$$

The weak formulation of problem (2.13) can be expressed as

$$\text{find } \mathring{u} \text{ in } V \quad : \quad a(\mathring{u}, v) = F(v) \quad \forall \quad v \in V. \tag{2.15}$$

The existence and uniqueness of solutions to weak formulations, such as the one presented above, can be proven be applying the Lax-Milgram theorem if the form $a(\cdot,\cdot)$ is bilinear, continuous and coercive, and the functional $F(\cdot)$ is linear and continuous [53]. However, due to the δ-distribution, the functional $F(\cdot)$ is not continuous, and Lax-Milgram cannot be directly applied. In this case, proving existence and uniqueness requires more care, and we postpone a further discussion until Section 2.6. An approximation to the weak formulation (2.15) can be found by searching in a smaller space approximating V. This is the approach in the following section.

2.4 The Finite Element Approximation

We will search for an approximate solution u_h in a finite dimensional subspace $V_h \subset V$, where V_h can be represented by a finite set of basis functions,

$$V_h = \mathrm{span}\{\varphi_i\}_{i=1}^{N_h}, \quad N_h < \infty. \tag{2.16}$$

The *Galerkin problem* is an approximation to the weak formulation in Equation (2.15), and can be expressed as

$$\text{find } u_h \in V_h \quad : \quad a(u_h, v_h) = F(v_h) \quad \forall \quad v_h \in V_h. \tag{2.17}$$

Any function $v_h \in V_h$ can be written as a linear combination of the basis functions,

$$v_h(\boldsymbol{x}) = \sum_{j=1}^{N_h} v_j \varphi_j(\boldsymbol{x}), \quad v_j \in \mathbb{R}.$$

Inserting such approximations for $\mathring{u}$ and v in equation (2.14), we obtain

$$\sum_{i=1}^{N_h} \sum_{j=1}^{N_h} \mathring{u}_i v_j \int_\Omega \nabla\varphi_i \nabla\varphi_j \mathrm{d}\Omega = \sum_{j=1}^{N_h} v_j \left(\int_\Omega f(\boldsymbol{x})\varphi_j(\boldsymbol{x})\mathrm{d}\Omega - \int_\Omega \nabla R_g \cdot \nabla\varphi_j \mathrm{d}\Omega \right). \tag{2.18}$$

Equation (2.18) is a linear system, of which appaerance will depend on the choice of the approximation space V_h. In this work, the space of choice is the one leading to a Galerkin finite element method. It is built by partitioning the domain Ω into smaller elements and defining a polynomial basis with local support on each such element. The partition of Ω is called a triangulation and is denoted by $\mathcal{T}_h$. This triangulation is defined by a set of nodes $\{\boldsymbol{x}_i\}_{i=1}^{N_h}$ and a set of elements, $\{K_j\}_{j=1}^{M}$. The number of nodes, N_h, coincides with the dimension of the approximation space V_h, i.e, the number of basis functions. The total number of elements in the grid is denoted by M. In our case, the elements of $\mathcal{T}_h$ will be restricted to non-overlapping tetrahedra, but other choices of elements are also possible. The set of vertices of the tetrahedra is a subset of the total set of nodes in the triangulation. The union of all elements will constitute the domain Ω,

$$\Omega = \bigcup_{K \in \mathcal{T}_h} K. \tag{2.19}$$

As an illustation, Figure 2.1 shows two typical decompostion of a square box in 3D, constructed with the meshing software GMSH [24].

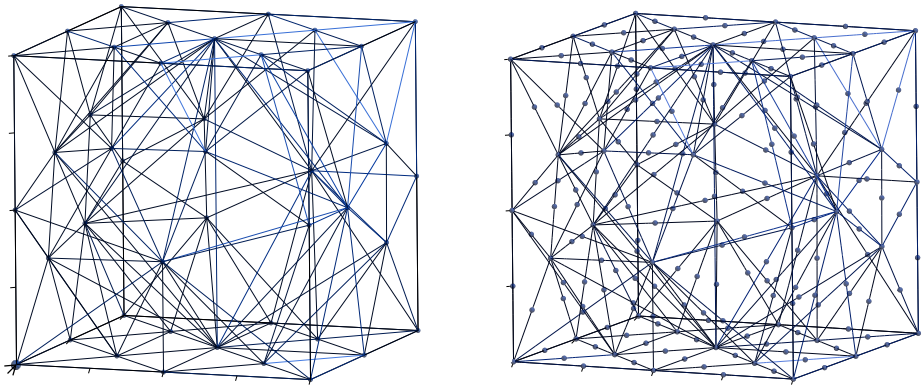
The set of basis functions $\{\varphi_i\}_{i=1}^{N_h}$ is constructed by polynomials satisfying, for any node $\boldsymbol{x}_j \in \{\boldsymbol{x}_i\}_{i=1}^{N_h}$,

$$\begin{aligned} \varphi_i(\boldsymbol{x}_j) &= \delta_{ij}, \\ \operatorname{supp}\{\varphi_i\} &= \{K \in \mathcal{T}_h : K \ni \boldsymbol{x}_i\}, \end{aligned} \tag{2.20}$$

where δ_{ij} denotes the Kronecker-δ,

$$\delta_{ij} = \begin{cases} 1 & \text{for } i = j, \\ 0 & \text{for } i \neq j. \end{cases} \tag{2.21}$$

This choice leads the coefficients u_i of the approximate solution u_h to be the approximated function value in node $\boldsymbol{x}_i$.



(a) (b)

Figure 2.1: A triangulation of a square box with (a) elements with 4 nodes per element at tetrahedra vertices and (b) elements with 10 nodes per element at vertices and edges.

Before Equation (2.18) can be rewritten to a linear system, the term arising from the use of the lifting function R_g must be specified. The function can be approximated as a linear combination of basis functions,

$$R_g \approx \sum_{i=1}^{N_h} g_i \varphi_i(\boldsymbol{x}), \tag{2.22}$$

with the coefficients $g_i \in \mathbb{R}$ defined as

$$g_i = \begin{cases} g(\boldsymbol{x}_i), & \boldsymbol{x}_i \in \partial\Omega_{\mathrm{D}} \\ 0, & \text{else.} \end{cases} \tag{2.23}$$

The *stiffness matrix* of the finite element linear system is defined as the matrix A with entries A_{ij} defined by the *stiffness integral*,

$$A_{ij} = a(\varphi_j, \varphi_i) = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \mathrm{d}\Omega, \tag{2.24}$$

and the *load vector* as the the vector $\boldsymbol{f} = [f_1, ..., f_{N_h}]^T$ with components

$$\begin{aligned} f_i &= \int_\Omega f(\boldsymbol{x})\varphi_i(\boldsymbol{x}) d\Omega = \int_\Omega \left(\sum_{p=1}^{N_p} q_p \delta(\boldsymbol{x} - \boldsymbol{x}_p)\varphi_i(\boldsymbol{x}) \right) \mathrm{d}\Omega \\ &= \sum_{p=1}^{N_p} q_p \varphi_i(\boldsymbol{x}_p). \end{aligned} \tag{2.25}$$

Collecting the coefficients of the linear combination of basis functions for the test function v_h,

the approximation to the homogeneous problem $\mathring{u}_h$ and the lifting function R_g, in vectors

$$\begin{aligned} \boldsymbol{v} &= [v_1, ..., v_{N_h}]^T, \\ \mathring{\boldsymbol{u}} &= [\mathring{u}_1, ..., \mathring{u}_{N_h}]^T, \\ \boldsymbol{g} &= [g_1, ..., g_{N_h}]^T, \end{aligned} \tag{2.26}$$

Equation (2.18) can be written as

$$\begin{aligned} \boldsymbol{v}^T A\mathring{\boldsymbol{u}} &= \boldsymbol{v}^T(\boldsymbol{f} - A\boldsymbol{g}) \quad \forall \boldsymbol{v} \in \mathbb{R}^{N_h} \\ \implies \quad A\mathring{\boldsymbol{u}} &= \boldsymbol{f} - A\boldsymbol{g}. \end{aligned} \tag{2.27}$$

To impose the homogeneous Dirichlet condition on $\mathring{\boldsymbol{u}}$, the rows and columns in the left hand side matrix A corresponding to Dirichlet boundary nodes are replaced with zeros on the off-diagonal and ones on the diagonal, resulting in a new matrix A_{BC} containing an identity block. Further, the entries of the complete vector $\boldsymbol{f} - A\boldsymbol{g}$ on the right hand side corresponding to Dirichlet nodes is set to zero, defining a new vector $\boldsymbol{f}_{\text{BC}}$ with entries

$$f_{\text{BC},i} = \begin{cases} 0 & \text{if } \{i : \boldsymbol{x}_i \in \partial\Omega_{\text{D}}\} \\ f_i - (A\boldsymbol{g})_i & \text{else.} \end{cases} \tag{2.28}$$

The modified linear system can now be written

$$A_{\text{BC}}\mathring{\boldsymbol{u}} = \boldsymbol{f}_{\text{BC}}. \tag{2.29}$$

When solving this system, $\mathring{\boldsymbol{u}}$ will have entires $\mathring{u}_i = 0$ for $\boldsymbol{x}_i \in \partial\Omega_{\text{D}}$. The vector

$$\boldsymbol{u} = \mathring{\boldsymbol{u}} + \boldsymbol{g} \tag{2.30}$$

will then be the collection of coefficients for the approximation u_h expressed in terms of the basis functions, approximately solving Equation (2.8). The system altered for the Dirichlet boundary has a unique solution since the matrix A_{BC} is symmetric positive definite [53]. The stiffness matrix A is sparse due to the local support property of the basis functions, and the system in Equation (2.29) can be solved by a suitable iterative method, as discussed in Section 3. But before Equation (2.29) can be solved, the stiffness integral must be computed in order to assemble the stiffness matrix, which require that we define the basis functions.

Basis functions with local support make it sufficient to define the functions on a *reference element* and use a suitable mapping for evaluation on a general element in the triangulation. The reference element is chosen as the unitary simplex shown in Figure 2.2. It will be denoted by $\hat{K}$. The coordinates in the reference system is denoted by $\boldsymbol{\xi} = (\xi, \eta, \zeta)$, and the nodes on the reference element with $\boldsymbol{N}_i$. The number of nodes on the reference element depends on the order of the element and is denoted by N_d.

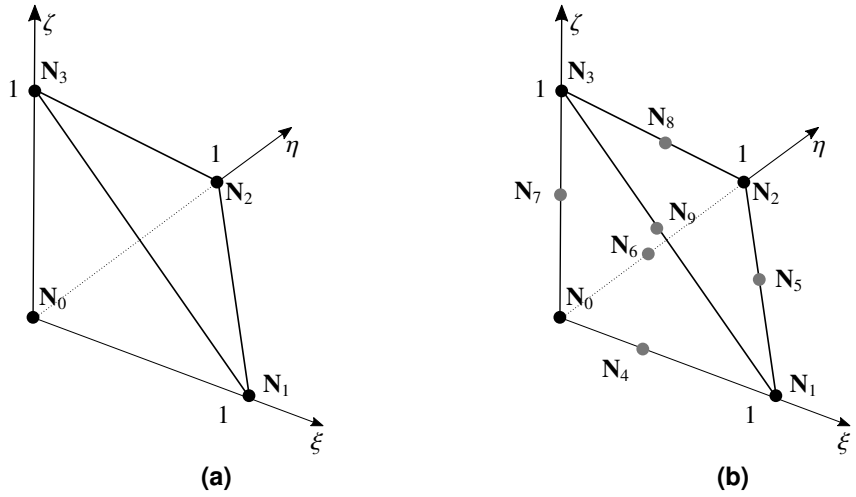


Figure 2.2: The unitary simplex as the 3D reference element. Figure (a) shows the linear element with 4 nodes and Figure (b) the quadratic element with 10 nodes.

Before continuing to define the basis functions, an introduction to an alternative coordinate system is presented. This is the *barycentric* coordinate system, first introduced by August Ferdinand Möbius in 1827 [67]. This system proves useful when defining polynomial basis functions and mappings between the reference element and a general element.

2.4.1 Barycentric Coordinates

As Quarteroni [53], we define the barycentric coordinates on the reference element in 3D as

$$\begin{aligned} \hat{\lambda}_0 &= 1 - \xi - \eta - \zeta, \\ \hat{\lambda}_1 &= \xi, \\ \hat{\lambda}_2 &= \eta, \\ \hat{\lambda}_3 &= \zeta. \end{aligned} \tag{2.31}$$

The coordinates are constructed such that the value of $\hat{\lambda}_i$ is one in vertex $\boldsymbol{N}_i$ on the reference element, and zero in the other vertices, $\boldsymbol{N}_j$, $j \neq i$, $i, j \in \{0, \ldots, 3\}$, given the node numbering as in Figure 2.2.

The barycentric coordinates of an arbitrary point $\boldsymbol{x} \in \Omega$ with respect to an arbitrary element $K \in \mathcal{T}_h$, with vertices $\boldsymbol{x}_i$, $i = 0, \ldots, 3$ can be calculated as

$$\lambda_i(\boldsymbol{x}) = 1 - \frac{(\boldsymbol{x} - \boldsymbol{x}_i) \cdot \boldsymbol{n}_i}{(\boldsymbol{x}_j - \boldsymbol{x}_i) \cdot \boldsymbol{n}_i}, \ 0 \leq i \leq 3 \tag{2.32}$$

The vector $\boldsymbol{n}_i$ denotes the outward pointing normal of the face F_i opposite to node $\boldsymbol{x}_i$, and $\boldsymbol{x}_j \neq \boldsymbol{x}_i$ is *any* of the other vertices belonging to K. The following properties hold for the barycentric

coordinates with respect to an element K,

$$\begin{aligned}
&\sum_{i=0}^{3} \lambda_i(\boldsymbol{x}) = 1, \\
&\lambda_i(\boldsymbol{x}_j) = \delta_{ij}, && i, j \quad \in \{0, \ldots, 3\}, \\
&\lambda_i(\boldsymbol{x}) = 0, \quad \forall \boldsymbol{x} \in F_i, && i \quad \in \{0, \ldots, 3\}, \\
&\lambda_i(\boldsymbol{x}) \geq 0, \quad \forall \boldsymbol{x} \in K, && i \quad \in \{0, \ldots, 3\}.
\end{aligned} \tag{2.33}$$

Given a set of barycentric coordinates $\boldsymbol{\lambda} = [\lambda_0, ..., \lambda_3]^T$ on K, the mapping from the barycentric coordinates and back to the cartesian coordinate system is simply

$$\boldsymbol{x} = \sum_{i=0}^{3} \lambda_i \boldsymbol{x}_i. \tag{2.34}$$

The barycentric coordinates can be defined similarly in the 2D-case. We are now ready to define the set of basis functions which will be used for the numerical approximation.

2.4.2 Polynomial Basis Functions

Let us denote by $\mathbb{P}_r$ the space of polynomials up to order r defined in $\mathbb{R}^3$. The polynomial spaces which will be considered are the linear space $\mathbb{P}_1$,

$$\mathbb{P}_1 = \{p(\boldsymbol{x}) = a + bx + cy + dz;\ a, b, c, d \in \mathbb{R}\},$$

and the quadratic space $\mathbb{P}_2$,

$$\mathbb{P}_2 = \{p(\boldsymbol{x}) = a + bx + cy, +dz + exy + fxz + gyz + hx^2 + iy^2 + jz^2;\ a, .., j \in \mathbb{R}\}.$$

The number of coefficients is the associated degree of freedom, or dimensionality, for each polynomial space. The dimensionality tells how many values of the polynomial that needs to be known in order to uniquely define a polynomial in the given space. The space $\mathbb{P}_1$ has dimensionality four, and $\mathbb{P}_2$ has dimensionality ten. In other words, it is the number of nodes needed on each element to define the basis functions. As on the reference element, the number of nodes is denoted by N_d.

Linear basis functions on each element are a common choice, and is also used by Aldegunde and Kalna [3] and Aldegunde et al. [4] in their work with FEM for MC simulations of semiconductors. Here, the discussion is extended to also include quadratic polynomials. The primary reason for this is the physical interpretation of the approximate solution. For the linear elements, the potential will be non-smooth on the element boundaries, and the resulting electric field will be constant in each element and discontinuous on interior element boundaries. Discontinuities in the electric field are equivalent to surface charges. In the simulations, allowing

for surface charges in the device volume is unphysical and contributes to noise. Upgrading to quadratic elements will lead to a smooth potential also at element boundaries, and the resulting electric field will be continuous. In addition, quadratic elements are found to yield better results for other applications when also taking into account computational resources [21, 36, 61]. Comparisons of simulations with linear and quadratic basis functions can be found in Chapter 7, where the performance of both choices is presented.

When we in the following present the local basis functions, the notation φ_i represents a basis function corresponding to a node $\boldsymbol{x}_i$, satisfying Equation (2.20), with $i \in \{1, \ldots, N_h\}$, while φ_i^K is the global basis function corresponding to a local node $\boldsymbol{x}_i \in K$ for $K \in \mathcal{T}_h$, with $i \in \{0, \ldots, N_d - 1\}$. The basis functions on the reference element is denoted with a hat, $\hat{\varphi}_i$, with $i \in \{0, \ldots, N_d - 1\}$. The linear set of basis functions coincides with the barycentric coordinates,

$$\hat{\varphi}_i = \hat{\lambda}_i, \qquad i = 0, ..., N_d - 1, \tag{2.35}$$

where we recall that $N_d = 4$. In order to provide an unambiguous representation of the quadratic basis functions, the functions are stated explicitly for each node on the reference element in correspondence with the node numbering given in Figure 2.2. They can be expressed in terms of the barycentric coordinates as

$$\begin{aligned}
\hat{\varphi}_0 &= \hat{\lambda}_0(2\hat{\lambda}_0 - 1), & \hat{\varphi}_5 &= 4\hat{\lambda}_1\hat{\lambda}_2, \\
\hat{\varphi}_1 &= \hat{\lambda}_1(2\hat{\lambda}_1 - 1), & \hat{\varphi}_6 &= 4\hat{\lambda}_0\hat{\lambda}_2, \\
\hat{\varphi}_2 &= \hat{\lambda}_2(2\hat{\lambda}_2 - 1), & \hat{\varphi}_7 &= 4\hat{\lambda}_0\hat{\lambda}_3, \\
\hat{\varphi}_3 &= \hat{\lambda}_3(2\hat{\lambda}_3 - 1), & \hat{\varphi}_8 &= 4\hat{\lambda}_2\hat{\lambda}_3, \\
\hat{\varphi}_4 &= 4\hat{\lambda}_0\hat{\lambda}_1, & \hat{\varphi}_9 &= 4\hat{\lambda}_1\hat{\lambda}_3.
\end{aligned} \tag{2.36}$$

To calculate the stiffness matrix A and the load vector $\boldsymbol{f}$ using the reference basis functions, an invertible mapping $\mathcal{F}_K$ from a general element K to the reference element $\hat{K}$ is necessary. For linear elements, the natural choice is an affine, linear map. For the quadratic elements, there is a choice between the use of the same affine, linear map or a quadratic map. An affine map results in *affine elements*, a quadratic map in *iso-parametric elements*. A clear advantage with the affine map is a constant Jacobian on each element, simplifying many of the computations. Iso-parametric elements are often preferred where highly irregular domains with curved boundary are discretized using elements with curved edges. Such domains are not considered in this work, and from a computational resources aspect, the most convenient is to use an affine map. In Chapter 7, it will be apparent that an extension to iso-parametric mappings should be considered in the future.

We define the affine mapping $\mathcal{F}_K$ by its inverse, $\mathcal{F}_K^{-1} : \hat{K} \mapsto K$, utilizing the barycentric coordinates. Let $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ be an arbitrary point in the reference element $\hat{K}$. This point can be

mapped to a point $\boldsymbol{x}$ in an element with vertices $\boldsymbol{x}_i$, $i = 0, ..., 3$ as

$$\boldsymbol{x} = \mathcal{F}_K^{-1}(\boldsymbol{\xi}) = \sum_{i=0}^{3} \hat{\lambda}_i(\boldsymbol{\xi})\boldsymbol{x}_i, \tag{2.37}$$

where $\hat{\lambda}_i(\boldsymbol{\xi})$ is given in Equation (2.31). Then the basis function associated to the local node i on an arbitrary element K can be written as

$$\varphi_i^K(\boldsymbol{x}) = \hat{\varphi}_i\left(\mathcal{F}_K(\boldsymbol{x})\right) = \hat{\varphi}_i(\boldsymbol{\xi}), \;\; i = \{0, \cdots, N_d - 1\}. \tag{2.38}$$

The gradient can be calculated as

$$\nabla\varphi_i^K(\boldsymbol{x}) = \nabla\hat{\varphi}_i\left(\mathcal{F}_K(\boldsymbol{x})\right) = J_K^T\nabla_{\xi}\hat{\varphi}_i(\boldsymbol{\xi}), \;\; i = \{0, \cdots, N_d - 1\}, \tag{2.39}$$

where ∇_{ξ} denotes the gradient with respect to the reference coordinates $\boldsymbol{\xi}$ and J_K is the Jacobian of the mapping $\mathcal{F}_K$,

$$J_K = \mathcal{J}(\mathcal{F}_{\mathcal{K}}(\boldsymbol{x})). \tag{2.40}$$

The Jacobian of a general multivariate function $\mathbf{h} : \mathbb{R}^n \mapsto \mathbb{R}^m$is defined here as

$$\mathcal{J}(\mathbf{h}) = \begin{bmatrix} \frac{\partial \mathbf{h}_1}{\partial x_1} & \frac{\partial \mathbf{h}_1}{\partial x_2} & \cdots & \frac{\partial \mathbf{h}_1}{\partial x_n} \\ \frac{\partial \mathbf{h}_2}{\partial x_1} & \frac{\partial \mathbf{h}_2}{\partial x_2} & \cdots & \frac{\partial \mathbf{h}_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{h}_m}{\partial x_1} & \frac{\partial \mathbf{h}_m}{\partial x_2} & \cdots & \frac{\partial \mathbf{h}_m}{\partial x_n} \end{bmatrix}. \tag{2.41}$$

For practical reasons, J_K is computed using the inverse mapping,

$$J_K = \left(\mathcal{J}\left(\mathcal{F}^{-1}(\boldsymbol{\xi})\right)\right)^{-1}. \tag{2.42}$$

With these definitions and tools, we can proceed to the assembly procedures for the stiffness matrix and the load vector.

2.5 Assembly Procedures

To compute the coefficients of the approximate solution u_h, the elements of the stiffness matrix A and the load vector $\boldsymbol{f}$ must be calculated. Let N_i^K denote the global node number of the local node $\boldsymbol{x}_i$ on element K, where $i = 0, \cdots, 3$ in the linear case and $i = 0, \cdots, 9$ in the

quadratic case. Then

$$\begin{aligned} A_{N_i^K N_j^K} &= \int_\Omega \nabla\varphi_{N_i^K} \cdot \nabla\varphi_{N_j^K} \mathrm{d}\Omega = \sum_K \int_K \nabla\varphi_i^K \cdot \nabla\varphi_j^K \mathrm{d}\Omega \\ &= \sum_K \int_{\hat{K}} \left[\left(J_K^T \nabla\hat{\varphi}_i \right) \cdot \left(J_K^T \nabla\hat{\varphi}_j \right) \det\left(J_K^{-1} \right) \right] \mathrm{d}\hat{\Omega}, \end{aligned} \tag{2.43}$$

where J_K is the Jacobian defined in Equation (2.42), J_K^T its transpose and J_K^{-1} its inverse. The last integral in Equation (2.43) is an integral over a tetrahedral region, which in general can be approximated by appropriate quadrature rules for such regions, as discussed by Yu [69] and Zhang et al. [70], among others. A quadrature rule Q consists of a set of paired weights and points $\{w_i, \boldsymbol{x}_i^q\}_{i=1}^{N_q}$, and is used for numerical approximation of a general integral as

$$\int_D f(\boldsymbol{x}) \mathrm{d}D \approx |D| \sum_{i=1}^{N_q} w_i f(\boldsymbol{x}_i^q), \tag{2.44}$$

where N_q is the total number of quadrature weight and point pairs, and $|D|$ is the volume of the domain D that we are integrating over. In cases where D is some type of tensor product domain, multidimensional quadrature rules can be constructed by tensor products of one-dimensional (1D) quadrature rules. When integrating over tetrahedral regions, maps from tensor product domains to the tetrahedral region introduce unnecessary numerical errors, and special, symmetric rules for tetrahedral regions have advantages [69]. A quadrature rule of order N is defined to be exact for any polynomial of degree less than or equal to N.

For linear basis functions, the integrand of the stiffness integral in Equation (2.43) is a constant, and any quadrature rule will be exact. In the quadratic case, the stiffness integrand will be a second order polynomial, and so a quadrature rule of order two will yield an exact evaluation of the integral. Yu [69] presents a table with Gaussian quadrature formulas up to order six on the reference element, but it will suffice to use the second order rule with four quadrature points, which is reproduced in Table 2.1. The quadrature points on the reference element is denoted by $\boldsymbol{\xi}_i^q = (\xi_i^q, \eta_i^q, \zeta_i^q)$. Gaussian quadrature rules are usually represented in barycentric coordinates [1], and so the reference coordinates are easily extracted from these rules.

Table 2.1: The second order quadrature rule with four points presented by Yu [69]. Note that the rule is symmetric and is simply a permutation of the barycentric coordinates. Higher floating point accuaracy can be found in the reference.

i	w_i	ξ_i^q	η_i^q	ζ_i^q
1	0.25	0.13819	0.13819	0.13819
2	0.25	0.58541	0.13819	0.13819
3	0.25	0.13819	0.58541	0.13819
4	0.25	0.13819	0.13819	0.58541

[1] In the quadrature literature often referred to as volume coordinates or natural coordinates

Algorithm 2.1 Assembly Procedure for Stiffness Matrix

Input: Triangulation $\mathcal{T}_h$,
Set of basis functions $\{\hat{\varphi}_i\}_{i=0}^{N_d-1}$,
Quadrature rule $Q = \{w_l, \boldsymbol{\xi}_l^q\}_{l=1}^{N_q}$
Output: Stiffness matrix A

for all Elements $K \in \mathcal{T}_h$ **do**
  $J_K^{-1} = \mathcal{J}\left(\mathcal{F}_K^{-1}\right)$
  $J_K^T = \left(J_K^{-1}\right)^{-T}$
  for $i = 0$ to $N_d - 1$ **do**
    for $j = i$ to $N_d - 1$ **do**
      $A_{ij}^K = \frac{1}{6}\det\left(J_K^{-1}\right)\sum_{l=1}^{N_q} w_l\left(J_K^T\nabla\hat{\varphi}_i(\boldsymbol{\xi}_l^q)\right)\cdot\left(J_K^T\nabla\hat{\varphi}_j(\boldsymbol{\xi}_i^q)\right)$
      N_i^K = global node number of local node $\boldsymbol{x}_i$ on K
      N_j^K = global node number of local node $\boldsymbol{x}_j$ on K
      $A_{N_i^K N_j^K} = A_{N_i^K N_j^K} + A_{ij}^K$ {Add local contribution to global matrix}
      if $N_i^K \neq N_j^K$ **then**
        $A_{N_j^K N_i^K} = A_{N_j^K N_i^K} + A_{ij}^K$ {Set symmetric contribution }
      end if
    end for
  end for
end for

The volume of the reference element $\hat{K}$ is ⅙, and the contribution to the global stiffness matrix A from a single element K is

$$A_{ij}^K = \frac{1}{6}\det\left(J_K^{-1}\right)\sum_{q=1}^{n_q} w_q\left(J_K^T\nabla\hat{\varphi}_i(\boldsymbol{\xi}_q)\right)\cdot\left(J_K^T\nabla\hat{\varphi}_j(\boldsymbol{\xi}_q)\right). \tag{2.45}$$

The global matrix can be computed as

$$A_{N_i^K N_j^K} = \sum_{K\in\mathcal{T}_h} A_{ij}^K. \tag{2.46}$$

The resulting algorithm used to assemble the stiffness matrix is presented in Algorithm 2.1.

For the load vector $\boldsymbol{f}$ the entries are defined in Equation (2.25), and the reference mapping gives

$$f_i = \sum_{p=1}^{N_p} q_p\varphi_i(\boldsymbol{x}_p) = \sum_{p=1}^{N_p} q_p\hat{\varphi}_i(\mathcal{F}_{K_p}(\boldsymbol{x}_p)), \tag{2.47}$$

where K_p is the particular element containing particle p. This element is the *element location* of the particle, and can be found by using a point location algorithm (PLA), as will be discussed in Chapter 4. Each particle will only be contained in one element, and thus only contribute to

the entries of the load vector corresponding to nodes in that element. The assembly procedure for the load vector loops over the particles, and adds the contribution from the particle to the respective entries as shown in Algorithm 2.2.

Algorithm 2.2 Assembly Procedure for Load Vector

Input: Triangulation $\mathcal{T}_h$,
Set of basis functions $\{\hat{\varphi}_i\}_{i=0}^{N_d-1}$,
Particle positions $\{\boldsymbol{x}_p\}$ and their element locations $\{K_p\}$
Output: Load vector $\boldsymbol{f}$

for all Particles p in simulation **do**
  $\boldsymbol{\xi}_p = \mathcal{F}_{K_p}(\boldsymbol{x}_p)$
  for $i = 0$ to $N_d - 1$ **do**
    $N_i^{K_p}$ = global node number of local node i on K_p
    $f_{N_i^K} = f_{N_i^K} + q_p \hat{\varphi}_i(\boldsymbol{\xi}_p)$
  end for
end for

The solution vector $\boldsymbol{u}$ defined in Equation (2.30) represents the approximation of the potential in each node on the domain. To obtain the electric field for each particle, an approximation of the gradient of the potential must be computed in each particle position. The simplest and most intuitive calculation of the electric field for a particle in a position $\boldsymbol{x}_p$ is done by exploiting the basis functions,

$$\boldsymbol{E}_h(\boldsymbol{x}_p) = -\nabla u_h(\boldsymbol{x}_p) = -\sum_{i=1}^{N_h} u_i \nabla \varphi_i(\boldsymbol{x}_p) = -\sum_{i=1}^{N_d} u_{N_i^K} \boldsymbol{J}_K^T \nabla_{\xi} \hat{\varphi}_i \left(\mathcal{F}_{K_p}(\boldsymbol{x}_p)\right), \tag{2.48}$$

where $\boldsymbol{E}_h$ denotes the approximation to the electric field $\boldsymbol{E}$. This interpolation scheme for the electric field is a scheme arising naturally from the use of local support basis functions. If the linear basis functions are utilized, the electric field is constant on each element, which is a rough approximation if one considers a coarse grid and a single particle placed in a volume with constant permittivity. This particle would feel what is known as *self-forces*. The electric field arising from the particle would act with a force on that particle, which is unphysical. However, as discussed by Aldegunde and Kalna [3], the effect of self-forces are minimal under high voltage simulations with many particles, which is the considered type of simulation in this thesis. Thus, time-saving measures have in this case been weighted higher than reducing these artificial self-forces, resulting in the use of the simple scheme in Equation (2.48). For a presentation and comparison of different interpolation schemes, the reader is referred to Aldegunde et al. [4].

Before moving on to the solution of the linear system in Chapter 3 and the point location problem in Chapter 4, we will take a closer look of the well-posedness of the problem defined in Equation (2.8).

2.6 Existence and Uniqueness

The existence and uniqueness of the solution to the weak formulation in Equation (2.15) is, as previously stated, only ensured if the functional $F(v)$ is linear and continuous. The right-hand side, arising from letting the δ-distribution act on the smooth test functions, is not continuous, and the Lax-Milgram existence theorem cannot be applied. It is however mathematically interesting to provide an alternative proof for the existence and uniqueness of the solution to our problem. We will return to the original problem as stated in Equation (2.8), and for simplicity assume $q_p = 1$ for all particles. We will analyze the equation in the setting of distributions and Green function theory, providing the minimal set of definitions and theorems necessary to prove the existence of a unique solution.

A possible workaround can also be to substitute the δ-distribution with a smooth, bounded Gaussian distribution, and consider this approximate problem. Since $a(\cdot, \cdot)$ is, in fact, bilinear, continuous and coercive, and the Gaussian approximation would belong to $L^2(\Omega)$, Lax-Milgram could be applied, and the weak formulation would be well posed. This approach is not further pursued here.

Numerical experiments discussed in Section 7 show that the approximate solution obtained by discretizing the weak formulation given in Equation (2.15) behaves nicely in a certain distance from a single discrete charge. A formal analysis of the stability and convergence behavior is unfortunately outside the scope of this thesis. For a more thorough theoretical viewpoint on the equivalence between different problem formulations, including the weak formulation, when encountered with a distributional right-hand side for the Poisson equation, the reader is referred to Babuška and Nistor [6]. Further, a convergent numerical scheme based on the weak formulation of the nonlinear Poisson-Boltzmann equation with the contribution from a single δ-distribution can be found in the work by Chen et al. [13].

For convenience, we restate the problem which will be under consideration in this section;

$$\begin{aligned} -\nabla^2 u &= \sum_{i=1}^{N} \delta(\boldsymbol{x} - \boldsymbol{x}_i) && \text{in } \Omega, \\ u &= g(\boldsymbol{x}) && \text{on } \partial\Omega_{\mathrm{D}}, \\ \frac{\partial u}{\partial \boldsymbol{n}} &= 0 && \text{on } \partial\Omega_{\mathrm{N}}. \end{aligned} \tag{2.49}$$

The distribution denoted by $\delta(\boldsymbol{x} - \boldsymbol{x}_i)$ is the Dirac measure giving unit mass to the point $\boldsymbol{x}_i \in \Omega$. We start our analysis with a short recap of distributional theory and fundamental solutions, before Green functions applicable to bounded domains is introduced. This theory will prove existence of the solution, and its uniqueness is then considered in a short concluding paragraph.

2.6.1 Distributions and Fundamental Solutions

Distributions are generalized functions defined through their action on smooth test functions with compact support. A formal definition can be found, e.g, in *Functional Analysis* by Rudin [55], or other textbooks on functional analysis or distribution theory. We will settle without a formal definition, but keep in mind the following properties.

Properties (Distribution)**.** *A distribution is infinitely differentiable (in the distributional sense), and has partial derivatives which are again distributions. Every continuous function is a distribution. A differentiable function has a distributional derivative that coincides with the usual derivative defined in calculus.*

We denote by $\mathcal{D}'(\mathbb{R}^3)$ the space of distributions on $\mathbb{R}^3$. The idea of using distributions to describe physical phenomena has been used by physicists for a long time, and fundamental solutions in the Green functions sense was introduced in 1828 [27]. However, the Dirac distribution was not introduced by Dirac until the 1920's, and the theory of distributions was formalized as late as in the middle of the 20th century, often accredited to L. Schwartz [9]. This theory can be taken advantage of when searching for analytic solutions to PDEs. For linear differential operators on infinite domains, the theory gives rise to the use of *fundamental solutions*.

Definition 1 (Fundamental Solution)**.** *A distribution Φ is a fundamental solution to the linear differential operator L with constant coefficients if it satisfies*

$$L\Phi = -\delta(\boldsymbol{x}),$$

where δ denotes the Dirac distribution.

The following standard result from functional analysis ensures that such a fundamental solution can always be found if the differential operator is sufficiently nice.

Theorem 1 (Malgrange-Ehrenpreis, Existence of Fundamental Solutions)**.** *There always exists a fundamental solution to the linear differential operator L with constant coefficients.*

Proof. See Rudin [55]. □

As a consequence, we can state the following corollary.

Corollary 1. *The distributional solution to*

$$-\nabla^2 u = \sum_i \delta(\boldsymbol{x} - \boldsymbol{x}_i)$$

exists and is a linear combination of fundamental solutions.

Proof. By translation of the Dirac measure and linearity of the Laplacian ∇^2, letting Φ_i be the solution to $-\nabla^2\Phi_i = \delta(\boldsymbol{x} - \boldsymbol{x}_i)$, the superposition principle gives $u = \sum \Phi_i$. □

Remark 1. *The solutions implied by Corollary 1 are, in general, not regular functions but distributions.*

2.6.2 Green Functions

The question now arising is if these fundamental solutions can be applied on the bounded domain Ω and additionally satisfy the set of mixed boundary conditions defined in Equation (2.49). The answer is yes, and to see this, let us consider the existence of a *Green function* in Ω [22, 59]. The Green function on Ω with mixed boundary conditions can be defined, for fixed $\boldsymbol{x}$, as the solution to

$$\begin{aligned} -\nabla_y^2 G(\boldsymbol{x}, \boldsymbol{y}) &= \delta(\boldsymbol{x} - \boldsymbol{y}) \quad && \text{in } \Omega, \\ G(\boldsymbol{x}, \boldsymbol{\sigma}) &= 0, && \sigma \in \partial\Omega_{\mathrm{D}}, \\ \frac{\partial}{\partial \boldsymbol{n}} G(\boldsymbol{x}, \boldsymbol{\sigma}) &= 0 && \sigma \in \partial\Omega_{\mathrm{N}}, \end{aligned} \tag{2.50}$$

see e.g., *Partial Differential Equations* by Evans [22] or *Partial Differential Equations in Action* by Salsa [59]. The Green function for the boundary value problem (2.50) can be constructed by subtracting a *corrector distribution* ϕ from the fundamental solution,

$$G(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x} - \boldsymbol{y}) - \phi(\boldsymbol{x}, \boldsymbol{y}), \tag{2.51}$$

where ϕ must solve the boundary value problem given by

$$\begin{aligned} -\nabla_y^2 \phi &= 0 && \text{in } \Omega, \\ \phi(\boldsymbol{x}, \boldsymbol{\sigma}) &= \Phi(\boldsymbol{x} - \boldsymbol{\sigma}) && \sigma \in \partial\Omega_{\mathrm{D}}, \\ \frac{\partial \phi}{\partial \boldsymbol{n}} &= \frac{\partial}{\partial \boldsymbol{n}} \Phi(\boldsymbol{x} - \boldsymbol{\sigma}) && \sigma \in \partial\Omega_{\mathrm{N}}, \end{aligned} \tag{2.52}$$

for fixed $\boldsymbol{x}$. A sufficient condition for the existence of a solution to Equation (2.52) is that the domain Ω is a Lipschitz domain [59]. From this and Theorem 1, the distribution G exists as defined in Equation (2.51) for our problem, which domain defined by the APD is a cuboid (see Section 7.2), and thus a Lipschitz domain [54]. A suitable sum of Green functions,

$$G(\boldsymbol{y}) := \sum_i G(\boldsymbol{x}_i, \boldsymbol{y}), \tag{2.53}$$

solves

$$\begin{aligned} -\nabla_y^2 G(\boldsymbol{y}) &= \sum_i \delta(\boldsymbol{x}_i - \boldsymbol{y}) \quad && \text{in } \Omega \\ G(\boldsymbol{\sigma}) &= 0, && \sigma \in \partial\Omega_{\mathrm{D}}, \\ \frac{\partial}{\partial \boldsymbol{n}} G(\boldsymbol{\sigma}) &= 0 && \sigma \in \partial\Omega_{\mathrm{N}}, \end{aligned} \tag{2.54}$$

by constructing a corrector distribution ϕ with the boundary conditions corresponding to the sum of fundamental solutions and their normal derivatives on the Dirichlet and Neumann boundary respectively.

Further, let $\tilde{u}$ solve the homogeneous Laplace equation with the Dirichlet and Neumann

conditions from Equation (2.49),

$$\begin{aligned} \nabla^2 \tilde{u} &= 0 && \text{in } \Omega, \\ \tilde{u} &= g(\boldsymbol{x}) && \text{on } \partial\Omega_{\mathrm{D}}, \\ \frac{\partial \tilde{u}}{\partial \boldsymbol{n}} &= 0 && \text{on } \partial\Omega_{\mathrm{N}}. \end{aligned} \tag{2.55}$$

The solution $\tilde{u}$ exists under the same conditions as for Equation (2.52). Defining the sum of the Green function from Equation (2.53) and $\tilde{u}$, the function

$$u(\boldsymbol{x}) := G(\boldsymbol{x}) + \tilde{u}(\boldsymbol{x}) \tag{2.56}$$

solves Equation (2.49) and exists under the Lipschitz assumption on Ω. We thus conclude positively the existence analysis of the solution to our problem. The next short section will give the final uniqueness result.

2.6.3 Uniqueness

A suitable max-min principle [22, 53, 59] cannot directly be applied to the solution u, since G, in general, is a distribution, and u cannot be assumed to lie in $C^2(\Omega)$. However, the following theorem [19] will provide us with the sufficient foundation for proving uniqueness.

Theorem 2 (Weyl's Lemma)**.** *Every distributional solution $u \in \mathcal{D}'(\Omega)$ to the homogeneous Laplace equation $-\nabla^2 u = 0$ is a classical solution, i.e., the solution is twice continuously differentiable.*

Assuming u_1 and u_2 both satisfy Equation (2.49), with $u_1 \neq u_2$, the proof of uniqueness is a simple proof by contradiction as $u_1 - u_2$ solves the homogeneous problem, and hence, by Theorem 2, a max-min principle can be applied to $u_1 - u_2$ and yields $u_1 - u_2 \equiv 0$. □

3 | Solving the Linear System

The finite element discretization of Poisson equation elaborated in the previous section resulted in a sparse linear system, represented in Equation (2.29). When applying finite element discretizations to MC particle simulations, the number of necessary nodes is often very large, leading to a large linear system of equations. The sparsity pattern for two stiffness matrices calculated for grids with different resolution and element order is shown in Figure 3.1. Many different methods, both direct [17] and iterative [56], can be applied to provide us with a solution to the linear system. Iterative methods based on Krylov subspace methods with preconditioners arising from direct methods are typically used to produce a solver which has favorable behavior with respect to speed, accuracy and stability [7, 8].

Taking advantage of the sparsity of the system is important, and to this aim, types for holding sparse matrix information are implemented in the MCFEM. The chosen formats are dictonary of keys (DOK) for assembly and compressed row storage (CRS) for calculations. A short

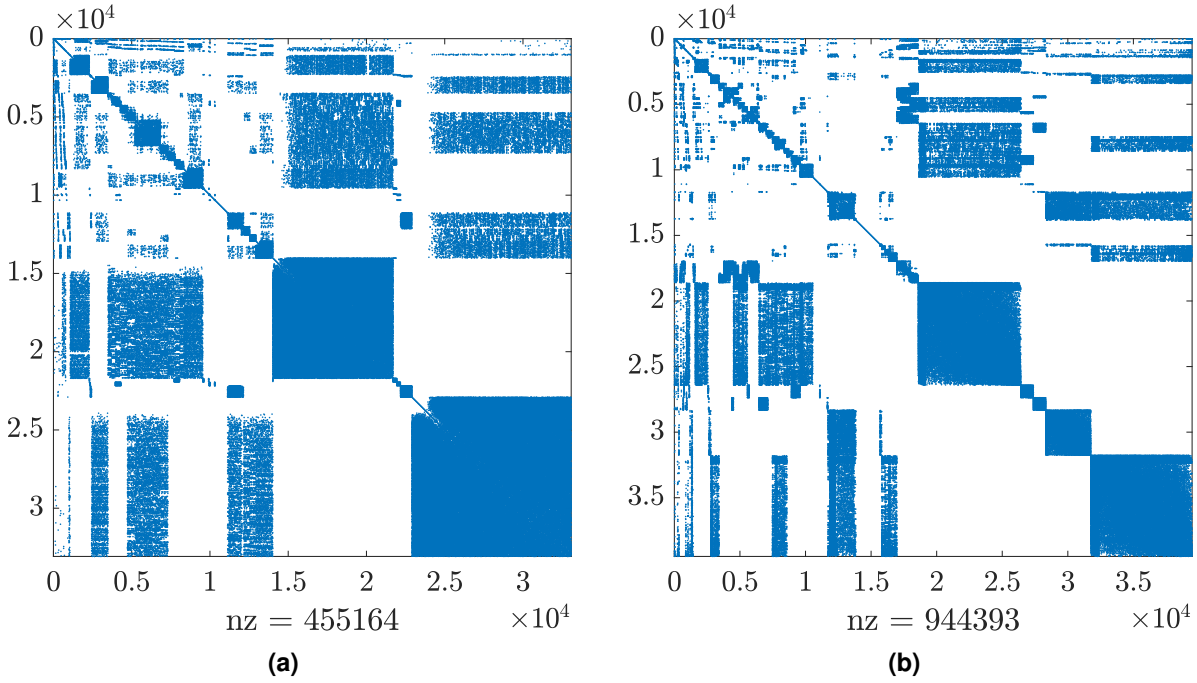


Figure 3.1: Sparsity patterns for two different stiffness matrices arising from the use of unstructured grids. The two matrices represent a grid with 33,080 nodes and linear elements (a), and a grid with 39,475 nodes and quadratic elements (b). The non-zero elements are illustrated with blue dots. The total number of non-zero elements is (a) $N_z = 455,164$ and (b) $N_z = 944,393$.

overview of these is presented in Section 3.1. A preconditioned conjugate gradient method (PCG) is applied to the system and briefly described in Section 3.2. Some numerical results for applying different methods in the first time-step of the MCFEM is presented in Section 3.3. In the code, the use of types and operator overwriting makes it easy to extend the number of storage formats or change solver in the future. The modules for sparse linear algebra operations are independent of the rest of the code and can be exported for use in other types of programs requiring sparse linear algebra structures. For a wider range of sparse linear algebra operations, it is also possible to integrate a general library such as *SPARSEKIT* [57].

3.1 Sparse Matrix Storage Schemes

The motivation for implementing special storage schemes for sparse matrices is the reduction of memory usage and computation time for matrix-vector multiplications. The matrix-vector multiplication is the most important operation in Krylov subspace applications. The idea of sparse storage schemes is to only store the non-zero elements of the matrix. The DOK-format is the format of choice to assemble the matrices, due to its simplicity and intuitive storage of matrix values coupled to their matrix position with a row-column pair key. When a new non-zero entry is added to the matrix, its row and column pair is appended to the list of keys, and its value is appended to the array of values. A non-zero value can easily be replaced by looking up the key and change the corresponding value.

The CRS-format is even less memory-demanding and provide fast access to rows for the computation of matrix-vector products. For the matrices with sparsity pattern visualized in Figure 3.1, the CRS-format needs only 0.09% and 0.12%, respectively for Figure 3.1(a) and Figure 3.1(b), of the memory used when saving the matrices in a full format. The format consists of three arrays and any sparse matrix can be saved without loss of information in this format. For an $m \times n$ matrix B with a total of N_z non-zero elements, an array $\boldsymbol{v}_{\text{CRS}}$ will contain the non-zero values of the matrix, where each row is stored sequentially. An array $\boldsymbol{c}_{\text{CRS}}$ contains the column indices for each corresponding value in $\boldsymbol{v}_{\text{CRS}}$. The last array, $\boldsymbol{r}_{\text{CRS}}$, is of length $m+1$, and its ith element contains the storage index of the first non-zero value in the ith row of the original matrix, with which one can index into $\boldsymbol{v}_{\text{CRS}}$. In other words, the index of the first non-zero element at row i is stored in $\boldsymbol{r}_{\text{CRS}}$ at index i. The last value in $\boldsymbol{r}_{\text{CRS}}$ is $(N_z + 1)$ and can be interpreted as the index of a first fictional value in the non-existing $(m + 1)$th row of the matrix B. The non-zero elements of the ith row of B can thus easily be accessed as by the indices ranging from the ith value of $\boldsymbol{r}_{\text{CRS}}$ to the $(i + 1)$th value of $\boldsymbol{r}_{\text{CRS}}$ subtracted by 1. The details are somewhat technical, and the following example hopefully clarifies the idea.

Example 1. *Given a* 4×5 *matrix*

$$B = \begin{bmatrix} b_{11} & b_{12} & 0 & 0 & b_{15} \\ 0 & b_{22} & b_{23} & 0 & 0 \\ b_{31} & 0 & 0 & b_{34} & b_{35} \\ 0 & 0 & b_{43} & 0 & 0 \end{bmatrix},$$

the three vectors holding the sparse information will look like

$$\begin{aligned} \boldsymbol{v}_{CRS} &= [b_{11}, b_{12}, b_{15}, b_{22}, b_{23}, b_{31}, b_{34}, b_{35}, b_{43}], \\ \boldsymbol{c}_{CRS} &= [1, 2, 5, 2, 3, 1, 4, 5, 3], \\ \boldsymbol{r}_{CRS} &= [1, 4, 6, 9, 10]. \end{aligned}$$

To access row number three, compute the range of indices for indexing into $\boldsymbol{v}_{CRS}$ *as*

$$I_3 = range(\boldsymbol{r}_{CRS,3}, (\boldsymbol{r}_{CRS,4} - 1)),$$

giving $I_3 = [6, 7, 8]$. *Then*

$$\boldsymbol{v}_{CRS,I_3} = [\boldsymbol{v}_{CRS,6}, \boldsymbol{v}_{CRS,7}, \boldsymbol{v}_{CRS,8}] = [b_{31}, b_{34}, b_{35}].$$

The values of $\boldsymbol{v}_{CRS,I_3}$ *are exactly the three non-zero values in row number three. If we are to compute the matrix-vector product of B with a vector* $\boldsymbol{a} = [a_1, a_2, a_3, a_4]$*, we must take advantege of the values in the vector with column-information,* $\boldsymbol{c}_{CRS}$*. For each row r we index into* $\boldsymbol{a}$ *with an index vector of column values,* $J_r = \boldsymbol{c}_{CRS,I_r}$*, such that*

$$B\boldsymbol{a} = \begin{bmatrix} \boldsymbol{v}_{CRS,I_1} \cdot \boldsymbol{a}_{J_1} \\ \boldsymbol{v}_{CRS,I_2} \cdot \boldsymbol{a}_{J_2} \\ \boldsymbol{v}_{CRS,I_3} \cdot \boldsymbol{a}_{J_3} \\ \boldsymbol{v}_{CRS,I_4} \cdot \boldsymbol{a}_{J_4} \end{bmatrix} = \begin{bmatrix} [b_{11}, b_{12}, b_{15}] \cdot [a_1, a_2, a_5] \\ [b_{22}, b_{23}] \cdot [a_2, a_3] \\ [b_{31}, b_{34}, b_{35}] \cdot [a_1, a_4, a_5] \\ b_{11}a_3 \end{bmatrix}.$$

The computation of matrix-vector products is fast with the matrix stored in CRS-format, because each row is stored sequantially in memory and is reachable with simple indexing. In addition, all floating point operations that would include multiplication with zero in the full format is avoided. The reason the CRS-format is not used for assembly is that adding matrix elements is cumbersome and requires position shifting of the three storage vectors. When the stiffness matrix is assembled into a DOK-format, it can easily be converted to a CRS-format when the assembly is finished and the number of non-zero elements and their positions is known. The temporary DOK-matrix can be deleted from the memory when the conversion is finished.

3.2 Preconditioned Conjugate Gradient Method

The method used to solve the linear system in Equation (2.29) is a PCG with an incomplete LU-factorization with zero fill-in (ILU0) as the chosen preconditioner. The presentation of the algorithm follows the discussion of Saad [56]. The complete set of available system solvers in the MCS is a simple conjugate gradient method (CG), the PCG and a bi-conjugate gradient stabilized method (BiCG-Stab). The PCG was introduced in the program by Åsen [5], and the BiCG-Stab by Harang [31]. The BiCG-Stab is a generalization of the CG-method for applications with non-symmetric matrices. The main work on these solvers during this thesis has been to increase the readability of the code by implementing types for holding the sparse matrices and overwrite operators in order to keep the intuitive, abstract mathematical notation within the code. The solvers are now easily exportable to other types of software if needed.

The need for preconditioners when applying CG to large systems is due to slow technical convergence, especially when applied to electronic device simulation [56]. This will also be apparent from the small experiment applied in the next section, Section 3.3. The CG-method can be applied to symmetric positive definite matrices, such as our modified stiffness matrix A_{BC}. To precondition an arbitrary matrix A with ILU0 is a preferable choice because the sum of the lower triangular matrix L and the upper triangular matrix U, denoted here by $L + U = A^{LU}$, has the same sparsity pattern as the original matrix A. This is the property obtained with the zero fill-in procedure. The two matrices can thus be stored in a copy of the original matrix, and only the values in $\boldsymbol{v}_{\mathrm{CRS}}$ in a CRS-format must be altered, while $\boldsymbol{c}_{\mathrm{CRS}}$ and $\boldsymbol{r}_{\mathrm{CRS}}$ can stay the same. An additional array of diagonal pointers is stored to easily be able to separate the lower and upper triangular parts. The matrix A^{LU} is computed by partial Gaussian elimination, as shown in Algorithm 3.1. This matrix should not be confused with the matrix-matrix-product of the two parts of the decomposition, denoted by LU.

Algorithm 3.1 Computation of ILU0 [56]

Input: matrix to be preconditioned, A
Output: matrix preconditioner, A^{LU}

$A^{LU} = A$
for i = 1, ..., n **do**
  for k = 1, ..., i-1, and (i, k) s.t. $A_{ik} \neq 0$ **do**
    ${A^{LU}}_{ik} = \frac{{A^{LU}}_{ik}}{{A^{LU}}_{kk}}$
    for j = k+1, ..., n, and (i, j) s.t. $A_{ij} \neq 0$ **do**
      ${A^{LU}}_{ij} = {A^{LU}}_{ij} - {A^{LU}}_{ik}{A^{LU}}_{kj}$
    end for
  end for
end for

The favorable property of a lower-upper triangular matrix product is that the equation

$$LU\boldsymbol{x} = \boldsymbol{b} \tag{3.1}$$

can easily be solved by the simple scheme

$$\text{solve } L\boldsymbol{y} = \boldsymbol{b}, \qquad \text{then solve } U\boldsymbol{x} = \boldsymbol{y},$$

where the first part is done by a simple forward substituion since L is lower triangular and the second part by a back substitution since U is upper triangular. Hence, the matrix (LU) is easily invertible. The preconditioner is applied to a system $A\boldsymbol{x} = \boldsymbol{b}$ as

$$(LU)^{-1}A\boldsymbol{x} = (LU)^{-1}\boldsymbol{b}, \tag{3.2}$$

which is a system equivalent to the original one, but easier to solve. For a full discussion on the use of preconditioners, the derivation of the PCG, and convergence properties, the reader is referred to *Iterative Methods for Sparse Linear Systems* by Saad [56]. Here we simply restate the utilized procedure in Algorithm 3.2.

Algorithm 3.2 Precondtioned Conjugate Gradient for solving $A\boldsymbol{x} = \boldsymbol{b}$ [56]

Input: matrix A,
incomplete matrix factorization A^{LU}, holding L and U
vector $\boldsymbol{b}$
initial guess $\boldsymbol{x}_0$
tolerance τ

Output: solution $\boldsymbol{x}$

$\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
Solve $LUz_0 = \boldsymbol{r}_0$ by forward and back substitution
$\boldsymbol{p} = z_0$
$i = 0$
while $\|\boldsymbol{r}_i\|_2 \geq \tau$ **do**
  $\alpha = \dfrac{\boldsymbol{r}_i \cdot \boldsymbol{z}_i}{\boldsymbol{p} \cdot A\boldsymbol{p}}$
  $\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \alpha A\boldsymbol{p}$
  $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \alpha A\boldsymbol{p}$
  Solve $LUz_{i+1} = \boldsymbol{r}_{i+1}$ by forward and back substitution
  $\beta = \dfrac{\boldsymbol{r}_{i+1} \cdot \boldsymbol{z}_{i+1}}{\boldsymbol{r}_i \cdot \boldsymbol{z}_i}$
  $\boldsymbol{p} = \boldsymbol{z}_{i+1} + \beta\boldsymbol{p}$
  $i = i + 1$
end while

3.3 Numerical Evaluation

To test the performance of the implemented PCG, the method was applied to the linear system arising from solving the Poisson equation on the triangulation of an APD, where the interior volume is neutral and Dirichlet conditions are applied to part of the boundary. This corresponds to the first time step of the bias simulations discussed in Chapter 7. Three different triangulations where used, with different degrees of freedom. The PCG was compared to the CG without preconditioning, and the BiCG-Stab. The result of this comparison is presented in Table 3.1. The table shows the necessary number of iterations and CPU-time for each solver. The tolerance used is an absolute error of $\tau = 10^{-10}$. From these numbers, it is clear that that the PCG performs best in all three cases, both with respect to time and number of iterations. The best gain in performance compared to the two other methods is achieved for the largest system. As expected, each iteration step requires more time for the PCG than for the other two methods, but the accumulated run-time is lower due to the much faster convergence rate of PCG. The BiCG-Stab is important if equations resulting in non-symmetric discretization matrices is integrated into the MC-model, or the solvers are exported for use in other settings. However, as seen by its slow convergence, it should be subject to convergence improvements before it becomes the method of choice. The use of CG without preconditioning is unnecessary and should be avoided.

Table 3.1: Performance of three different methods for solving three different Poisson systems of the type given in Equation (2.29). N_h specifies the number of nodes in the triangulation used to construct the stiffness matrices, such that each matrix is of size $N_h \times N_h$. For each matrix, the column with header I holds the number of iterations, and the column with header t denotes the CPU-time in seconds. The PCG clearly outperforms the two other methods.

N_h	12.689		39.475		75.165	
	I	t[s]	I	t[s]	I	t[s]
PCG	**89**	**0.14**	**138**	**0.79**	**161**	**1.98**
BiCGStab	627	0.71	1492	6.34	5058	54.52
CG	722	0.31	1404	2.14	1919	7.05

4 | Point Location Algorithms

The computation of the load vector for the finite element discretization of the Poisson system requires the knowledge of the element location of electrons and holes, as mentioned in Section 2.5. In any structured grid, explicit formulas can be used to compute the current element location of a particle. In computer simulations with an unstructured decomposition of the domain, the search for this location is not trivial. The point location or particle tracking problem is defined within computational geometry [18]. Its solution is especially interesting in computer simulation of fluid mechanics, biomedicine and in our case for particle simulation in solid state physics, when applying unstructured grids. The implemented point location algorithms (PLAs) are applicable to both 2D and 3D triangulations, with only minor changes between the two cases. The 2D-routine is used to handle injection of particles at the Ohmic contacts (OCs), and the 3D-routine is used for assembling the load vector and interpolate the electric field in each particle position. In this chapter, we will present the implemented PLA in the two different cases. The injection routine is developed in Chapter 5.

Several PLAs have been suggested during the last decades [14, 15, 33, 39, 41, 48, 71]. To the best of our knowledge, the most recent contribution is presented by Capodaglio and Aulisa [12], handling unstructured, hybrid meshes with non-planar element faces in parallel finite element applications. Common to these methods is that they are all introduced in the setting of multiphase flow, where particle paths are determined by the velocity field calculated independently of the particle positions. For semiconductor simulation, this framework is altered and we define the problem and discuss some aspects of this particular case in Section 4.1. A solution to the point location problem, based mainly on the work by Chordá et al. [15], is presented in Section 4.2.

4.1 The Point Location Problem

We assume a particle p is in a position given by its Cartesian coordinates, $\boldsymbol{x}_p \in \Omega$, and that $\mathcal{T}_h$ is a triangulation of Ω. The point location problem consists of finding the element in the triangulation which contains the particle, i.e.

$$\text{find} \quad K_p \in \mathcal{T}_h \ \text{ s.t. } \ K_p \ni \boldsymbol{x}_p.$$

The general approach is to search through the elements of $\mathcal{T}_h$ from an initial element guess K_{init}, and test for inclusion of $\boldsymbol{x}_p$. We will consider triangulations which consists exclusively of tetrahedral volume elements or triangles in a plane. The use of the affine map given in Equation (2.37) ensures planar faces (triangles) for tetrahedra or straight faces (edges) for triangles. This can be taken advantage of to simplify the PLAs. The procedures presented by Chordá et al. [15] applies to these cases. If an extension to iso-parametric mappings for the quadratic elements is made, the use of face decompositions as described by Capodaglio and Aulisa [12] and Kuang et al. [45] would be necessary. This approach is not considered during this work.

As opposed to fluid applications where particles will follow the calculated velocity fields, the carriers in a MC simulation are subject to random particle scattering, which at any time can change the particle's flight direction. At the time of solving Poisson equation, the particles are considered to be frozen in their current position. Each particle's coordinates are passed to the PLA, with the initial element guess equal to its location in the previous step. Between Poisson steps, particles will on average move at most a few elements from their old position, so efficiency on short search distances are the most important. However, for the initialization of element locations for each particle, the algorithm must be able to handle long searches through the entire triangulation. Correct particle positions are important for the approximation of the self-consistent forces in the MC-simulation.

4.2 Implemented Solution to the Point Location Problem

The considered algorithms for point location is based on a search from an initial element guess, denoted by K_{init}, where the search path is constructed by traversing elements in the triangulation that lies on the path from this initial element to the point searched for. The path is determined by geometrical calculations. In both 2D and 3D triangulations, a search direction $\boldsymbol{S}$ is defined from the center of the initial element guess K_{init} to the particle position $\boldsymbol{x}_p$,

$$\boldsymbol{S} = \boldsymbol{x}_p - \boldsymbol{x}_{\text{init}}, \tag{4.1}$$

where $\boldsymbol{x}_{\text{init}}$ denotes the center of K_{init}. Elements on the search path are traversed by always moving to the neighboring element sharing the face that intersects $\boldsymbol{S}$ closest to the particle. When the algorithm has entered the element K_p containing the particle, it terminates. If the domain Ω is convex and the point does not lie outside the domain, the geometrical computations will ensure the convergence of the method [15]. A sketch showing the algorithm's path through a 2D grid is shown in Figure 4.1. We consider planes and volumes separately and start with the 2D-case.

4.2.1 Point Location in Two Dimensional Triangulations

In 2D, the coordinates of a point is $\boldsymbol{x} = (x, y)$ and the element faces are edges. If an edge is intersected by $\boldsymbol{S}$ can be decided by applying a test called the trajectory-to-the-left test (TTL),

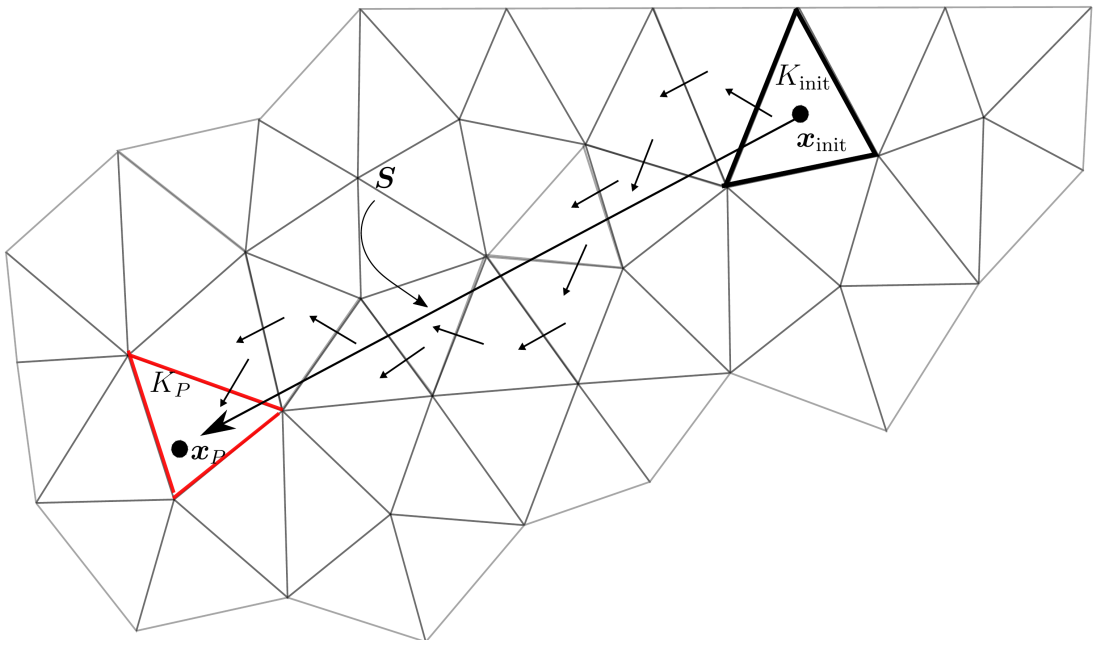


Figure 4.1: A sketch showing how the PLA traverses elements in a triangulation, to locate the point $\boldsymbol{x}_p$. It is initiated from the element K_{init} and the initial search point $\boldsymbol{x}_{\text{init}}$ is calculated as K_{init}'s center. The search direction $\boldsymbol{S}$ is computed from Equation (4.1). The algorithm traverses elements on the path to $\boldsymbol{x}_p$, always walking to the neighboring element sharing the edge intersecting the search trajectory $\boldsymbol{S}$. The algorithm terminates when reaching $K_P \ni \boldsymbol{x}_p$, marked in red.

explained in the following. Assume, without loss of generality, that the algorithm has entered an element K_{current} on the path from K_{init} to K_p, as visualized in Figure 4.2. This element has three directed edges, e_0, e_1 and e_2, all of which are oriented counterclockwise around the triangle. Each edge is defined by a local start node $\boldsymbol{x}_0$ and a local end node $\boldsymbol{x}_1$. Defining two new vectors

$$\begin{aligned} \boldsymbol{a}_0 &= \boldsymbol{x}_0 - \boldsymbol{x}_{\text{init}}, \\ \boldsymbol{a}_1 &= \boldsymbol{x}_1 - \boldsymbol{x}_{\text{init}}, \end{aligned} \tag{4.2}$$

and computing the third component of the cross product $\boldsymbol{a}_i \times \boldsymbol{S}$,

$$(\boldsymbol{a}_i \times \boldsymbol{S})_3 := \alpha_i = (x_i - x_{\text{init}})(y_p - y_{\text{init}}) - (x_p - x_{\text{init}})(y_i - y_{\text{init}}), \tag{4.3}$$

can help us determine edge intersection. The following can be interpreted from the sign of α_i:

$$\begin{aligned} \alpha_i < 0 &\iff \boldsymbol{a}_i \text{ lies to the left of } \boldsymbol{S} &&\implies \text{TTL returns true} \\ \alpha_i = 0 &\iff \boldsymbol{S} \text{ goes through } \boldsymbol{x}_i &&\implies \text{TTL returns true} \\ \alpha_i > 0 &\iff \boldsymbol{a}_i \text{ lies to the right of } \boldsymbol{S} &&\implies \text{TTL returns false} \end{aligned}$$

If $\text{sign}(\alpha_0) \neq \text{sign}(\alpha_1)$, the search trajectory $\boldsymbol{S}$ must intersect the given edge, as shown in Figure 4.2(b). This is equivalent to the TTL being true for one vector $\boldsymbol{a}_i$ and false for the other. The case when $\boldsymbol{S}$ passes through one of the nodes is treated as an intersection. If an

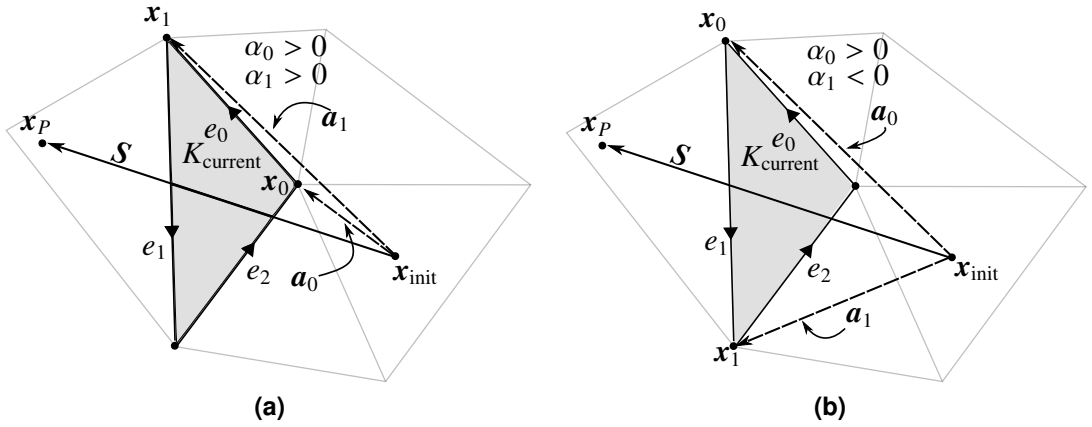


Figure 4.2: Edge e_0 (a) and edge e_1 (b) is tested for intersection with the search trajectory $\boldsymbol{S}$. The vectors $\boldsymbol{a}_0$ and $\boldsymbol{a}_1$ is computed locally for each edge. For edge e_0, both lies to the right, and $\text{sign}(\alpha_0 \cdot \alpha_1) > 0$. For edge e_1, $\boldsymbol{a}_0$ lies to the right, and $\boldsymbol{a}_1$ to the left, so $\text{sign}(\alpha_0 \cdot \alpha_1) < 0$. The test concludes that e_1 is intersected by $\boldsymbol{S}$, and continues to the PTL. The PTL will confirm that e_1 is the exit face of K_{current}.

edge is intersected, it must either be the *entry face* or the *exit face* of the current element. The intersected edge where the search trajectory points into the element is the entry face, the edge where it points out is the exit face. The algorithm must find which of the intersected edges is the exit face, and go to the element adjacent to this face. This introduces the need for an additional test, the particle-to-the-left test (PTL). The PTL is used to determine if $\boldsymbol{x}_p$ lies to the left of the intersected edge. If so, the edge must be the entry edge, and if not, it must be the exit face. The PTL is computed as the third component of the cross product between the edge vector e and a vector from $\boldsymbol{x}_0$ to $\boldsymbol{x}_p$,

$$\beta_e = (x_1 - x_0)(y_P - y_0) - (x_P - x_0)(y_1 - y_0). \tag{4.4}$$

The interpretion of β_e is

$$\begin{aligned}
\beta_e < 0 &\iff \boldsymbol{x}_p \text{ lies to the left of } e &&\implies \text{PTL returns true} \\
\beta_e = 0 &\iff \boldsymbol{x}_p \text{ lies on } e &&\implies \text{PTL returns true} \\
\beta_e > 0 &\iff \boldsymbol{x}_p \text{ lies to the right of } e &&\implies \text{PTL returns false}
\end{aligned}$$

When the point lies to the right of the current edge, the algorithm continues to the neighbor element adjacent to that edge. This will be the edge closest to $\boldsymbol{x}_p$. If $\boldsymbol{x}_p$ is found to lie to the left, or on e, the next edge in the triangle is chosen and tested for intersection. As can be observed in Figure 4.2, any element intersected by the search trajectory will have one and only one edge which both intersects $\boldsymbol{S}$ and has the particle postion to its right, unless the triangle contains $\boldsymbol{x}_p$. Thus, if β_e is positive (the PTL is true) for both edges having $\alpha_0 \cdot \alpha_1 \leq 0$, the particle must be contained in the current element, and the algorithm has converged. The procedure can be implemented as in Algorithm 4.1. The next step is the extension of the algorithm to the 3D-case.

Algorithm 4.1 The 2D Point Location Algorithm

Input: Coordinates $\boldsymbol{x}_p$ of particle,
triangulation $\mathcal{T}_h$,
initial search element K_{init}
Output: Element $K_p \ni \boldsymbol{x}_p$

$\boldsymbol{x}_{\text{init}}$ = center of K_{init}
$\boldsymbol{S} = \boldsymbol{x}_p - \boldsymbol{x}_{\text{init}}$
$K_p = -1$
$K_{\text{current}} = K_{\text{init}}$
while $K_p == -1$ **do**
  exit face = -1
  for all Faces e of K_{current} **do**
    Compute α_0 and α_1 from Equation (4.3)
    if $\alpha_0 \cdot \alpha_1 \leq 0$ **then**
      Compute β_e from Equation (4.4)
      if $\beta_e < 0$ **then**
        exit face = e
      end if
    end if
  end for
  if exit face == -1 **then**
    $K_p = K_{\text{current}}$
  else
    K_{current} = neighbor(K_{current}, exit face)
  end if
end while

4.2.2 Point Location in Three Dimensional Triangulations

Element faces are now triangles and denoted by F_i. As in the 2D-case, a test for intersection of the trajectory with faces of the current element is necessary to determine the path of the algorithm. Further, a test to determine which side of a face the particles lies is used to separate the entry face from the exit face. These two tests are called the face-trajectory-intersection test (FTI) and the particle-to-the-inside test (PTI).

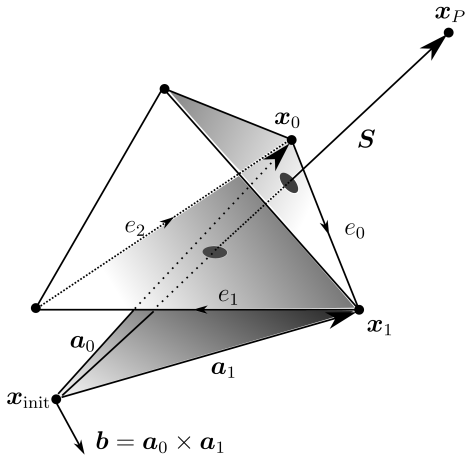
Each face consists of three directed edges, oriented counterclockwise with respect to the face normal pointing out of the element. As in the 2D-case, each edge has a start node $\boldsymbol{x}_0$ and an end node $\boldsymbol{x}_1$, and we associate the following vectors to each edge of the face:

$$\begin{aligned} \boldsymbol{a}_0 &= \boldsymbol{x}_0 - \boldsymbol{x}_{\text{init}}, \\ \boldsymbol{a}_1 &= \boldsymbol{x}_1 - \boldsymbol{x}_{\text{init}}, \\ \boldsymbol{b} &= \boldsymbol{a}_0 \times \boldsymbol{a}_1. \end{aligned} \tag{4.5}$$

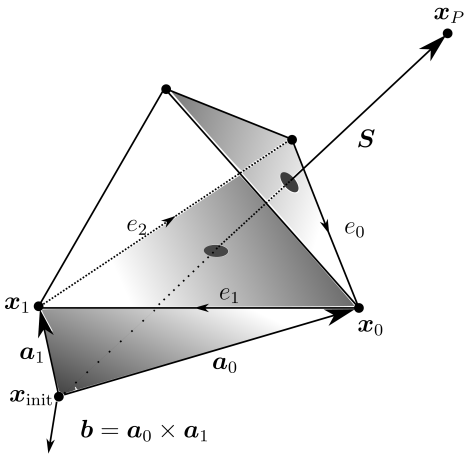
See Figure 4.3 for a visualization of these vectors for each edge of a face of a tetrahedron. If the scalar product between $\boldsymbol{b}$ and $\boldsymbol{S}$,

$$\omega_e = \boldsymbol{b} \cdot \boldsymbol{S}, \tag{4.6}$$

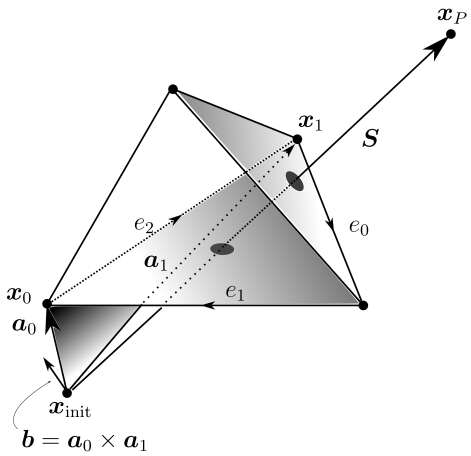
has the same sign for all three edges, the search trajectory intersects the face, and FTI returns true [15]. For a face intersected by $\boldsymbol{S}$, the PTI is then used to determine if the particle lies to the inside or outside of the element, with respect to the given face. An entry face will have the particle to the inside, meaning the particle lies on the same side of the face as the element, but not necessarily within the element. For an exit face, the particle will lie on the outside, i.e., on the opposite side of the face with respect to the element. To determine the value of the PTI, a



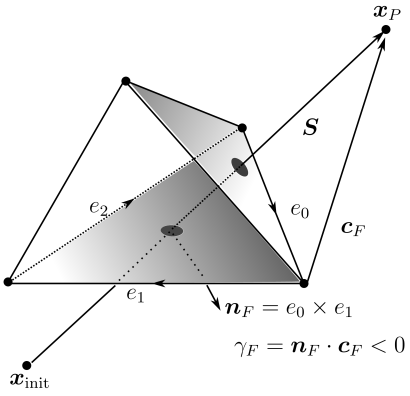
(a) Associated vectors to edge e_0



(b) Associated vectors to edge e_1



(c) Associated vectors to edge e_2



(d) Vectors for PTI

Figure 4.3: The steps of the 3D PLA for one face of a tetrahedron, here the bottom face. For each edge of the face (a-c), the vectors defined in Equation (4.5) is calculated. In each case shown here, $\omega_e < 0$, and the search trajectory must intersect the face. The PTI (d) returns true, so the lower face can not be the exit face for the search trajectory. The algorithm continues its search for an exit face by considering the same calculations applied to the next face of the tetrahedron. For the other face intersected by the search trajectory, PTI will return false, and the exit face is found.

new vector from any node $\boldsymbol{x}_F$ of the face F to the point searched for is computed as

$$\boldsymbol{c}_F = \boldsymbol{x}_p - \boldsymbol{x}_F. \tag{4.7}$$

The outward pointing normal vector of the face is computed as

$$\boldsymbol{n}_F = e_0 \times e_1, \tag{4.8}$$

and we can define

$$\gamma_F = \boldsymbol{n}_F \cdot \boldsymbol{c}_F, \tag{4.9}$$

with the following interpretation:

$$\begin{aligned}
\gamma_F < 0 &\iff \boldsymbol{x}_P \text{ lies to the inside of } F &&\implies \text{PTI returns true,} \\
\gamma_F = 0 &\iff \boldsymbol{x}_P \text{ lies in the plane defined by } F &&\implies \text{PTI returns true,} \\
\gamma_F > 0 &\iff \boldsymbol{x}_P \text{ lies to the outside of } F. &&\implies \text{PTI returns false.}
\end{aligned}$$

Figure 4.3(d) shows an example of the first case. The algorithm will then continue its search for intersections with $\boldsymbol{S}$ on the other faces F of the element. If $\gamma_F > 0$, the next search element is set to the element adjacent to this face. If $\gamma_F < 0$ for both intersected faces, the point must lie inside the element and the algorithm has converged. The procedure is listed in Algorithm 4.2.

Remark 2. *The node ordering is important for correct results of the PLAs. If GMSH [24] is used for the grid construction, the correct node ordering is achieved by ensuring that all elements are associated with a physical group. See also Appendix A.*

4.2.3 Comparison with Other Methods

In the preparing work for this thesis [23], different approaches to the solution of the point location problem were discussed, and a new algorithm was suggested. In the 2D-Poisson solver implemented by Åsen [5], an edge traversing algorithm presented by Guibas and Stolfi [29], and refined by Brown and Faigle [11] was used. This 2D-algorithm is difficult to extend to the 3D-case. To compare the implemented method, called the Chordá's, Blasco's and Fueyo's algorithm (CBF), with other methods, we do a comparison on a 2D-triangulation. The CBF is tested against the face-to-point algorithm (FP) discussed in [23] and the run-times of the Guiba's and Stolfi's algorithm (GS) presented in [5]. The three different algorithms were tested in the same grid, and their CPU-time is compared in Figure 4.4. Since the old GS is not applicable to the 3D-case, the CBF is the best choice for an extension to 3D among the three compared algorithms. The FP previously suggested is abandoned due to lack of robustness and slower convergence than for the CBF. The CBF shows almost equal performance as the GS over short distances, which is the most important case in the MCFEM. One of the reasons the FP needs additional run-time is that it does not require ordered node numbers, leading to additional time

Algorithm 4.2 The 3D Point Location Algorithm

Input: Coordinates $\boldsymbol{x}_P$ of particle,
triangulation $\mathcal{T}_h$,
initial search element K_{init}
Output: Element $K_p \ni \boldsymbol{x}_p$

$\boldsymbol{x}_{\text{init}}$ = center of K_{init}
$\boldsymbol{S} = \boldsymbol{x}_p - \boldsymbol{x}_{\text{init}}$
$K_p = -1$
$K_{\text{current}} = K_{\text{init}}$
while $K_p == -1$ **do**
  exit face = -1
  for all Faces F of K_{current} **do**
    for all Edges e of face F **do**
      Compute ω_e from Equation (4.6)
    end for
    if ω_e has same sign for all e **then**
      Compute $\gamma_F = \boldsymbol{n}_F \cdot \boldsymbol{c}_F$
      if $\gamma_F > 0$ **then**
        exit face = F
      end if
    end if
  end for
  if exit face = -1 **then**
    $K_p = K_{\text{current}}$
  else
    K_{current} = neighbor(K_{current}, exit face)
  end if
end while

for computing orientation of faces. This is unnecessary work when applying physical groups in GMSH, which provides correct node ordering.

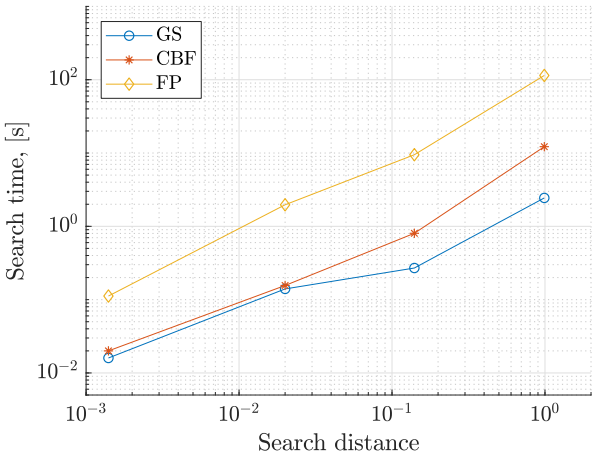


Figure 4.4: Run times of three different algorithms for point location, plotted against the search distance from an initial point to the point searched for. The times for the Guiba's and Stolfi's algorithm (GS) is taken from the thesis by Åsen [5], and slight variation might occur due to an unknown operative system for these calculations. The two other algorithms were tested in the same grid as used in Åsen's thesis, on a machine with Intel Core i7-4770 processor, 4 CPUs, clock frequency 3.40 GHz, running Ubuntu 16.04 LTS. The runtimes represent the time to locate 100,000 particles. The Chordá's, Blasco's and Fueyo's algorithm (CBF) is the algorithm used in MCFEM.

5 Boundary Conditions for Carrier Dynamics

When the free carriers move towards the boundary of the domain, they will be subject to particle boundary conditions. The conditions applied to the particles is of great importance and will influence how well the simulations mimic physical behavior [26]. The most common particle boundary conditions are the ones described by Hockney and Eastwood [35], with some variations on the use of velocity distribution for injected particles at contact surfaces [68]. For the free surfaces, Hockney and Eastwood proposes an elastic reflective boundary condition, implemented by a change of sign for the velocity component normal to the boundary. This causes a new flight direction for particles colliding with non-contact boundaries. For the Ohmic contacts (OCs), particles must be absorbed and injected according to physical models. Absorption is implemented by letting any particle reaching the contact surface be absorbed by the contact, and thus deleted from the simulation. Both reflection and absorption is simple and does not require any further considerations. However, there exist different models for particle injection at OCs, and this is the most challenging part of the boundary conditions for carrier dynamics. In the following, a method adapted to device simulation in unstructured grids is proposed.

5.1 Neutral Region

As described by González and Pardo [26], an OC remains in thermal equilibrium, also in situations where a potential is applied or other physical phenomena cause currents to flow through the contact. To achieve this behavior in a simulation, a region adjacent to the contact is kept in neutral equilibrium by inserting possibly missing charges at each time step. For simulation in structured grids, the imposed neutral region is in most cases the cells adjacent to the contact. For simulations in unstructured grids, the neutral region must be generalized to this type of application.

Aldegunde et al. [2] defines the neutral region as the set of elements with at least one node on the contact surface, and the number of particles to be inserted is calculated as the deficit of charge in this region. Then, particles are inserted at the surface of the contact according to a random distribution weighted by the charge density of the surface triangles in an equilibrium state.

To expand the horizon for unstructured grid contact modeling a new model is tested in the MCFEM. It differs from the method suggested by Aldegunde et al. [2] as

- The depth of the region is defined independently of mesh size

- No need to locate the volume elements which has nodes on the contact surface
- Injection of particles is done with respect to the current distribution of particles

In this new method, the neutral region is a region with depth Δy adjacent to the contact. Global neutrality in this region is imposed by inserting particles according to charge deficit at each time step of the simulation. The total number of particles that needs to be injected, denoted by N_{insert}, is calculated as

$$N_{\text{insert}} = \eta A_{\text{contact}} \Delta y - N_t. \tag{5.1}$$

Here, A_{contact} is the surface area of the contact, such that $A_{\text{contact}}\Delta y$ is the volume of the neutral region, η is the target carrier density of the region adjacent to the contact, and N_t is the actual number of carriers in the neutral region at any given time t. When new particles are created, they must be given a position in the simulation domain. In the method suggested, this position is based on local charge deficit, as described in the following.

5.2 Injection of Particles

The first OC-scheme introduced in the MC Software was implemented by Kirkemo [43], based on the description given by Fischetti and Laux in the collection *Monte Carlo Device Simulation : Full Band and Beyond* [34]. This framework is still maintained for the new OC-scheme, where inserted particles are created with the same carrier dynamics as in the old OC-scheme. The part that is altered is the injection point of the created particle.

If N_{insert} calculated from Equation (5.1) is positive, new particles are created. In order to decide where these particles should be inserted, the contact mesh on the surface is extrapolated in the y-direction to form a set of prisms that together constitute the neutral region. This is visualized in Figure 5.1. The positions of the new particles are decided by applying a neutrality condition to each of these prisms.

To compute the number of particles in each prism, the positions of the particles currently

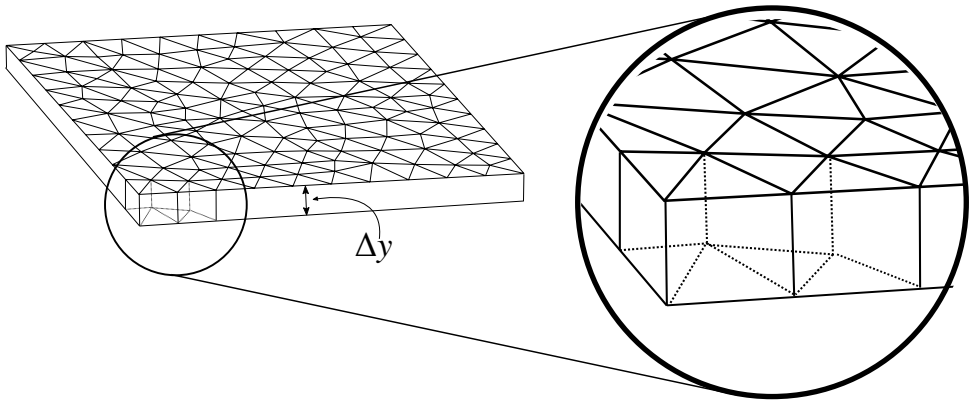


Figure 5.1: A surface triangulation of a contact region, with an imposed neutral region of thickness Δy. The triangulation is projected to form regular prisms in the y-direction, so the neutrality in each small prism can be accounted for.

contained in the neutral region is projected to the contact surface, and the 2D-PLA from Section 4.2 is used to find the triangle containing this projected particle. The projection is visualized in Figure 5.2. With this procedure, the number of particles in each prism can be counted. Each prism is then assigned a surplus or a deficit of carriers, according to the current number of particles within it. Prisms with a surplus of charge obtain no treatment. Actively ejecting particles from surplus cells can lead to instabilities, and surplus of charge must diffuse by the internal forces acting on the particles in the device, and lead the particles to find their way out through the contact without adding external, artificial forces to this procedure. If there was only local monitoring of charge deficit or surplus in each cell, one might risk to overinject charge, since prisms with charge surplus would not contribute to the calculation of charge neutrality. That is why the global estimate of charge in Equation (5.1) is implemented. If there is a need for global injection, the local prisms with charge deficit are chosen as candidates for inserting new particles. The inserted particle is positioned on the surface triangle of a randomly selected prism candidate with charge deficit, such that it enters the device during the next time step. The injection procedure is shown in Algorithm 5.1.

It is known that the boundary conditions applied to the OCs will influence the obtained characteristics of the device. Since the current through the device is directly calculated by counting particles that leaves or enters through the contact, this characteristic is especially vulnerable to non-physical injection models. Due to time limitations, the impact of using Algorithm 5.1 has not been tested against other methods for injection. Comparison with physical experiments would also be important in order to test its correctness.

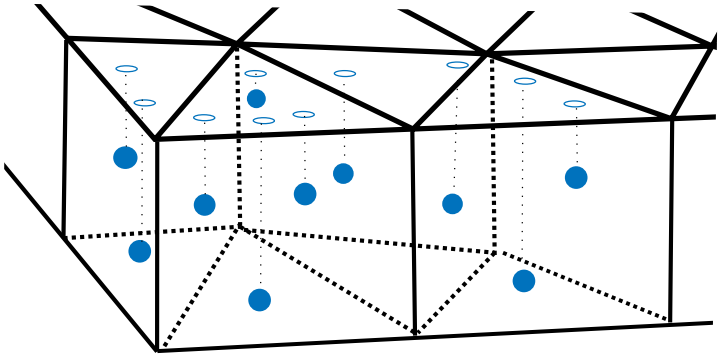


Figure 5.2: The position of particles are projected to the contact surface, and a 2D-PLA is used to decide a particles affiliation to the triangulation. The filled, blue dots resembles the particle positions in the volume, and the open circles show their projection to the contact triangulation. Any type of particle is projected in the same manner.

Algorithm 5.1 Injection of Particles at Contacts

Input: Contact specifications with 2D triangulation $\mathcal{T}_h$,
depth of neutrality region Δy
target carrier density η,
Particle positions $\{\boldsymbol{x}_p\}$

Output: Appended new particle positions to $\{\boldsymbol{x}_p\}$

Count number of particles,N_t, in the contact neutrality region
Calculate N_{insert} from Equation (5.1)
for all $K \in \mathcal{T}_h$ **do**
    Calculate charge in extrapolated prism: surplus or deficit
end for
while $N_{\text{insert}} > 0$ **do**
    Choose random triangle $K \in \mathcal{T}_h$
    if Charge deficit in extrapolated prism from K **then**
        Insert particle with postion in triangle center
        Give particle velocity direction into device
        $N_{\text{insert}} = N_{\text{insert}} - 1$
    end if
end while

6 Program Flow

The material presented in the previous chapters are all important building blocks for adapting the existing FFI-MCS to simulations on unstructured domain decompositions. The aim of this chapter is to give an overview of how these blocks are integrated into the MCS, resulting in the new program Monte Carlo software with finite element Poisson solver (MCFEM). The overall program flow of MCFEM is presented in Section 6.1. The part of the initialization concerning device geometry and triangulation is treated in Section 6.2 and the subflow performing the assembly is presented in Section 6.3. Finally, the program flow for the electric field calculations in each loop iteration is treated in Section 6.4.

In addition to the description given here, interactive documentation of the code has been generated using a program called FORD (FORtran Documenter)[1], which automatically generates HTML-documentation based on the source code and comments therein. This interactive documentation is obtained together with the source code. Due to the extensive number of modules and subroutines of the full program, the full program flow is not included here but can be found in the HTML-documentation. Appendix B provides an additional list of modules in the program, mainly intended for future developers of MCFEM.

6.1 Monte Carlo Program Structure

The focus of this thesis has been to integrate the new 3D FEM Poisson solver into the old MCS. The part of the code concerning the solid-state physics calculations have only seen minor readability updates, and a reader interested in this part of the code should consult the theses written by Kirkemo [43], Norum [50], and Olsen [51]. Figure 6.1 shows the outline of a MC-simulation performed with the MCFEM. The program starts by initializing the material parameters and tabulating scattering rates with respect to different energy levels for particles in their respective energy bands. The initialization also contains the geometry specifications and some calculations for the finite element routines. The particles are initialized according to the specified impurity charge densities in the device. After the initialization, the simulations enters the time loop and iterates as specified by the user of the program. In a Poisson solving time step, the electric field is updated before free flights are performed. Particles leaving the domain through the contacts during their free flights is deleted, and the injection of particles is performed according to the method described in Section 5. The remaining and newly created

[1] Available at GitHub: `https://github.com/cmacmackin/ford`

particles are scattered by the scattering calculations, and new flight directions are given to each scattered particle. Statistics on parameters of interest are gathered at user-defined loop intervals.

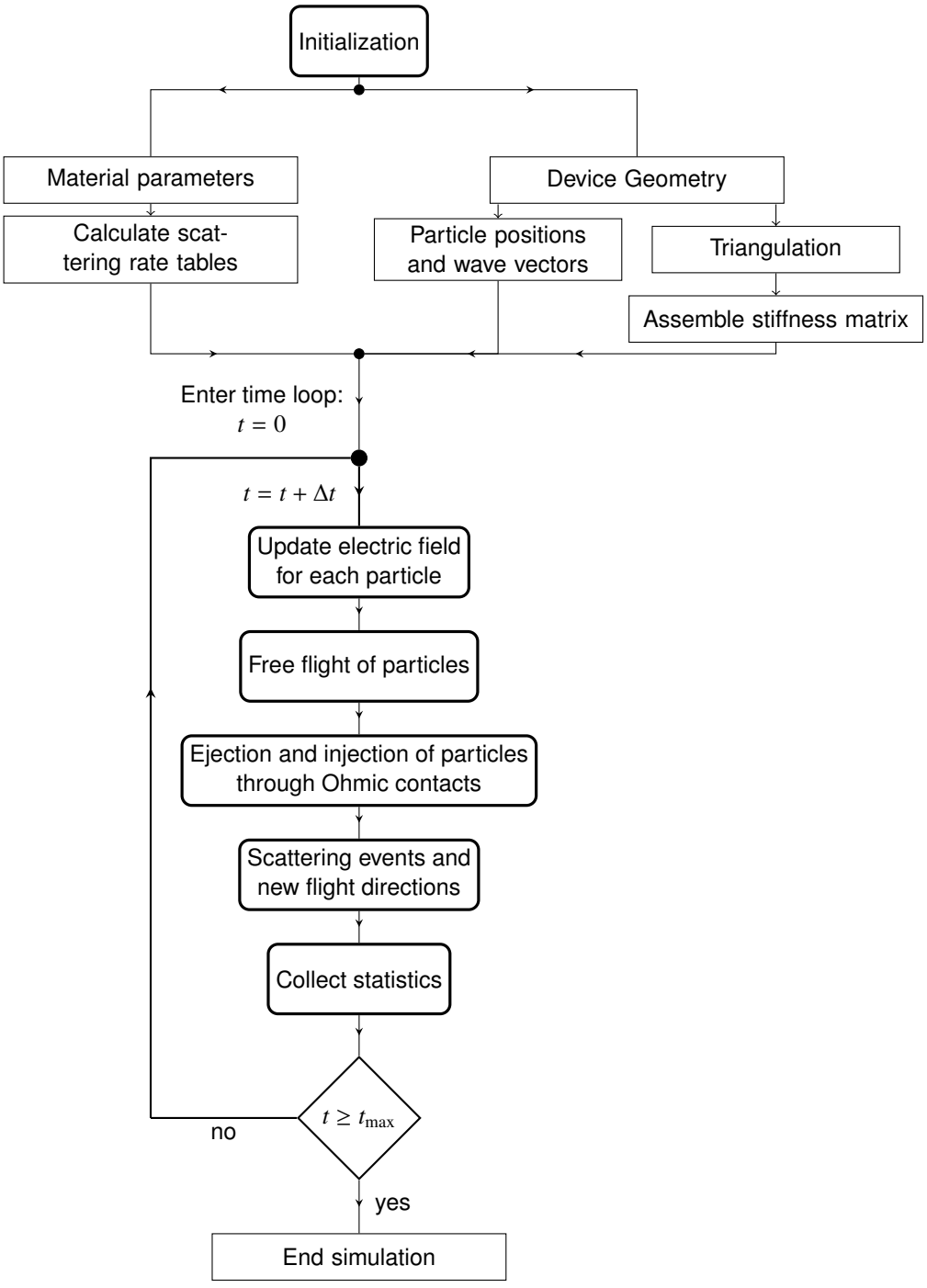


Figure 6.1: A simplified flowchart of the Monte Carlo simulation program, showing the most important segments of the program flow. The initialization routines for material parameters and device geometry are performed before entering the time loop. In the time loop, the particles are subjected to free flights and scattering events, with self-consistent updates of the electric field by the solution of the Poisson equation.

6.2 Device Geometry

The device geometry is constructed using GMSH [24]. A geometry file is the basis for creating the domain discretization with the built-in algorithms of GMSH. The software is a fast and powerful meshing tool, and the constructed triangulations can easily be used by the MCFEM with some simple read-in-tools. The constructed triangulation is taken as an input file to the device geometry initialization in the program, and a read-in-routine constructed to read the specific `*.msh` output-format from GMSH reads the information and saves it into a triangulation object. In addition to nodes and elements, the triangulation object holds information on boundary nodes for imposing Dirichlet conditions and neighboring elements for use in the point location algorithms (PLAs). The boundary information is saved during read-in, and the neighbors are found post-read-in based on the connectivity matrix. The program flow of the triangulation initialization is shown in Figure 6.2. Additional information on the use of GMSH and the construction of the triangulation can be found in Appendix A.

6.3 Pre-Loop Assembly Procedures

The linear system must be assembled according to the discussion in Section 2.5. Some of the assembly procedures can be done before the loop because they are state-independent. The stiffness matrix A, the matrix A_{BC} modified for boundary nodes, the lifting vector $\boldsymbol{g}$, the lifting term $A\boldsymbol{g}$, and the ILU0 preconditioner A^{LU} can be computed before entering the time loop. These variables are some of the parts needed to construct the linear system which is solved at every Poisson step. The triangulation is needed as input to some of the routines in this subflow. The flowchart for the assemblies is shown in Figure 6.3.

6.4 In-Loop Electric Field Calculations

With the pre-loop preparations explained in the previous sections, the only state-dependent part of the linear system is the load vector contribution from the free carriers. When the load vector is calculated, the system can be passed to the PCG-routine together with the ILU0 preconditioner. When Equation (2.29) is solved for the potential, and the Dirichlet contributions on the boundary are added, the potential is passed to the routine calculating the electric field in each carrier position, according to Equation (2.48). The subflow for calculating the electric field at each Poisson solving time step is shown in Figure 6.4.

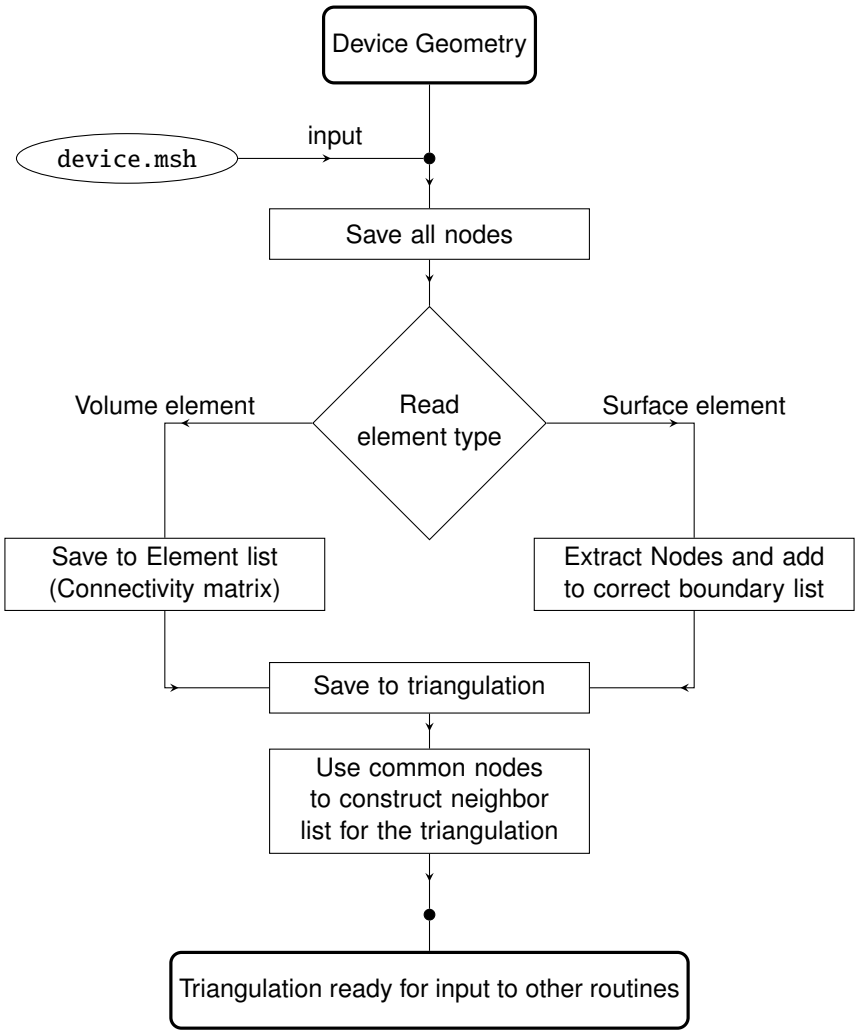


Figure 6.2: The flowchart representing the subflow of the MCFEM which task is to construct the triangulation. A mesh file, `device.msh`, constructed with GMSH [24], is read by the construction routine, and the necessary information about nodes, elements and boundaries are saved. The list of neighbors is constructed when the read-in is finished.

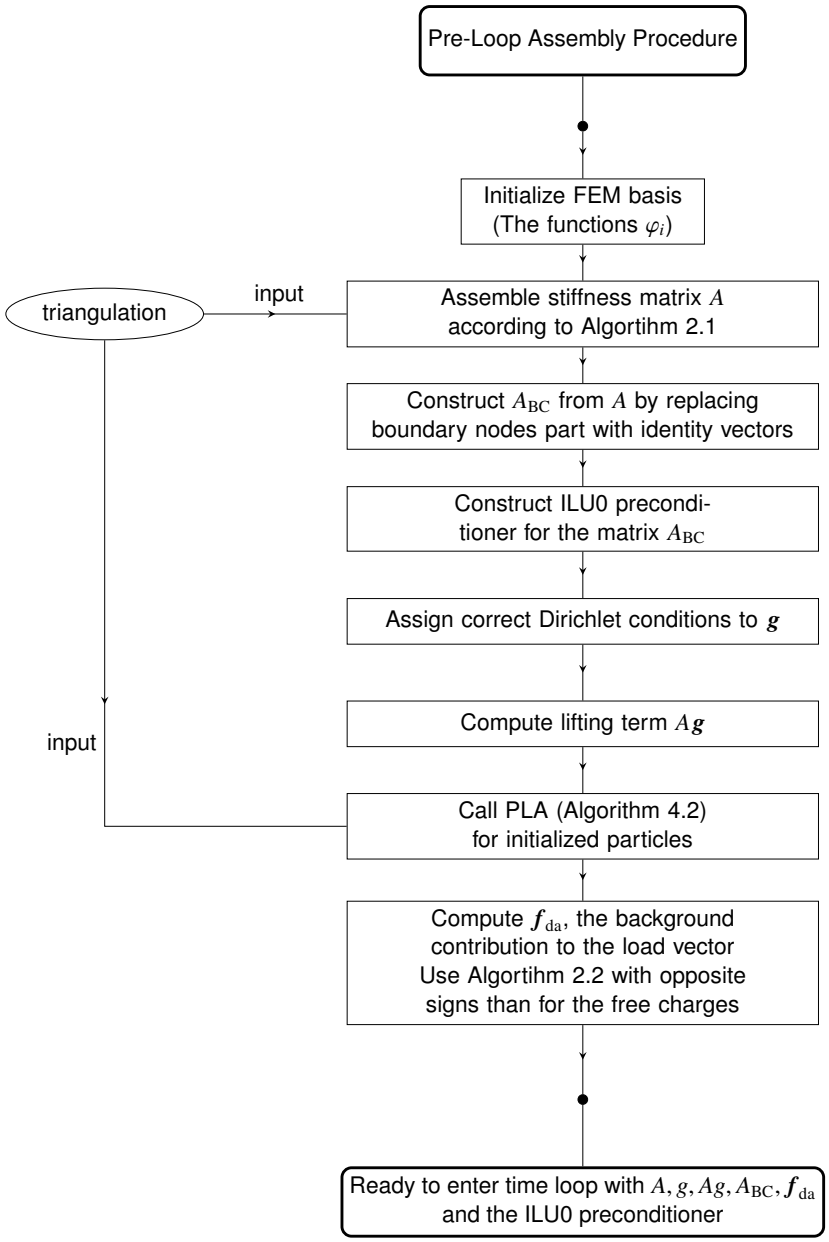


Figure 6.3: The subflow for assembling the vectors and matrices of the linear system which are state-independent. The triangulation is necessary to construct the stiffness matrix. The 3D-PLA is utilized for initializing element positions for particles. With this information, the contribution from the background charge to the load vector can be computed.

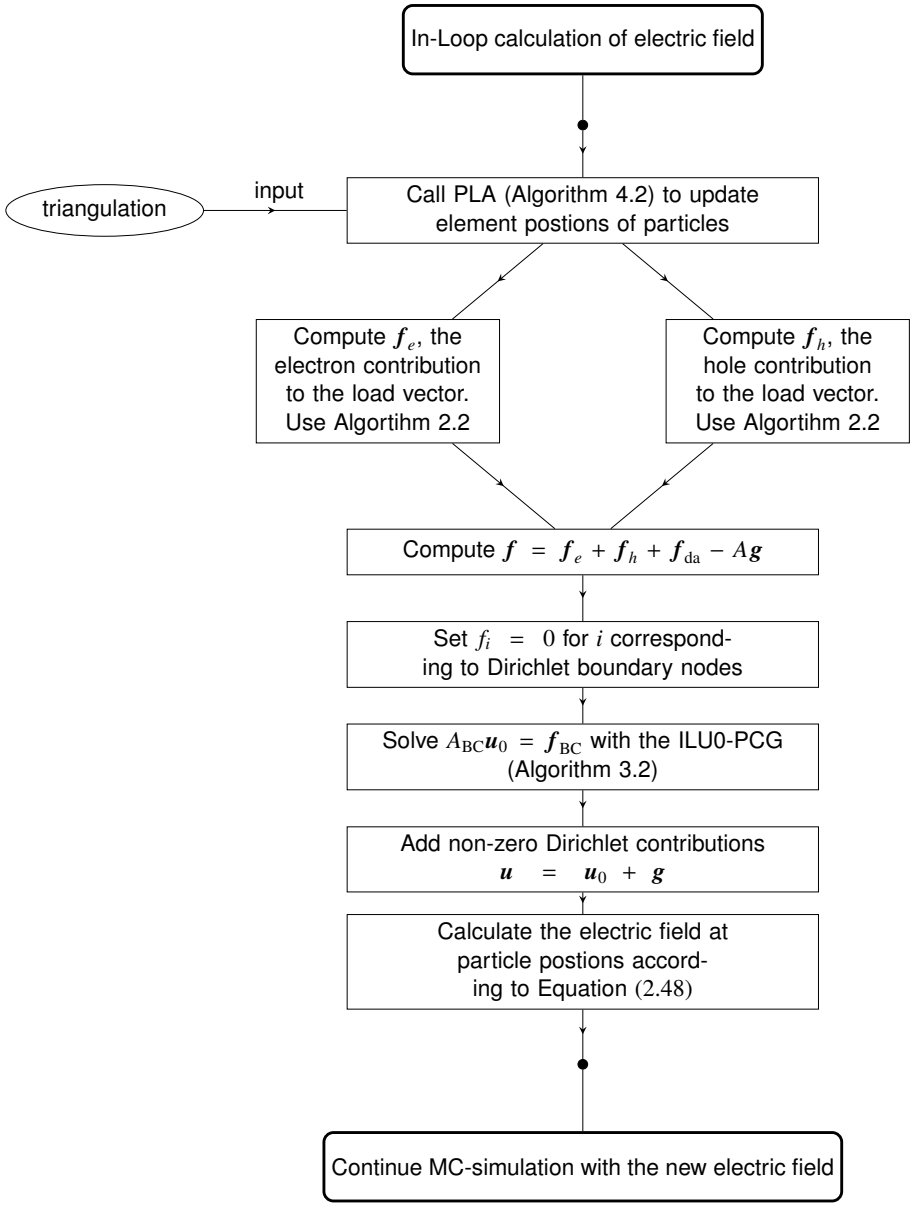


Figure 6.4: A flowchart demonstrating the program flow for the Poisson solving part of the MCS. The load-vector contribution from electrons and holes must be calculated before the complete linear system can be passed to the iterative linear system solver. The background vector $\boldsymbol{f}_{\mathrm{da}}$ was computed during the initialization. The Dirichlet contribution is added to the approximation of the homogeneous problem, and the approximate electric field is calculated from the potential. The simulations continue with the updated field.

7 | Simulations

With the finite element method successfully integrated as a Poisson solver in the FFI-MCS, some results are presented in this chapter to investigate the capabilities of the method, and to point out current weaknesses. Firstly, a simple case study of calculating the potential around a particle in a box is treated in Section 7.1, where numerical evidence of a correct implementation and convergence is given, supporting some of the existence and uniqueness considerations presented in Section 2.6. The APD-model used for bias simulations is presented in Section 7.2 and a short note on scaling within the program is given in Section 7.3. In Section 7.4, results from four different bias simulations are presented and discussed. The performance and workload of the different parts of the program are briefly discussed in Section 7.5, which includes a call-tree with CPU-percentages and number of calls for some of the routines.

7.1 Case Study: Particle in a Box

To evaluate the performance of the method suggested in Chapter 2, a simple case study of a particle in a box has been performed. The finite element approximation to the dimensionless potential around a unit charge in the center of a $[0, 1]^3$ cube is computed. It is compared to the analytic potential (the fundamental solution)

$$u(x) = \frac{1}{4\pi|\boldsymbol{x} - \boldsymbol{x}_p|}, \tag{7.1}$$

with $\boldsymbol{x}_p = (0.5, 0.5, 0.5)$. For the approximation, a Dirichlet condition on the boundary of the cube was imposed by calculating the analytic potential in each boundary node. The potential is approximated in Fortran with the implemented FEM code, and the result is read into MATLAB for visualization and error estimation. In order to estimate the error of the approximation, it is linearly interpolated on a structured grid with $(101 \times 101 \times 101)$ nodes. The analytic solution is directly computed on the nodes in the structured grid. The center node of the grid and its 6 adjacent nodes are not taken into account in this comparison, as the analytic function is discontinuous in the particle position. Three of the approximations computed on grids with different grid coarseness and quadratic elements are shown in a 1D plot in Figure 7.1.

The error $|u - u_h|$ is approximated by using the average L^2-norm as

$$|u - u_h| \approx |u - \tilde{u}_h| = \sqrt{\overline{(u_{ijk} - \tilde{u}_{h,ijk})^2}}, \tag{7.2}$$

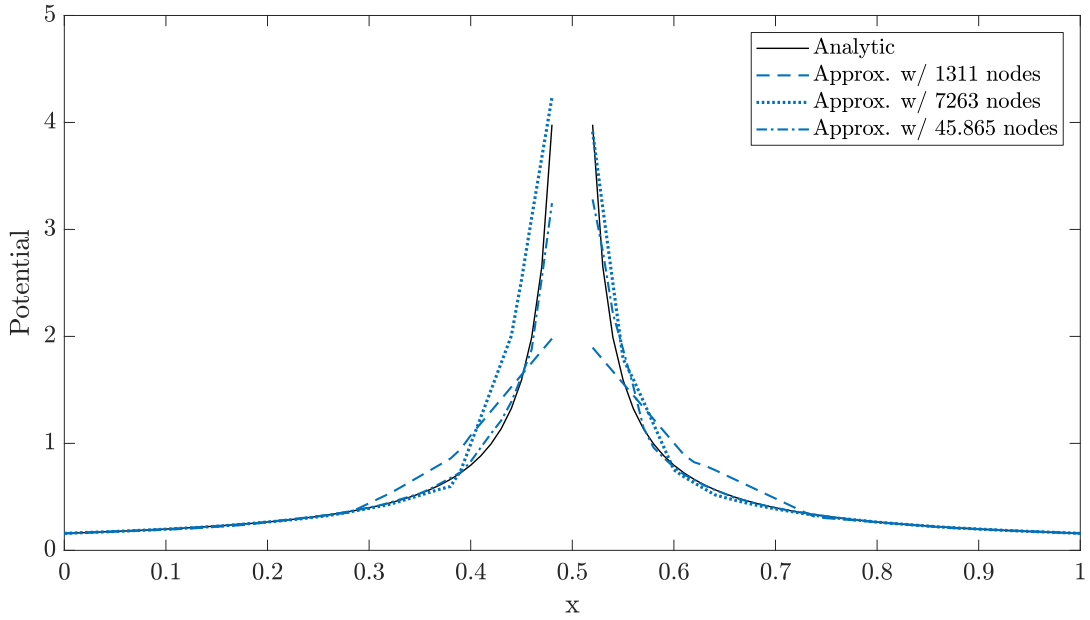


Figure 7.1: The figure shows the analytic one dimensional potential in the ($y = 0.5$, $z = 0.5$)-plane, together with three approximations computed on a grid with 1311, 7263 and 45.865 nodes respectively.

where u_{ijk} is the analytic potential computed in the node x_i, y_j, z_k of the structured grid, and $\tilde{u}_{h,ijk}$ is the approximation interpolated from the scattered nodes in the triangulation onto the structured grid, and evaluated in node x_i, y_j, z_k. The bar denotes the average over all nodes. The interpolation from the triangulation to the structured grid is done with the built in MATLAB-function `scatteredInterpolant`. Some loss of accuracy must be expected in this interpolation procedure. Figure 7.2 visulaizes the calculated error. Comparing the error of the linear and quadratic approximation on the same triangulation, i.e., the same discretization with an equal number of elements, the quadratic approximation has on average slightly better approximation, but the rate of convergence is not improved from the linear approximation. This is shown in Figure 7.2(a). From Figure 7.2(b), it is clear that a comparison of the number of nodes favors the linear approximation in the sense that the error is smaller than the error obtained by the same amount of nodes in the quadratic case. The size of the system of equations to be solved is dependent on the number of nodes, and the time used to compute its solution will thus also be node number dependent.

The error also shows irregular behavior with respect to grid refinement. A possible reason for this behavior is the influence of the particle position within an element, i.e., the distance between element vertices and the particle. To test this hypothesis, a second experiment was performed, with particle positions given at different points on an edge of an element and within the element volume. The test was performed in a grid with 6810 elements, for both linear and quadratic elements. The average error was calculated by disregarding values in the structured grid with analytic potential greater than 5, i.e., nodes within a distance of $|\boldsymbol{x} - \boldsymbol{x}_p| = 0.0159$ from the particle. The error estimates are shown in Figure 7.3. The results verify the hypothesis, where the placement of the particle is seen to influence the error. However, it is not proportional

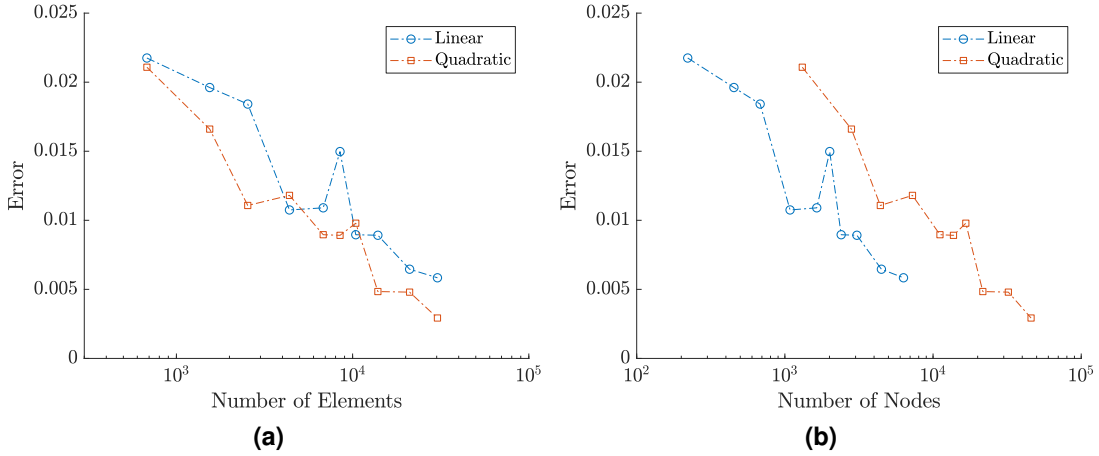


Figure 7.2: The computed average L^2-error for different grids, for both linear and quadratic elements. The error is ploted against the number of elements (a), and against the number of nodes (b). The quadratic approximation is on average an improvement to the linear if applied on the same triangulation, but when comparing a coarser grid with quadratic elements to a finer grid with linear elements but approximately the same total number on nodes, the linear approximation produces a smaller error.

to the distance from the vertex, and particles close to vertices seems to introduce larger errors than particles closer to the center. The relatively small error for large distances in the quadratic case might be caused by the proximity of the additional edge-nodes. The similar behavior for the linear case can be caused by how the particle is weighted to each node, such that a more symmetric weighting increases the accuracy of the approximation. This type of fluctuations will have an impact in all grid refinements, and since particle positions are allowed in the continuum of Ω, this can only be controlled by how the particles are interpolated to the nodes, i.e., if the basis functions are used for particle mesh coupling or another method is imposed.

The general lack of better convergence for the quadratic approximation might occur due to the use of an affine mapping. A possible solution can be to apply an iso-parametric mapping. However, this will lead to the need of higher order quadrature rules to compute the stiffness matrix elements, as the Jacobian will no longer be constant on an element. Further, it introduces the possibility for curved element faces, requiring more advanced particle location methods. Such an extension is unfortunately outside the scope of this thesis.

Even though some irregularity in the error estimates can be observed, the tests confirm that the finite element approximation of the single particle potential converges to the fundamental solution of the equation. These results are adequate, allowing us to extend the testing to the Monte Carlo particle simulations, which are the main area of interest.

7.2 An Avalanche Photodiode Model

To test how the finite element Poisson solver combines with 3D device simulations, bias simulations of a HgCdTe APD is performed. The APD-model chosen for the simulations is

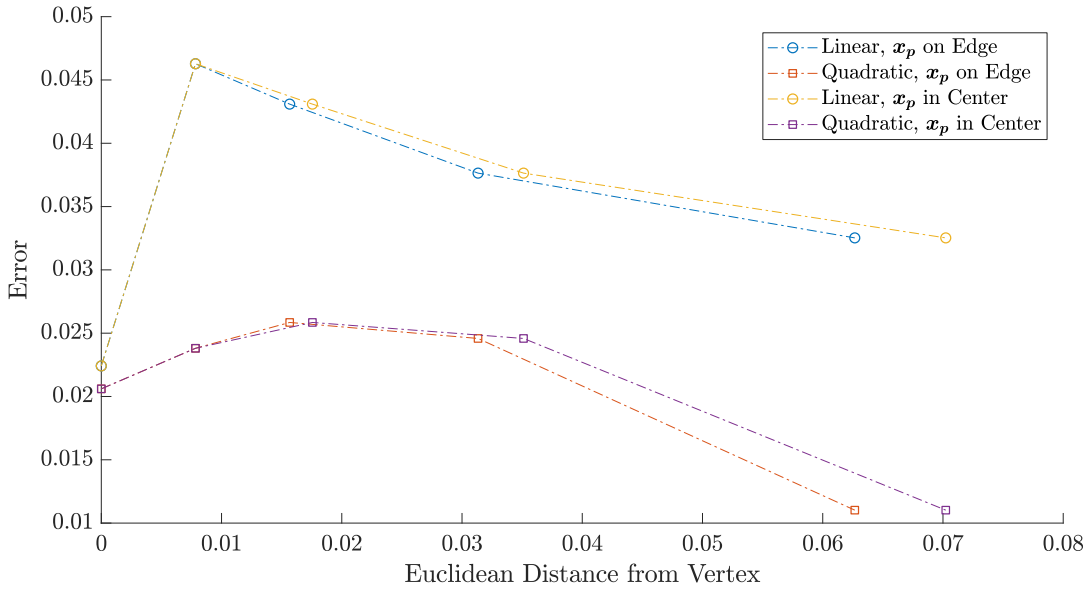


Figure 7.3: A comparison of error with respect to the particle position within an element. The error is plotted against the distance from the closest vertex. Whether the particle is placed in the volume or on an edge does not have a noteworthy impact on the error, but it is apparent that the distance from a grid node does have an influence. In the quadratic case, the approximation shows smaller errors when the particle is close to mid-edge-points. For the linear case, the approximation is best for in-vertex particle placement.

used in research performed by FFI and is presented in the Full Band simulations described by Storebø et al. [66]. A planar view of the simulated device is shown in Figure 7.4. Both the field calculations and the carrier dynamics are performed in 3D.

Research in device simulation is often focused on nanoscale devices and the non-equilibrium transport phenomena that are dominant in such devices. However, it is interesting to investigate the application of MC-simulations on bigger microscale devices with other applications. Simulating larger devices lead to other numerical challenges than those present for smaller ones. Since the number of particles is proportional to the volume of the device, a larger device naturally requires simulation of more particles, and thus more computational resources. In addition, the self-consistency provided by the Poisson solver applied to a larger device will only scale down to the resolution of the discretization mesh, and requires the simulation of superparticles, as real particles will not necessarily be detected by the Poisson solver. However, when simulating weak infrared signals, it is necessary to simulate the behavior of the real particles. To overcome this, the suggested solution is a split up of the simulation in two parts, where a bias simulation is performed with the use of superparticles and a Poisson solver, to obtain a steady state solution of the electric field under the applied bias. The small signal analysis is then performed with the applied frozen field obtained from the bias simulation [66]. Testing the MCFEM on APD bias simulations will uncover some of the possibilities that lie in the application of MC-simulations coupled with self-consistent finite element computations of the electric field.

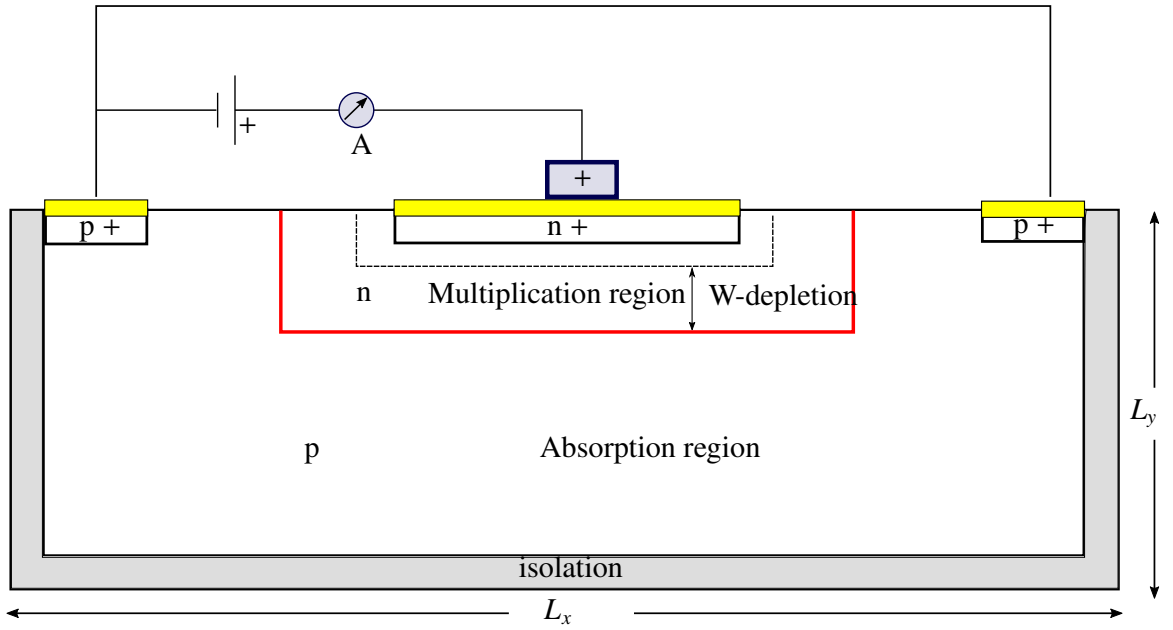


Figure 7.4: Planar sketch of the APD that will be the simulation domain. The third dimension extends into the paper plane. The gray isolation layer is included in the simulations in order to later be able to apply an alloy gradient within the device, without violating the Neumann-condition on the boundary. The red line is the interface between the p- and n-doped regions.

7.3 Scaling within the Program

When deriving the numerical scheme for the Poisson equation, a scaling was performed to obtain the dimensionless problem given in Equation (2.8). In the base code for the FFI-MCS, each variable is fitted with a unit adapted to a suitable convention for the calculation it is a part of. This is done to easily be able to verify the value of each variable from a physical perspective by comparing it to a known value in similar applications. In addition, the units are chosen to avoid under- and overflow. The MC-kinetics are implemented to work on unscaled particle positions, denoted here by $\boldsymbol{x}_p^*$, which in the modeled APD are in the order of magnitude $\sim 10^{-6}$ m. However, this is not consistent with the scaled mesh necessary for good condition numbers and numerical float precision in the finite element discretization and assembly process. Thus, for each Poisson-solving step, the scaled particle positions are calculated as

$$\boldsymbol{x}_p = \frac{\boldsymbol{x}_p^*}{L} = 10^6\,\mathrm{m}^{-1} \cdot \boldsymbol{x}_p^*, \tag{7.3}$$

where we have used the lenght scale L introduced in Section 2.2. The scaled positions must be updated before the call to the 3D-PLA, for consistency with the dimensionless grid. Further, the parameter α defined in Equation (2.6) is given a value according to

$$\alpha = \frac{q_{\mathrm{sup}}}{\varepsilon_0 \varepsilon_{\mathrm{r}}}, \tag{7.4}$$

resulting in a potential $u^*(\boldsymbol{x}) \in [A, B]$, where A denotes the minimum applied Dirichlet condition and B the maximum, such that the solution is obtained without the need of scaling the boundary conditions. The potential u^* is then directly calculated in its physical unit Volt. The components of the electric field are calculated individually for each particle and can be directly used in the MC-kinetics, which continue to use the unscaled particle positions with the physical unit meter.

7.4 Bias Simulations

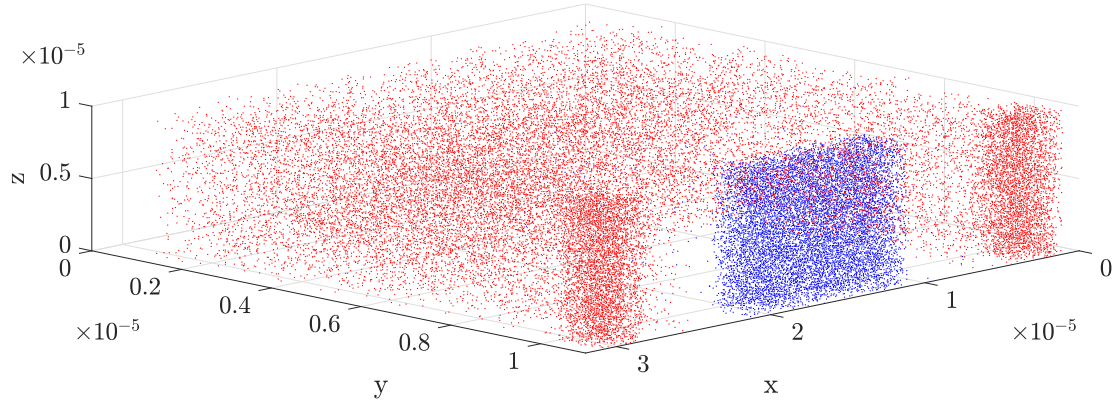
Bias simulations of the APD described in Section 7.2 will in a full simulation process be applied to obtain a steady state potential, which can be used in the following small signal analysis. The bias simulation is a simulation of a transient state, and a steady state will, ideally, appear after simulation of 70 ps to 80 ps physical time, corresponding to 80.000 to 160.000 simulation steps, depending on the time step used in the simulations. A special concern is the memory consumption under these simulations. Longtime steady simulations have not been achieved during this work, among other things due to limited computational power, memory and time. There is in addition still the need for debugging and finishing touches within the new MCFEM, before all the desired functionality can be obtained.

The results included and discussed here are shorter simulations, ranging up to $\sim$ 10 ps, where it is possible to observe the initial particle behavior and the calculated potential. The simulations have been carried out on four different triangulations. Common parameters for the simulations are given in Table 7.1, and each simulation uses initial particle distribution as shown in Figure 7.5. The initial distribution of particles is uniform according to the specified doping density in each area, and the doping background is set to the initial position of free particles and given the opposite charge. The Poisson solver is called every second time-step in all the presented simulations.

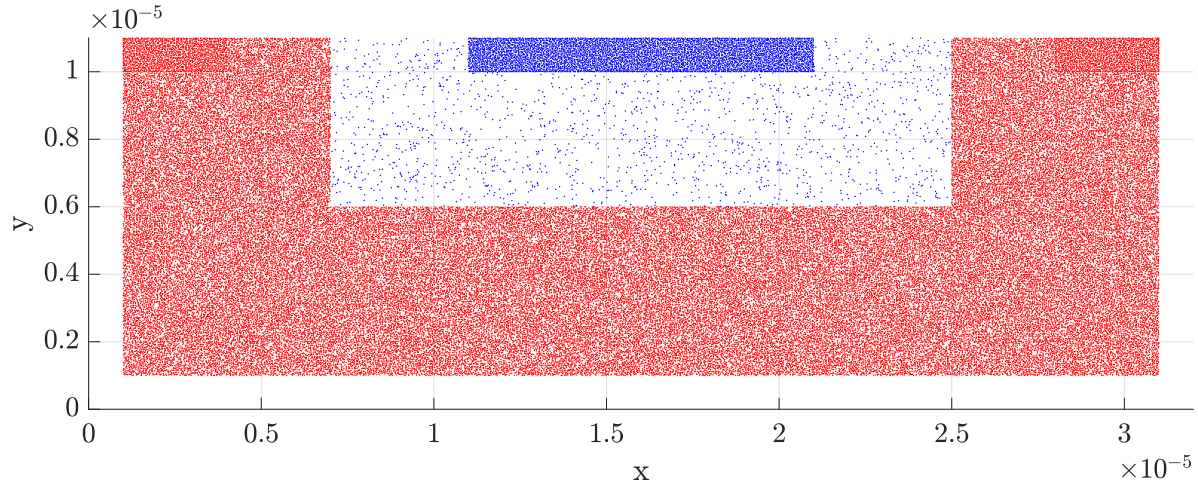
The contacts are placed on top of the device and extend over the whole z-surface, as visualized by color in Figure 7.6(a). A reverse bias of -7 V is applied by imposing a constant Dirichlet boundary of -7 on the p-contacts, and a constant Dirichlet boundary of 0 on the n-contact.

Table 7.1: Common parameters for the four presented simulation runs of MCFEM.

Parameter	Value	Explanation
L_x	30 µm	Device length in x-direction, without isolation
L_y	10 µm	Device length in y-direction, without isolation
Δy	0.1 µm	Depth of neutral area below contact
η_{n+}	$2.5 \cdot 10^{17}\,\mathrm{cm}^{-3}$	Doping density in $n+$ region
η_{n-}	$5.0 \cdot 10^{14}\,\mathrm{cm}^{-3}$	Doping density in $n-$ region
η_{p+}	$2.5 \cdot 10^{17}\,\mathrm{cm}^{-3}$	Doping density in $p+$ region
η_p	$2.0 \cdot 10^{16}\,\mathrm{cm}^{-3}$	Doping density in p region
U_p	-7 V	Applied voltage on hole-injecting OCs
U_n	0 V	Applied voltage on electron-injecting OC



(a) Initial particle positions in a 3D view.



(b) Initial particle positions in a 2D view.

Figure 7.5: Initial particle positions from both a 3D (a) and a 2D (b) perspective. The uniform distribution of initial positions according to doping density is common to all simulations presented here. Blue points represent electrons and red points represent holes. Only 20 % of the simulated particles are shown. Higher doping concentrations in the $p+$ and $n+$ regions can be observed on top of the device. In the 3D-view, the axis are turned such that the top of the device is facing outwards and to the right.

In Figure 7.6, 7.7, 7.8 and 7.9, a set of particle positions and a potential from the four different simulations are visualized. Each figure includes the triangulations utilized to obtain the visualized results. The first three devices are isolated and performed with quadratic elements, while the fourth simulation is performed without isolation layer and linear elements. Varying parameters for the simulation are shown in Table 7.2, where the different cases are referenced according to the figure where they are presented.

7.4.1 Particle Behavior

The MCFEM is a particle simulator, and a statistic of interest gathered from the program is the movement of the particles. In an APD under a large reverse bias, the majority of the free electrons will be absorbed by the OC in the $n+$-doped region, and a depletion zone with few free

Table 7.2: Varying parameters between the different simulations. Each run is referenced by the number of the figure where the result is presented.

	Figure (Simulation)				
Parameter	**7.6**	**7.7**	**7.8**	**7.9**	**Explanation**
L_z	10 µm	10 µm	10 µm	8 µm	Device length in z-direction
L_{iso}	1 µm	1 µm	1 µm	0 µm	Isolation layer thickness
N_e^{init}	500,000	250,000	1,000,000	250,000	Initial number of superelectrons
N_h^{init}	1,098,425	549,212	2,196,850	709,725	Initial number of superholes
$\frac{N_{\text{real}}}{N_{\text{sup}}}$	~ 50	~ 100	~ 25	~ 65	Real particles per superparticle
Δt	2 fs	1 fs	1 fs	1 fs	Time step between iterations
t	3 ps	5 ps	4 ps	6 ps	Represented time in visualization
N_h	75,165	449,289	245,710	97,746	Number of nodes in triangulation

carriers will be formed on the n-side of the pn-junction. To verify that this behavior also occurs in the simulations performed with MCFEM, the particle positions are written to file at user-defined timesteps. For each simulation, the particle positions are visualized in Figure 7.6(b), 7.7(b), 7.8(b) and 7.9(b). Since the particle behavior in the simulated APD is on average planar, the formed depletion zone is visualized in the 2D-view. The general particle behavior is consistent throughout the performed simulations, independent of element order and grid refinement. The initial particle behavior corresponds to what is expected, where a depletion zone is formed in the n-doped area of the device.

7.4.2 Effect of Grid Refinement

Another result of interest is the calculated potential in the different cases. These are visualized in Figure 7.6(c), 7.7(c), 7.8(c) and 7.9(c) for each of the simulations. How the solution reacts to a different number of superparticles and grid refinement is especially interesting, but too few simulations have been carried out in order to draw conclusions on these areas. Ideally, an even larger number of superparticles should be simulated, but memory restrictions made this unfeasible.

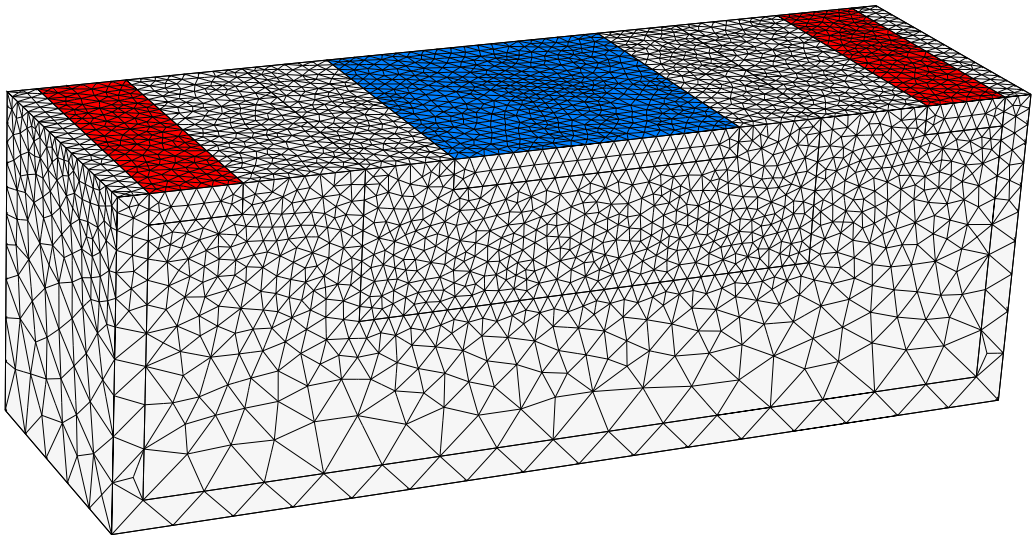
When the depletion zone has been formed in the device, a clear potential drop along the pn-junction is expected. The visualizations of the potential are the average potential in the z-direction, and all potentials, except for the one in Figure 7.7(c), show lack of potential drop for the junction in the middle of the device volume. In addition, a potential close to the maximum 0 V is calculated in the p-doped region far from the contacts, which is unexpected.

The potential drop at the pn-junction is evident on the refined grid in Figure 7.7, but the refinement also introduces significant noise in the p-doped region, which propagates to the boundaries along the isolation layer and also to the imposed Dirichlet conditions. The simulation on this refined grid was performed with very few superparticles, which could also contribute to the arising noise. This hypothesis will, however, need further investigation. What is obvious, is that the combination of an extremely large number of nodes with few particles is introducing

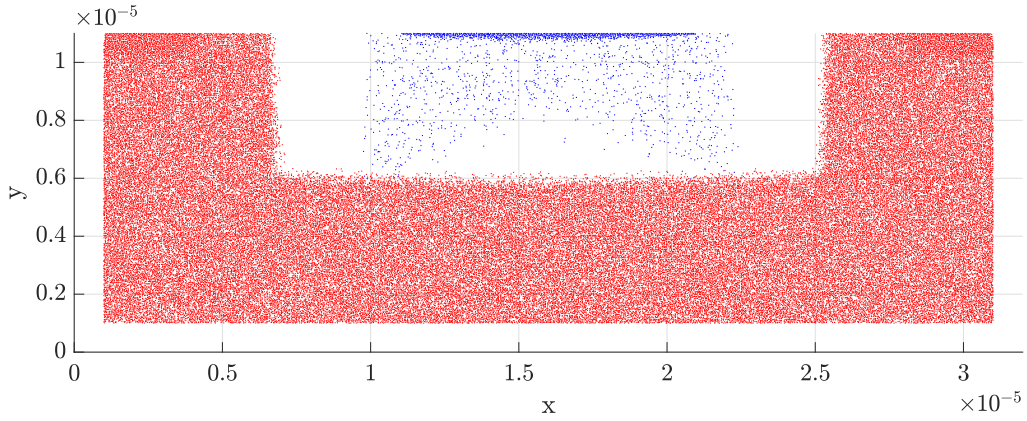
strange responses.

Further, it is clear that the simulated isolation layer between the device volume and the imposed Neumann condition is not working as intended when simulating on the refined grid. The potential drop in the isolation layer, which is clearly unphysical, causes holes to move towards the boundary, giving a surplus of positive charge along the isolation layer, amplifying the change of charge density from an element in the doped region to a neighboring element in the isolation layer. Applying the Neumann boundary directly to the doped region can remedy this effect since there then will be no electric field perpendicular to the region with particles. To apply the Neumann condition directly on the free surfaces of the doped regions are justified by Hockney and Eastwood [35]. This was tested for the simulation shown in Figure 7.9. This simulation was performed with linear elements, and the potential drop could not be observed. Running a simulation on a similar grid with quadratic elements might yield interesting results. However, it should be kept in mind that the isolation layer is necessary when applying an alloy gradient, and it should not cause the unstable behavior that can be observed in Figure 7.7(c), as it is neutral. The adjacent elements in the region with free carriers should also be close to neutral, so the expected behavior is close to zero change in the potential across this interface.

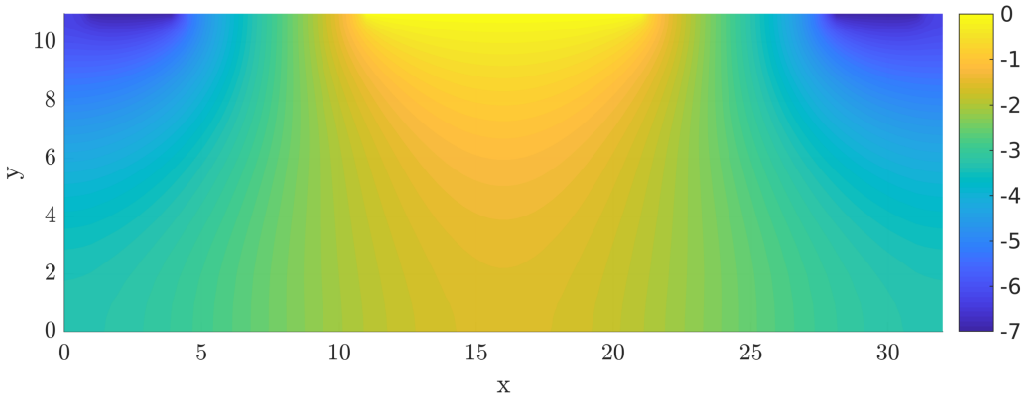
Examining further how the number of superparticle correlates with grid refinement and searching for the optimal global grid taking into account local grid refinement should be addressed in the further development of MCFEM. The noise on the refined grid might be reduced by replacing the discrete background charge with a continuous function within each doped region. Explicit smoothing of the potential can also be considered if other noise-minimizing measures are fruitless. To the best of our knowledge, there has been little research on the combination of FEM in 3D applied to MC-simulations of large-scale devices, and it is clear that more research is necessary to find the capabilities and limitations for such a combination.



(a)

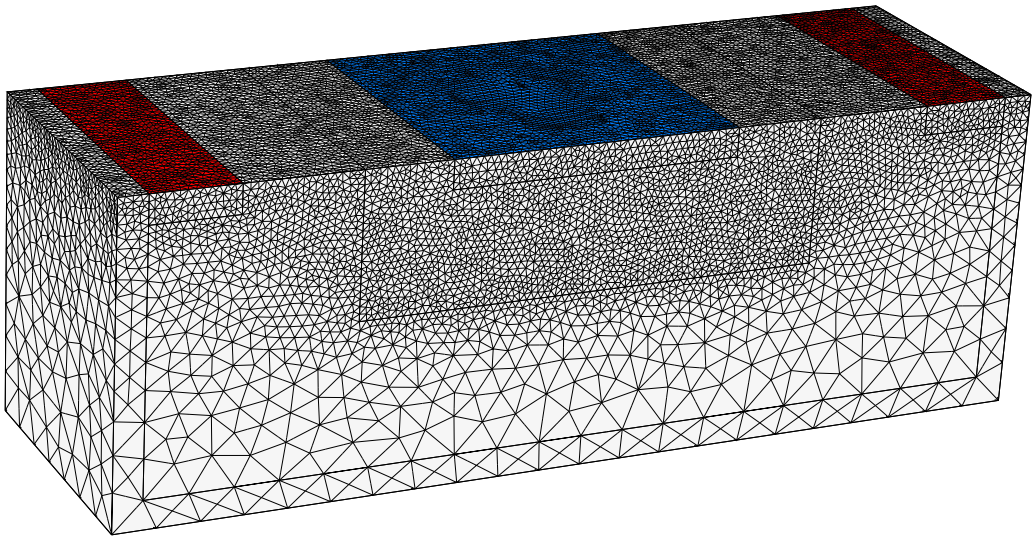


(b)

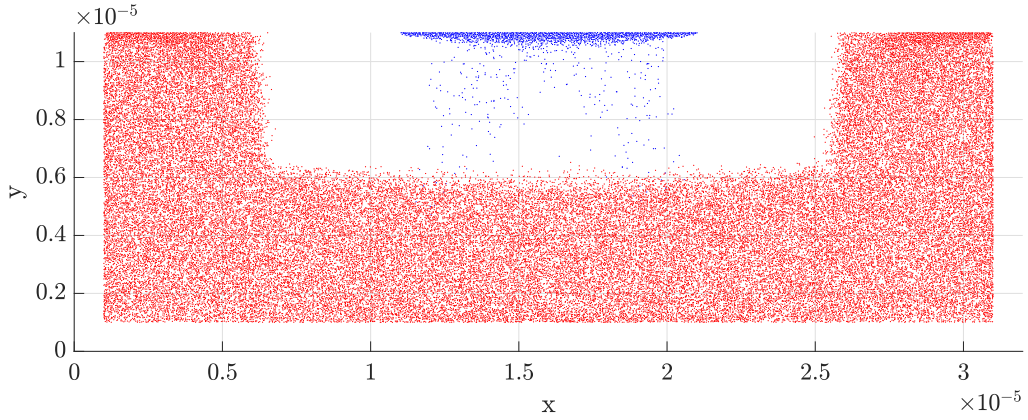


(c)

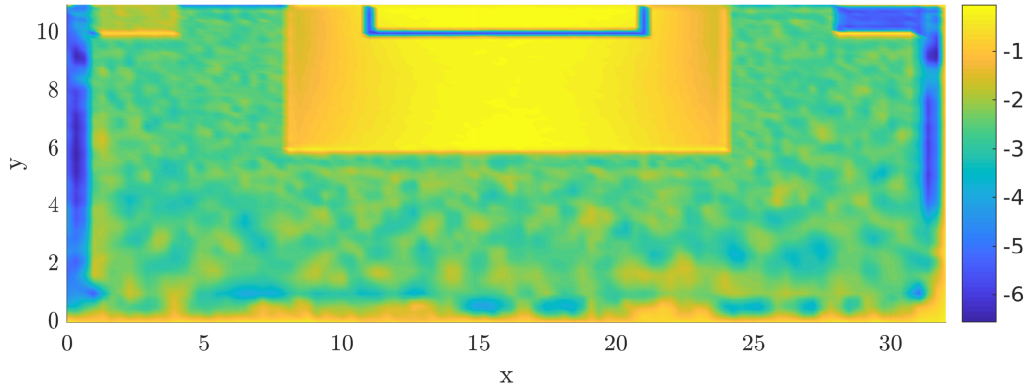
Figure 7.6: In (a), the grid of the simulation is shown. It consists of 51,207 quadratic elements and 75,165 nodes. The blue surface represents the *n*-contact, and the red surfaces the *p*-contacts. The particle positions after 3.0 ps is shown in (b), with holes represented by red dots and electrons by blue. The initial number of electrons and holes in this simulation was 500,000 and 1,098,425 respectively. The corresponding average potential in z-direction is shown in (c). The potential in smooth and the noise is minimal on this coarse grid. However, the potential drop expected at the *pn*-junction does not appear.



(a)

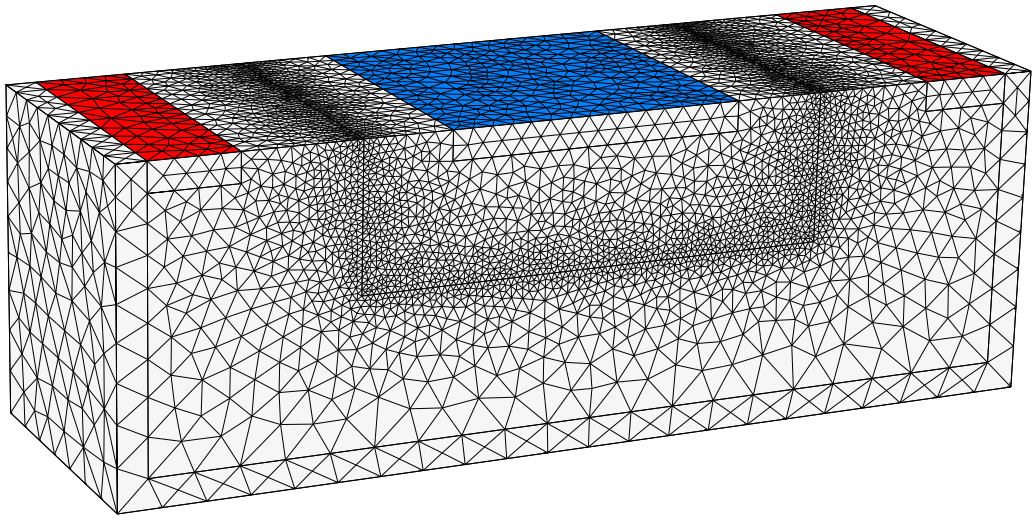


(b)

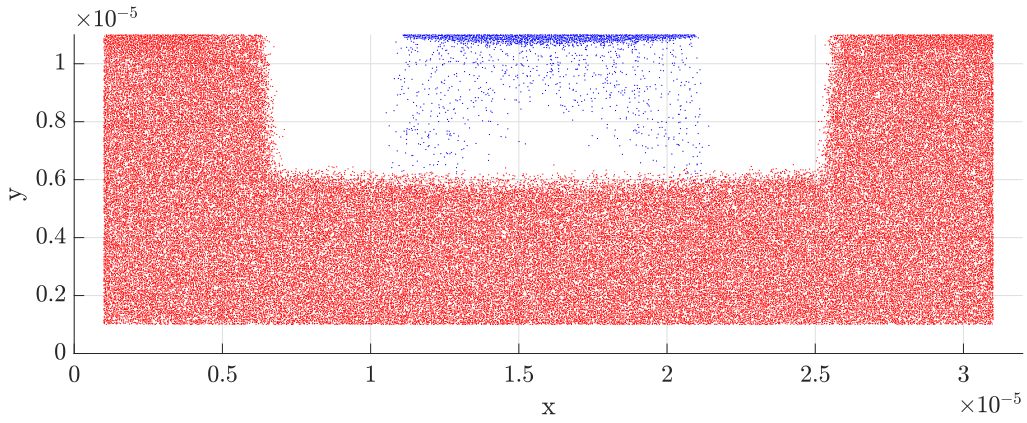


(c)

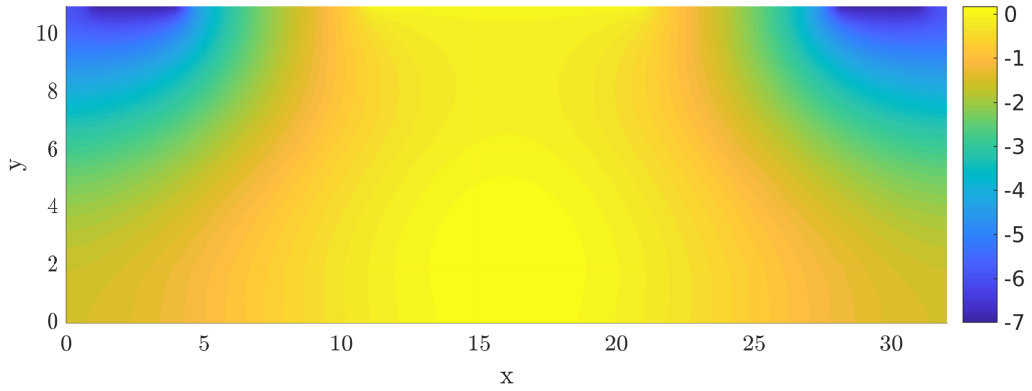
Figure 7.7: In (a), the grid for simulation is shown. It consists of 321,303 quadratic elements with a total of 449,289 nodes. The particle positions after 5.0 ps is shown in (b), and the corresponding potential in (c). The initial number of electrons and holes was 250,000 and 549,212 respectively. The potential drop is clear over the *pn*-junction, but the refinement introduces noise in the *p*-doped region. This can be due to the combination of few superparticles and small elements. The additional potential drop in the isolation layer is also unphysical, and causes holes to be attracted to the free surfaces.



(a)



(b)



(c)

Figure 7.8: In (a), the grid for simulation is shown. It consists of 177,074 quadratic elements with a total of 245,710 nodes, and is refined along the *pn*-junction. The particle positions after 4.0 ps is shown in (b), and the corresponding potential in (c). The initial number of electrons and holes was 1,000,000 and 2,196,850 respectively. As for the simulation in Figure 7.6, no potential drop is observed along the *pn*-junction in the volume of the device. The relatively large number of superparticles in this simulation does not seem to affect the potential in an observable way.

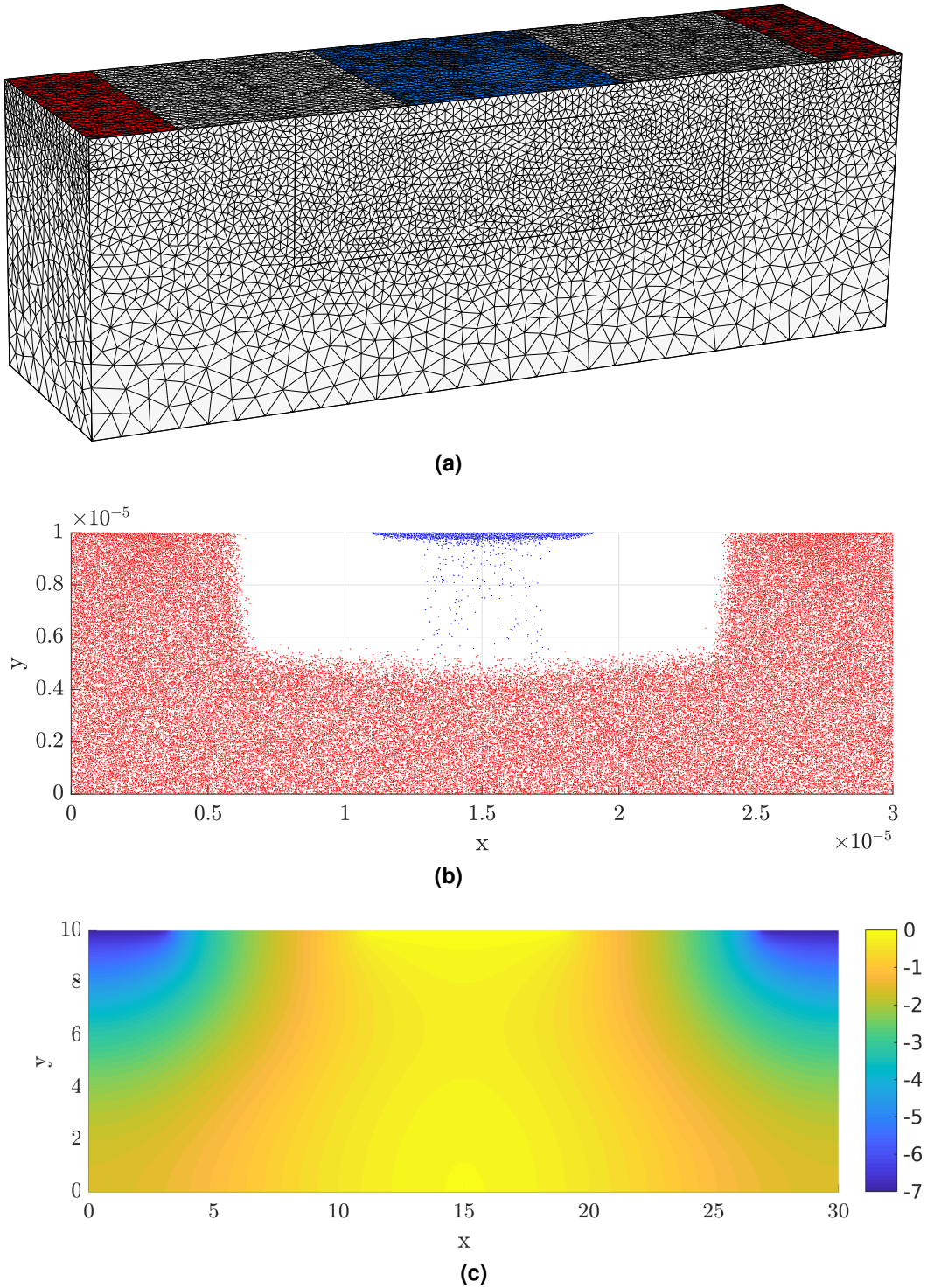


Figure 7.9: In (a), the grid for simulation is shown. It differs from the other grids as the isolation layer is removed, in addition to a slightly shorter extension in the z-direction and a smaller n-contact. It consists of 554,726 linear elements with a total of 97,746 nodes. The particle positions after 6.0 ps is shown in (b), and the corresponding potential in (c). The initial number of electrons was 250,000 and the number of holes was 709,725, which differs from the number in Figure 7.7 due to the change of specified dimensions. The removal of the isolation layer does not influence the simulation in this case.

7.4.3 Effect of Element Order

In addition to the four simulations already discussed, a small comparison of linear and quadratic elements is provided here. The performance of linear and quadratic elements was previously discussed for the single particle case study in Section 7.1. Similar results were found when comparing the effect of applying different element orders in MCFEM. Figure 7.10 shows contour plots of the potential for a simple comparison of the use of linear and quadratic elements on a grid with an equal number of elements at the same timestep, where the same initial positions of particles have been used. Only a minor improvement in smoothness can be observed from the linear to the quadratic approximation, strengthening the assumptions previously presented.

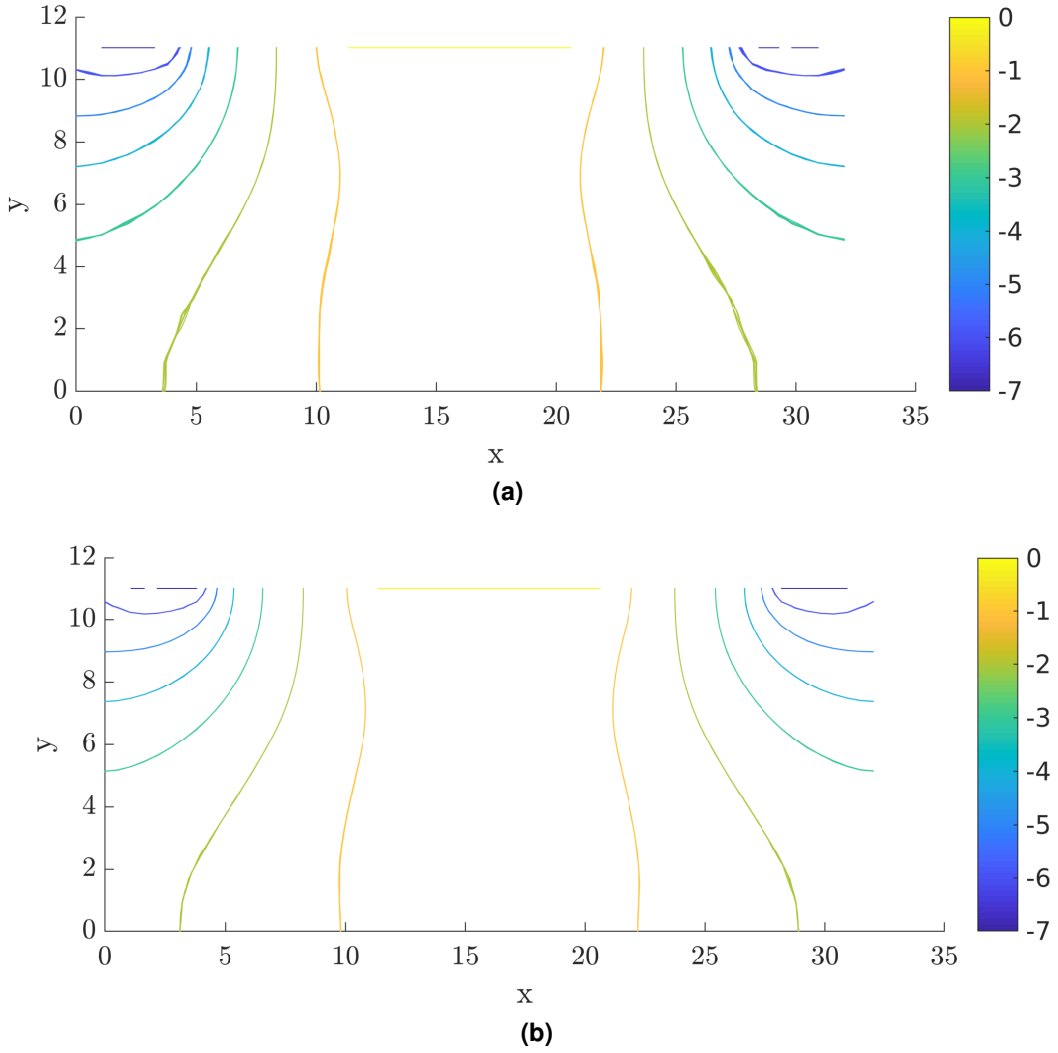


Figure 7.10: Comparison of approximated potential with linear and quadratic elements, on a coarse grid, at $t = 7.5\,\text{ps}$. Only a slight reducement of smoothness can be observed, confirming the previous discussion for the particle in box case study. The particle behaviour under the different approximations are very similar to what is shown previously.

7.5 Workload for the Program

For optimizing measures, it is useful to be aware of the bottlenecks and the most time-consuming parts of the program, such that effort for optimization can be concentrated at the parts with the most to gain. The profiling tool *gprof* is used to extract run-times and a complete call-tree for a typical, but relatively short, simulation. The parameters with the most impact on the simulation time are listed in Table 7.3, and Figure 7.11 shows part of the call-tree for this simulation. The included nodes are those consuming more than 2.5% of the total CPU-time. The PLA-nodes are also included because it is interesting to see that these are below this threshold and thus not the bottlenecks they were assumed to be. This is in contrast to the 2D-FEM solver implemented by Åsen, where the PLA was in fact the main bottleneck [5].

Table 7.3: Parameters for the profiling run presented in Figure 7.11

Parameter	**Value**
Timestep, Δt	2 fs
Poisson Solution	Every 2nd timestep
Simulation steps	1000
Initial number of electrons	500,000
Initial number of holes	1,098,425
Element order	Quadratic
Number of nodes	75,165
Non-zero entries in A_{BC}, $N_z(A_{\mathrm{BC}})$	1,944,787

As expected are routines with many calls, such as *free flights* and *hole scattering* represented in the tree. More interesting is the fact that the insertion of particles at the OCs consumes above 11% of the total CPU-time, which is unexpected when considering the relatively small task of this routine. Thus, this part of the program is an obvious candidate for improvement.

The linear system solver is the bottleneck when solving Poisson equation and consumes 15.1% of total CPU-time. It is expected that the complete process of calculating the self-consistent electric field by solving the 3D Poisson equation in ensemble MC-simulations will require up to 90% of the total CPU-time for the simulations, leaving about 10% to the MC-kinetics [42]. From the information presented in Figure 7.11, the approximate CPU-time for the sum of routines directly connected to the Poisson solver and the calculation of the electric field can be calculated to below 30%. Some additional contribution might be added from the self-time in the main program. This indicates that the implemented FEM is an efficient choice of Poisson solver in this particular case. It might also be expected that simulation of bipolar materials with both electrons and holes uses more processor time in the MC-kinetics than a simulation of unipolar materials. Even so, the Poisson solver is a major bottleneck in the self-consistent simulations, and looking for more efficient solvers, e.g. by considering parallel iterative system solvers such as multigrid methods or performing a decomposition of the domain to perform the solution on each subdomain in parallel , are possible measures to further reduce the time spent on the electric field calculations [53, 56].

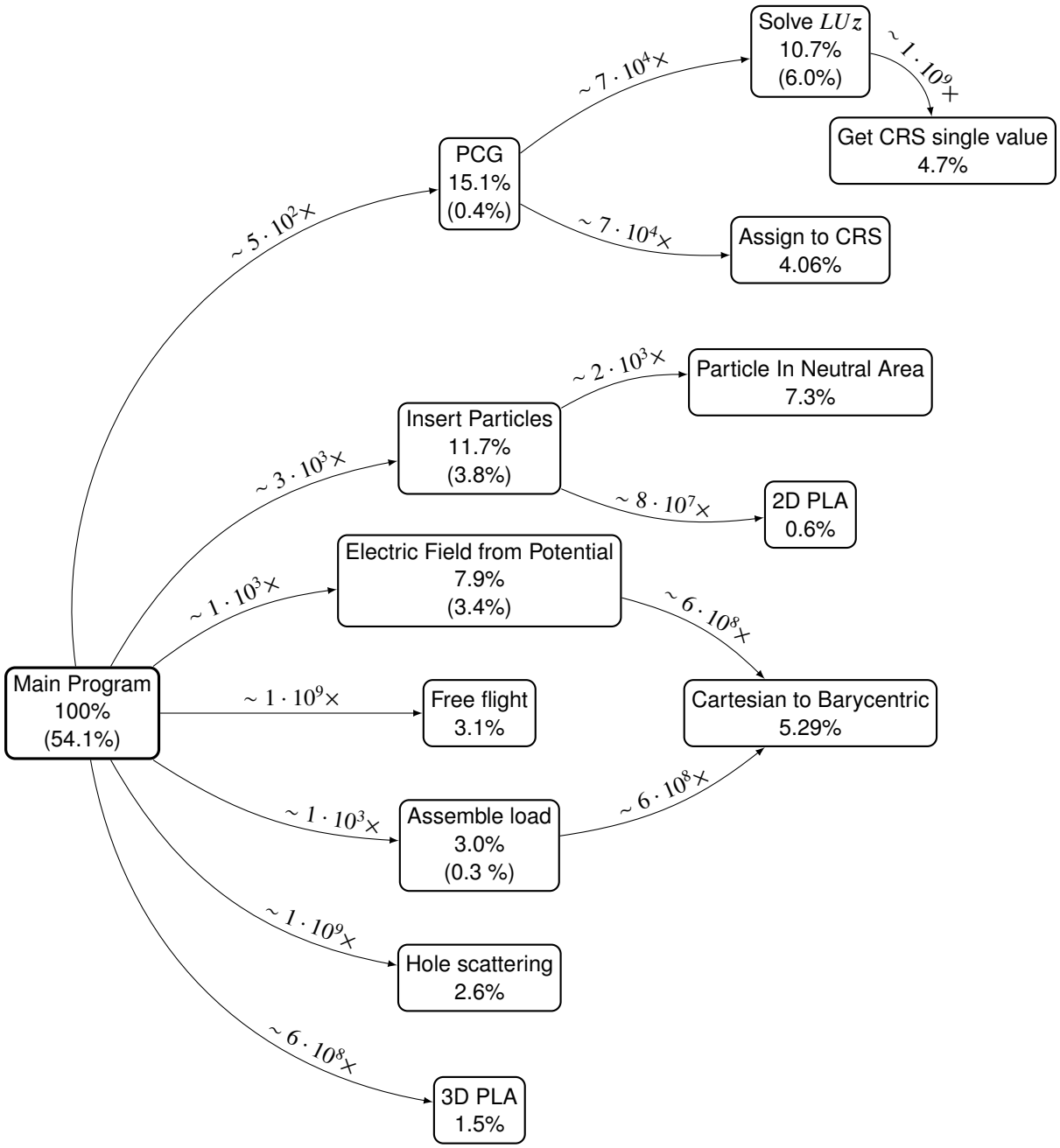


Figure 7.11: A call-tree for evaluating bottlenecks of the program. Each node contains a percentage representing the proportion of total run time spent in the routine and its children. The percentage shown in parenthesis in each node is the self-time for each routine. Above each branch is an approximate number of times a given routine has been called from its parent. From the tree, the three main bottlenecks are found to be, most likely, the particle position scaling in the main program, the PCG and the insertion of particles at the OCs.

The small function performing the *barycentric to cartesian* mapping consumes a total of 5.29% of total CPU-time, but it must be taken into consideration that it is called above 10^{10} times during the simulation. It is doubtful that much improvement is possible, but bearing in mind how values are stored in memory might lead to some improvements.

Further, it can be observed that the self-time spent in the main program is above 50%, and without a line to line profiling, it is difficult to point out the exact bottlenecks. One definite candidate is the scaling performed to solve Poisson equation. This occurs because the particle positions in the base code are not originally scaled, so for each Poisson solving time step, each particle position is scaled to its corresponding dimensionless value before the solver, and then scaled back after the solution is found. A consistent scaling for the particle positions within the entire program is thus an important measure for reducing CPU-times.

In addition to *gprof*, there are several other possible profiling tools freely available. As an example, *Valgrind* is used to evaluate and minimize memory leakage in MCFEM. If the time and memory consumption of MCFEM can be reduced, it will be easier to use the software for research in the future.

8 | Further Work

As the program during this work has seen its first full 3D Poisson solver, work remains before the implementation can be seen as robust, accurate and efficient, and a notable amount of testing remains before it can be ensured that the program fulfills all its purposes. Nevertheless, this section contains further paths for improvement of the MCFEM, where the propositions arise from observations made under testing of the program or are general ideas from the studied literature.

Large CPU-time and memory requirements for the program are currently a major issue, and effort should be made to decrease this by further optimizing the source code. It is natural to improve CPU-times by distributing the workload on several cores by appropriate use of OpenMP [16] and MPI [28], which are parallelization tools that include Fortran-interfaces. The Poisson solver is the primary candidate for parallelization. Performing domain decompositions and implementing parallel iterative solvers, such as a suitable multigrid method, should be considered [53, 56]. A domain decomposition would require handling of interfaces for particle movement across the different subdomains, introducing a need to adapt the PLAs, e.g. as suggested by Capodaglio and Aulisa [12]. Parallelization of MC-kinetics will require more work since certain scattering mechanisms, such as carrier-carrier scattering, depend on exchange of information between particles. This requires good and correct synchronization for efficient and scalable parallelization.

One of the current bottlenecks, which also requires memory, is the inconsistency introduced with the need for scaling of the particle positions in the Poisson equation, while the rest of the MCS uses the unscaled particle positions. This adds an extensive amount of floating point operations to the program, which could be avoided with a consistent scaling throughout the program. Such an improvement should be of high priority.

When the program has seen improvements in memory and CPU-time usage, the stability issues for longer simulations need to be addressed. Both analytic and technical insight should be sought to increase the knowledge on the combination of FEM with MC-simulations of large devices. Some of the stability issues might arise from the suggested injection routine for particles at OCs, so this must be subject to thorough testing.

If smaller devices under low applied voltage are to be simulated in the future, there will be a need for an improved electric field interpolation, which aims to minimize self-forces. A first improvement would be to interpolate the electric field taken into account node contribution

from neighboring elements by applying a modified scheme for the electric field,

$$\boldsymbol{E}_h(\boldsymbol{x}_p) = -\sum_{K \ni \boldsymbol{x}_i} \omega_{K,i} \left[\sum_{j \in K} u_j \nabla \phi_j(\boldsymbol{x}_p) \right]. \tag{8.1}$$

Different weights, $\omega_{K,i}$, dependent on the current node $\boldsymbol{x}_i$, can be applied to each neighboring element K, before summing the contribution from each basis functions for the nodes in that element. Another measure is to apply a reference potential in the nodes of the grid. The reduction of self forces in MC-simulations of particle transport is deliberately discussed in the work by Aldegunde et al. [2], Aldegunde and Kalna [3], and Aldegunde et al. [4], and was also mentioned by Åsen [5] in his work with the FFI-MCS.

The comparison of linear and quadratic elements in Section 7.1 and Section 7.4.3 shows only minor improvements with the use of quadratic elements, and even advantages with applying finer grids with linear elements rather than quadratic elements on coarser grids. Refinement of the grid must also be weighted against increased noise for particle simulations. With these observations, it is natural to suggest the extension to isoparametric mappings when applying quadratic elements, to see if this can improve convergence properties. The use of isoparametric mappings on the elements is another alteration requiring adaption of the PLAs [12]. In addition should the search for and use of optimal meshes be considered for higher accuracy and improved efficiency.

With a broader perspective on the complete MCS, applying quantum corrections through a solution of the Schrödinger equation and including wave propagation would certainly extend the capabilities of the program. Using FEM for these tasks will require more analysis, research, and implementation of new functionality, but much of the basic needs are contained within the new MCFEM.

To summarize, the following list includes keywords for improvements.

- Stability
- Lower memory-requirements
- Domain decomposition
- Parallelization
- Consistent scaling of particle positions
- Improved electric field interpolation
- Isoparametric elements for quadratic basis functions
- Optimal mesh generation
- Including wave propagation and quantum corrections

9 Conclusion

A finite element Poisson solver for the calculation of the three-dimensional (3D) electric field in self-consistent particle simulations has been implemented in Fortran and integrated into an existing Monte Carlo simulator for particle transport developed at the Norwegian Defence Research Establishment (FFI). This resulted in a new program structure called Monte Carlo software with finite element Poisson solver (MCFEM). This new program has been tested by performing bias simulations of an avalanche photodiode on different grid refinements with the use of both linear and quadratic polynomial basis functions, with mixed results. Further improvements to the program are necessary in order to obtain long-time stable simulations, but the solver shows promising results for further development. The solver consumes below 30% of the total CPU-time for the program, which is only one third of what is expected for 3D Poisson solvers in Monte Carlo transport software.

Use of unstructured grids in particle simulations require a robust and efficient algorithm for point location. This was implemented for both two- and three-dimensional cases, with the support of an additional implemented triangulation class. This class was constructed to hold information about unstructured grids, including neighboring elements which are used to traverse through the triangulation in the search for particles.

The two-dimensional (2D)-point location algorithm was implemented to handle injection of carriers at Ohmic contacts (OCs). An injection routine based on neutrality in local prisms extrapolated from the surface mesh of the contact was suggested. This approach for handling boundary conditions at the OCs will need further investigation in order to evaluate how well it resembles physical characteristics of devices.

The linear system arising from the finite element approximation of the Poisson equation was solved using a Conjugate Gradient method preconditioned with an incomplete LU factorization. Special storage schemes were implemented to construct and store the matrices of the linear system and were seamlessly combined with the preconditioned conjugate gradient method (PCG). The performance of the PCG was compared to the performance of a simple conjugate gradient scheme without preconditioning, and a bi-conjugate gradient stabilized method (BiCG-Stab) method, where the performance of the PCG clearly outperforms that of the two other methods.

References

[1] R. A. Adams and J. J. F. Fournier. *Sobolev Spaces*. 2nd ed. Pure and Applied Mathematics. Amsterdam: Academic Press, Elsevier Ltd., 2003.

[2] M. Aldegunde, A. J. García-Loureiro, and K. Kalna. "3D Finite Element Monte Carlo Simulations of Multigate Nanoscale Transistors". In: *IEEE Transactions on Electron Devices* 60.5 (May 2013), pages 1561–1567.

[3] M. Aldegunde and K. Kalna. "Energy Conserving, Self-Force Free Monte Carlo Simulations of Semiconductor Devices on Unstructured Meshes". In: *Computer Physics Communications* 189 (Apr. 2015), pages 31–36.

[4] M. Aldegunde, N. Seoane, A. J. García-Loureiro, and K. Kalna. "Reduction of the Self-Forces in Monte Carlo Simulations of Semiconductor Devices on Unstructured Meshes". In: *Computer Physics Communications* 181.1 (2010), pages 24–34.

[5] D. Åsen. "Self-Force Reduced Finite Element Poisson Solvers for Monte Carlo Particle Transport Simulators". Master's Thesis. NTNU, 2016.

[6] I. Babuška and V. Nistor. "Boundary Value Problems in Spaces of Distributions on Smooth and Polygonal Domains". In: *Journal of Computational and Applied Mathematics* 218.1 (Aug. 15, 2008), pages 137–148.

[7] Z.-Z. Bai. "Motivations and Realizations of Krylov Subspace Methods for Large Sparse Linear Systems". In: *Journal of Computational and Applied Mathematics* 283 (Aug. 1, 2015), pages 71–78.

[8] L. Beilina, E. Karchevskii, and M. Karchevskii. *Numerical Linear Algebra: Theory and Applications*. Springer International Publishing: Imprint: Springer, 2017.

[9] H. Bohr. "L. Schwartz". In: S. M. Atiyah and D. Iagolnitzer. *World Scientific Series in 20th Century Mathematics*. 2nd edition. Volume 9. World Scientific, Nov. 2003, pages 19–45.

[10] H. Brezis. *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Springer Science & Business Media, Nov. 2, 2010.

[11] P. J. Brown and C. T. Faigle. *A Robust Efficient Algorithm for Point Location in Triangulations*. University of Cambridge, Computer Laboratory, 1997.

[12] G. Capodaglio and E. Aulisa. "A Particle Tracking Algorithm for Parallel Finite Element Applications". In: *Computers & Fluids* 159 (Dec. 15, 2017), pages 338–355.

[13] L. Chen, M. J. Holst, and J. Xu. “The Finite Element Approximation of the Nonlinear Poisson–Boltzmann Equation”. In: *SIAM Journal on Numerical Analysis* 45.6 (Jan. 2007), pages 2298–2320.

[14] X.-Q. Chen and J. Pereira. “A New Particle-Locating Method Accounting for Source Distribution and Particle-Field Interpolation for Hybrid Modeling of Strongly Coupled Two-Phase Flows in Arbitrary Coordinates”. In: *Numerical Heat Transfer, Part B: Fundamentals* 35.1 (1999), pages 41–63.

[15] R. Chordá, J. A. Blasco, and N. Fueyo. “An Efficient Particle-Locating Algorithm for Application in Arbitrary 2D and 3D Grids”. In: *International Journal of Multiphase Flow* 28.9 (2002), pages 1565–1580.

[16] L. Dagum and R. Menon. “OpenMP: An Industry Standard API for Shared-Memory Programming”. In: *IEEE Computational Science and Engineering* 5.1 (Jan. 1998), pages 46–55.

[17] T. Davis. *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms. Society for Industrial and Applied Mathematics, 2006.

[18] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. “Point Location”. In: *Computational Geometry*. Springer, Berlin, Heidelberg, 2008, pages 121–146.

[19] M. R. Ebert and M. Reissig. *Methods for Partial Differential Equations*. Cham: Springer International Publishing, 2018.

[20] M. A. Elmessary, D. Nagy, M. Aldegunde, J. Lindberg, W. G. Dettmer, D. Períc, A. J. García-Loureiro, and K. Kalna. “Anisotropic Quantum Corrections for 3-D Finite-Element Monte Carlo Simulations of Nanoscale Multigate Transistors”. In: *IEEE Transactions on Electron Devices* 63.3 (Mar. 2016), pages 933–939.

[21] A. F. Emery and W. W. Carson. “An Evaluation of the Use of the Finite-Element Method in the Computation of Temperature”. In: *Journal of Heat Transfer* 93.2 (May 1, 1971), pages 136–145.

[22] L. C. Evans. *Partial Differential Equations*. 2nd ed. Volume 19. Graduate studies in mathematics. Providence, R.I: American Mathematical Society, 2010.

[23] S. Fatnes. *Monte Carlo Particle Simulation in Unstructured Three-Dimensional Grids*. Project thesis. 2017.

[24] C. Geuzaine and J.-F. Remacle. “Gmsh: A 3-D Finite Element Mesh Generator with Built-in Pre- and Post-Processing Facilities”. In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pages 1309–1331.

[25] D. Goldar. “Calculation of Wavefunction Overlaps in First Principles Electronic Structure Codes”. Master’s Thesis. NTNU, 2017.

[26] T. González and D. Pardo. “Physical Models of Ohmic Contact for Monte Carlo Device Simulation”. In: *Solid-State Electronics* 39.4 (Apr. 1, 1996), pages 555–562.

[27] G. Green. *An Essay on the Application of Mathematical Analysis to the Theories of Electricity and Magnetism*. author, 1828.

[28] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, United States: MIT Press, 2015.

[29] L. Guibas and J. Stolfi. “Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi”. In: *ACM Trans. Graph.* 4.2 (Apr. 1985), pages 74–123.

[30] H. Hanche-Olsen. *Buckingham's Pi-Theorem, Lecture Note in TMA4195 Mathematical Modelling*. 2004.

[31] J. J. Harang. “Implementation of Maxwell Equation Solver in Full-Band Monte Carlo Transport Simulators”. Project Thesis. NTNU, 2015.

[32] G. W. Hart. *Multidimensional Analysis: Algebras and Systems for Science and Engineering*. Springer New York, 1995.

[33] A. Haselbacher, F. M. Najjar, and J. P. Ferry. “An Efficient and Robust Particle-Localization Algorithm for Unstructured Grids”. In: *Journal of Computational Physics* 225.2 (Aug. 10, 2007), pages 2198–2213.

[34] K. Hess. *Monte Carlo Device Simulation : Full Band and Beyond*. The Springer International Series in Engineering and Computer Science, VLSI, Computer Architecture and Digital Signal Processing. Springer US, 1991.

[35] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Bristol: Institute of Physics Publishing, 1988.

[36] V. Hoppe. “High Order Polynomial Elements with Isoparametric Mapping”. In: *International Journal for Numerical Methods in Engineering* 15.12 (Dec. 1, 1980), pages 1747–1769.

[37] C. Jacoboni and P. Lugli. *The Monte Carlo Method for Semiconductor Device Simulation*. Computational Microelectronics. Springer Vienna, 1989.

[38] C. Jacoboni and L. Reggiani. “The Monte Carlo Method for the Solution of Charge Transport in Semiconductors with Applications to Covalent Materials”. In: *Rev. Mod. Phys.* 55.3 (June 1983), pages 645–705.

[39] H. Jin, C. He, S. Chen, C. Wang, and J. Fan. “A Method of Tracing Particles in Irregular Unstructured Grid System”. In: *The Journal of Computational Multiphase Flows* 5.3 (2013), pages 231–237.

[40] J.-M. Jin. *The Finite Element Method in Electromagnetics*. John Wiley & Sons, Feb. 18, 2015.

[41] G. S. Ketefian, E. S. Gross, and G. S. Stelling. “Accurate and Consistent Particle Tracking on Unstructured Grids”. In: *International Journal for Numerical Methods in Fluids* 80.11 (Apr. 20, 2016), pages 648–665.

[42] H. R. Khan and D. Vasileska. “3d Monte Carlo Simulation of FinFET Using FMM Algorithm”. In: *Abstracts 10th International Workshop on Computational Electronics*. 10th International Workshop on Computational Electronics. Oct. 2004, pages 192–193.

[43] C. N. Kirkemo. “Monte Carlo Simulation of PN-Junctions”. Master’s Thesis. University of Oslo, 2011.

[44] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev. “Why the Monte Carlo Method Is so Important Today”. In: *Wiley Interdisciplinary Reviews: Computational Statistics* 6.6 (June 20, 2014), pages 386–392.

[45] S. B. Kuang, A. B. Yu, and Z. S. Zou. “A New Point-Locating Algorithm under Three-Dimensional Hybrid Meshes”. In: *International Journal of Multiphase Flow* 34.11 (Nov. 1, 2008), pages 1023–1030.

[46] S. E. Laux. “On Particle-Mesh Coupling in Monte Carlo Semiconductor Device Simulation”. In: *Simulation of Semiconductor Devices and Processes: Vol. 6*. Edited by H. Ryssel and P. Pichler. Vienna: Springer Vienna, 1995, pages 404–407.

[47] J. Lindberg, M. Aldegunde, D. Nagy, W. G. Dettmer, K. Kalna, A. J. García-Loureiro, and D. Perić. “Quantum Corrections Based on the 2-D Schr #x00F6;Dinger Equation for 3-D Finite Element Monte Carlo Simulations of Nanoscaled FinFETs”. In: *IEEE Transactions on Electron Devices* 61.2 (Feb. 2014), pages 423–429.

[48] G. B. Macpherson, N. Nordin, and H. G. Weller. “Particle Tracking in Unstructured, Arbitrary Polyhedral Meshes for Use in CFD and Molecular Dynamics”. In: *Communications in Numerical Methods in Engineering* 25.3 (2009), pages 263–273.

[49] C. Moglestue. *Monte Carlo Simulation of Semiconductor Devices*. London: Chapman & Hall, 1993.

[50] O. C. Norum. “Monte Carlo Simulation of Semiconductors - Program Structure and Physical Phenomena”. Master’s Thesis. NTNU, 2009.

[51] Ø. Olsen. “Construction of a Transport Kernel for an Ensemble Monte Carlo Simulator”. Master’s Thesis. NTNU, 2009.

[52] A. Quarteroni. *Numerical Approximation of Partial Differential Equations*. In collaboration with A. Valli. Volume 23. Springer series in computational mathematics. Berlin: Springer, 1994.

[53] A. Quarteroni. *Numerical Models for Differential Problems*. Second Ed. Volume 2. MS&A. Springer-Verlag Italia, 2009.

[54] K. Rektorys. *Variational Methods in Mathematics, Science and Engineering*. Springer Netherlands, 1977.

[55] W. Rudin. *Functional Analysis*. McGraw-Hill Series in Higher Mathematics. New York: McGraw-Hill, 1973.

[56] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, 2003.

[57] Y. Saad. *SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations*. May 21, 1990.

[58] M. Salazar-Palma, L.-E. García-Castillo, and T. K. Sarkar. "The Finite Element Method in Electromagnetics". In: *European Congress on Computational Methods in Applied Sciences and Engineering*. Sept. 2000.

[59] S. Salsa. *Partial Differential Equations in Action: From Modelling to Theory*. Second edition. Unitext volume 86. Springer International Publishing Switzerland, 2015.

[60] S. Salsa, F. Vegni, A. Zaretti, and P. Zunino. *A Primer on PDEs: Models, Methods, Simulations*. Volume UNITEXT – La Matematica per il 3+2. 65 volumes. UNITEXT. Milano: Springer-Verlag Italia, 2013.

[61] J. S. Savage and A. F. Peterson. "Higher-Order Vector Finite Elements for Tetrahedral Cells". In: *IEEE Transactions on Microwave Theory and Techniques* 44.6 (June 1996), pages 874–879.

[62] K. Schwarz and P. Blaha. "Solid State Calculations Using WIEN2k". In: *Computational Materials Science* 28.2 (2003), pages 259–273.

[63] A. Singh and R. Pal. "Infrared Avalanche Photodiode Detectors". In: *Defence Science Journal* 67.2 (Mar. 14, 2017), pages 159–168.

[64] A. Singh, V. Srivastav, and R. Pal. "HgCdTe Avalanche Photodiodes: A Review". In: *Optics & Laser Technology* 43.7 (Oct. 1, 2011), pages 1358–1370.

[65] C. M. Snowden. *Introduction to Semiconductor Device Modelling*. World Scientific Publishing Co. Pte. Ltd., 1986.

[66] A. K. Storebø, D. Goldar, and T. Brudevoll. "Simulation of Infrared Avalanche Photodiodes from First Principles". In: *Proceedings of SPIE, Vol. 10177* (2017).

[67] A. A. Ungar. *Barycentric Calculus In Euclidean And Hyperbolic Geometry: A Comparative Introduction*. Singapore: World Scientific, 2010.

[68] D. Vasileska, S. M. Goodnick, and G. Klimeck. *Computational Electronics; Semiclassical and Quantum Device Modeling and Simulation*. 1st edition. Boca Raton: CRC Press, Taylor & Francis Group, 2010.

[69] J. Yu. "Symmetric Gaussian Quadrature Formulae for Tetrahedronal Regions". In: *Computer Methods in Applied Mechanics and Engineering* 43.3 (May 1, 1984), pages 349–353.

[70] L. Zhang, H. Liu, and T. Cui. “A Set of Symmetric Quadrature Rules on Triangles and Tetrahedra”. In: *Journal of Computational Mathematics* 27.1 (Jan. 2009), pages 89–96.

[71] Q. Zhou and M. A. Leschziner. “An Improved Particle-Locating Algorithm for Eulerian-Lagrangian Computations of Two-Phase Flows in General Coordinates”. In: *International Journal of Multiphase Flow* 25.5 (Aug. 1, 1999), pages 813–825.

Appendices

A | Generating Triangulations with GMSH

In this Appendix, a short overview of simple mesh generation procedures for use with the meshing software GMSH [24] is given. A simple cube example is used to illustrate the minimal amount of commands that should be known when using GMSH. It should be noted that GMSH is much more powerful and provides a much wider range of possibilities for meshing than what is presented here. The structure of the output from GMSH is described, and some details on the triangulation class are provided.

A.1 Meshing Using the GMSH Software

The meshing software GMSH [24] can be used both with a graphical user interface (GUI) and directly from the terminal using the scripting language of GMSH to provide geometries to mesh. Using the scripting language provides more control over the geometry, and simplifies the process of altering already existing grids. The geometry can be defined in a text file with extension `.geo`, providing entities such as points, lines, surfaces and volumes. A simple geometry that can be used to make a volume mesh of a cube is

```
// Characteristic step length
c1 = 0.025;
// Lengths of the domain
lx = 1;
ly = 1;
lz = 1;
// Define the points which outlines the plate
Point(1) = {0, 0, 0, c1};
Point(2) = {lx,0, 0, c1};
Point(3) = {lx,ly,0, c1};
Point(4) = {0, ly,0, c1};
// Define boundary lines
Line(20) = {1,2};
Line(21) = {2,3};
Line(22) = {3,4};
Line(23) = {4,1};
// Mesh plane region
```

```
Line Loop(32) = {20,21,22,23};
Plane Surface(33) = {32};
// Extrude in z-direction and set physical tag
box[] = Extrude{0,0,lz}{Surface{33};};
Physical Volume(1) = {box[1]};
//Add tag to Dirichlet surface
Physical Surface(6) = {33, box[0],box[2],box[3],box[4],box[5]};
```

Lines starting with // are comment lines ignored by GMSH. As in any programming language, values can be assigned to variables, as is done here on the domain length specifications and the characteristic step length. Points are defined by setting 3D coordinates in the curly brackets, and an additional characteristic length is given to the point. This characteristic length is used to define the mesh size when the geometry is meshed. Different characteristic lengths can be given to each point. The index passed in the parenthesis of the `Point`-command is the name of the point, used for reference when constructing lines. Lines are defined between two points, and can be used to form line loops. Closed line loops can be used to define surfaces. To construct volumes, the easiest is to use the `Extrude`-command, extruding surfaces in the given coordinate direction given in its first curly bracket argument. The extruded object can be saved in an array type variable, here represented by `box[]`. This array then contains the opposite surface of that extruded in its zero-position, the volume on index 1 and the remaining surfaces in the following indices.

Adding physical tags is convenient in order to separate different parts of the mesh, for example, a Dirichlet surface. If physical tags are used, *only* the physical elements of the geometry are meshed, so it is important to add physical tags to all important parts of the geometry. Saving this in a file called e.g. `cube.geo`, with the GMSH software installed, the mesh is generated by the bash command

```
gmsh cube.geo -3
```

The output is a file `cube.msh`, containing information about node coordinates and elements. The option -3 specifies that the generated mesh should be a volume mesh in 3D. Replacing -3 with -2 will create a surface mesh only. Further options are

`-order <2>`	Element order. 2 gives quadratic elements, 3 cubic.
`-o <output>.msh`	Specify directory and name of output file.

An example of a quadratic mesh file-output from GMSH is given below.

```
$MeshFormat
2.2 0 8
$EndMeshFormat
$Nodes
1311
1 0 0 0
```

```
2 1 0 0
...
1310 0.08354166666670734 0.8367957170919162 0.9227289554953506
1311 0.06585188802083881 0.7638464417297364 0.9227289554953506
$EndNodes
$Elements
1060
1 9 2 6 33 125 10 11 142 15 143
2 9 2 6 33 141 9 10 144 14 145
...
1059 11 2 1 1 2 90 363 256 94 366 365 265 1173 273
1060 11 2 1 1 66 65 663 460 70 710 709 494 1175 493
$EndElements
```

The three numbers in the mesh format section are GMSH descriptors describing the version number, file type and data size, and are unrelevant for our purpose. The nodes section contains the complete list of nodes for the triangulation, with the first number indicating the node number and the three next its coordinates. The first number in the element section is the total number of listed elements. Further it lists the complete set of elements, as

```
<Number> <Type> <Number of tags> <Physical tag>
          <elementary geometrical entity>...<node-number-list>
```

In most cases, the number of tags are two, beeing only the two listed. The element types used in the MCFEM are

2	Linear triangles with three nodes
4	Linear tetrahedron with four nodes
9	Quadratic triangles with six nodes
11	Quadratic tetrahedron with ten nodes

As seen in the example mesh file given above, both surface triangles (type 9) and volume tetrahedra (type 11) are listed, with their respective list of nodes. When the mesh is imported to the triangulation class in `Fortran`, this information is used for construction, as described in the next section.

A.2 Triangulation Class

To simplify access to the elements and nodes during the simulation, a *triangulation class* was implemented during the specialization project [23]. The class now contains five member variables and six member-lists, in addition to member-functions for initialization, mesh reading and neighbor construction. The class interface is shown in Appendix B, under the module

`mod_triangulation`. The list of nodes is constructed be directly reading in the nodes list from the mesh file produced by GMSH. For the volume triangulation, only the elements corresponding to tetrahedra are saved in the triangulation's element list. Correspondingly, if the surface mesh (for the contacts) is constructed, only the triangles are saved in the element list. Thus, the number of elements in the triangulation will not correspond to the total number of elements given in the mesh file. Further, constructing the volume mesh, the physical tags on the surface elements are used to construct list of Dirichlet and Neumann nodes. If a surface element has a tag corresponding to one of the Dirichlet surfaces, each node in this element, which is not already saved to the list, is added to the list of corresponding Dirichlet nodes. It is possible to construct as many distinct Dirichlet surfaces as one desires, but the list of Neumann nodes is currently restricted to a single list, for simplicity.

After constructing the element list, the list of neighbors must be constructed. The mesh file does not contain information abour neighboring elements, and so a simple linear search algorithm is used to find common faces for elements. For the volume mesh, a face F of an element K is defined by three of the vertices of K, and the element sharing this face with K is defined as the Fth neighbour of K, $F = 1, .., d$. The neighbor-list is important for simple implementation of the point location algorithms (PLAs), see Section 4. The list requires computational time and memory, but is constructed only once during the simulations, and is used for all the particles in the grid at each time step of the simulation, saving computational time during simulations of long time intervals.

The CPU-times for computing the neighbor list is plotted in Figure A.1 for different number of elements in a 3D box with side lengths 1. The testing was done on an Intel Core i7-4770 processor with 4 CPUs, clock frequency 3.40 GHz, running Ubuntu 16.04 LTS. The CPU-times was recorded using the intrinsic function `CPU_TIME` in Fortran, and the program was compiled with the GNU Fortran compiler, version 5.4.1 with optimization flags `-O3`.

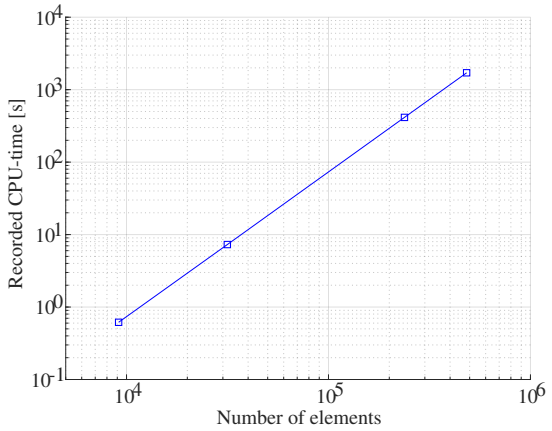


Figure A.1: CPU-times for the generation of the neighbor structure in the triangulation class. The plot is logarithmic, and shows linear performance with respect to the number of elements in the grid. The CPU-time recorded for generating the neighbor structure in a 3D grid with 483 051 elements was 28.5 minutes.

B | Overview of Source Code

The new finite element version of the Monte Carlo software package, MCFEM, is a restructured version of the previous code. On request, this appendix is provided as a minimal reference for future developers. It contains an overview of the modules that are currently included in the source code. Some of the old functionality is left in an archive and not included in MCFEM as of today. However, it should be an easy task to include this functionality by structuring it in a new module if found necessary. The reason it was left out was to ease compatibility with the new structure and minimize the number of pitfalls. Some revision, especially with respect to line length, has been done in all modules, to make it compile with *gfortran*. In the following, a short introduction to the compilation process is presented, before a complete list of source code files is given.

A Makefile (for use with GNU Make) has been added for simpler compilation and linking, and also contains a cleaning procedure for removing old build files. This eases the compatibility between different systems. The source code is tested on both *ubuntu 16.04 LTS* and *MacOSX*. The current compiler specified in the Makefile is *gfortran*, but this can easily be changed to e.g. *ifort*, as long as the compilation flags are changed accordingly. Necessary changes are explicitly stated with comments in the Makefile. There are currently three available options for compiling:

- `make` Compiles with debug flags. To be used under development.
- `make DEBUG=0` Compiles with optimization flags for optimal run time (`-O3`)
- `make DEBUG=2` Compiles with profiling flag `-pg`. The code will run slower, but produce an extra output file, `gmon.out`, which can be passed to the profiling tool *gprof*.

The dependence between modules is specified in the Makefile-included file `mksource.mk`, found in the source directory for the code, `src`. Thus, any change to a file which other files depend on, will rebuild the dependent files, but other files will not be rebuilt. For a full rebuild of all modules, write

- `make clean`

followed by a compilation option from the ones listed above. Running a clean and a rebuild once in a while is recommended to ensure that all changes are compatible among all modules.

All module names have been prefixed with `mod_`[1], for a clear distinction between modules and other functionality (types, subroutines, functions, variables, etc.). Types are prefixed with

[1] No rules without exception, the module `randomgen` is missing this prefix

`t_`. Abstract classes and derived types are not introduced, as this is stated as unwanted in the programming guide for the MCS. Each module is contained in its own source file with the same name as the module and the extension `.f90`. The main program is called MCFEM, and is contained in the source file `MCFEM.f90`. The following is a complete list of source code files for the program, where modules that have been added or seen extensive revision during the work with this thesis is listed in boldface. It should be noted that for each module, the list and description of its content are *not* complete, and any developer should consult the interactive HTML documentation generated by *FORD*, which is found in the program folder[2]. The list contains types with variables and procedure names, and important subroutines and functions with short explanations of their purpose, and always with the full list of input parameters. Modules with old content are included in the list, but not revised in the same manner as new functionality. The reader is referred to the old program manual for a more thorough explanation of these.

- **`MCFEM.f90`** Contains the main driver for the program, `PROGRAM MCFEM`. Before entering the time loop, the program calls all initialization routines for tabulating scattering rates, reads the geometry from file and constructs the triangulation, and loads or assembles the stiffness matrix for the Poisson solution. It then enters the time loop, where the electric field is calculated at specified time intervals, free flights of all particles are done and scattering routines are called. Then neutrality is assured in the contact regions, and the loop continues until the user-specified number of iterations are terminated. The most important simulation variables which are declared and initiated in the main program are

`TYPE(t_APD_device) :: APD`	Device specification, with a member function to initialize particles in correct positions
`TYPE(t_contact) :: cntRP`	Contact specifications, right P-contact
`TYPE(t_contact) :: cntLP`	Contact specifications, left P-contact
`TYPE(t_contact) :: cntN`	Contact specifications, center N-contact
`TYPE(t_base) :: base`	Basis functions and quadrature rules for FEM
`TYPE(t_triangulation) :: tri`	Triangulation of the domain
`TYPE(t_spCRS) :: stiff`	The stiffness matrix for FEM calculations
`TYPE(t_spCRS) :: stiffBC`	The stiffness matrix altered with an identity block for Dirichlet nodes
`TYPE(t_spCRS) :: LU`	The LU decomposition of the stiffness matrix
`CHARACTER(255) :: stiff_file`	Path to stiffness matrix file
`CHARACTER(255) :: stiffBC_file`	Path to stiffness BC matrix file
`INTEGER :: diagptr(:)`	A list of diagonal entries to seperate
`INTEGER :: e_elem(:), h_elem(:)`	the lower triangular matrix L from the upper triangular U
`INTEGER :: illegalParticle(:)`	An array of the same length as the number of

[2] Access to the program is given by FFI

	electrons, everywhere zero except at positions corresponding to particles that should be removed because they have left through the contacts.
`INTEGER :: hillegalParticle(:)`	Same for holes
`INTEGER :: timestep`	Loop counter for the time loop simulation
`REAL :: eload(:)`	Contribution from electrons to load vector
`REAL :: hload(:)`	Contributions from holes to load vector
`REAL :: background_load(:)`	The contribution from the background charge to the load vector
`REAL :: load(:)`	Complete load vector (sum of the previous)
`REAL :: loadBC(:)`	Load vector with zeros in Dirichlet nodes
`REAL :: g(:)`	The lifting vector for the Dirichlet boubdart
`REAL :: Ag(:)`	The matrix-vector product of the stiffness matrix and the lifting vector
`REAL :: pot(:)`	Potential, the solution vector of the Poisson linear system, unit: Volt
`REAL :: pot0(:)`	Initial guess for PCG, the potential without added boundary conditions.
`REAL :: epos(:,:)`	Scaled electron postions (dimensionless), `epos(p,:)` is the three coordinates of electron p
`REAL :: hpos(:,:)`	Scaled hole positions (dimensionless)
`REAL :: kvec(:,:)`	Wave vectors on matrix form for electrons
`REAL :: hkvec(:,:)`	Wave vectors on matrix form for holes
`REAL :: E(:,:)`	Electric field, `E(p,:)` gives the three components of the electric field for the pth electron
`REAL :: hE(:,:)`	Electric field for holes

- **mod_base.f90** Provides basis functions, their derivatives and quadrature rules for the 3D reference tetrahedra, with the following public content:

  - `TYPE :: t_base`, with variables

`INTEGER:: num_func`	number of basis functions
`INTEGER:: num_quad`	number of quadrature nodes
`REAL:: x_quad(:,:)`	list of quadrature points
`REAL:: w_quad(:)`	list of quadrature weights
`REAL:: phi(:,:)`	Basis functions evaluated at quadrature points
`REAL:: dphi(:,:,:)`	Gradient of basis functions evaluated at quadrature points

  - `SUBROUTINE :: new_3d_base(base, degree)` initializes a new object *base* with the given *degree*. Current possible degerees is 1 (linear) or 2 (quadratic)
  - `FUNCTION eval_3d_phi(nf,xi) RESULT(phi)` Evaluates the basis functions. The integer *nf* specifies number of functions, either 4 (linear) og 10 (quadratic), and *xi* is the evaluation points on the reference tetrahedra.
  - `FUNCTION eval_3d_dphi(nf,xi) RESULT(dphi)` Evaluates the gradient of the basis functions, works as above.

- `mod_carrierBCs.f90` Contains the procedure for deleting particles leaving through contacts

  - `SUBROUTINE DeleteParticl(ptype, illegalPart, nump, EjectNumb)` based on the routine by Vasileska et al. [68].

`INTEGER:: ptype`	Type of particle, -1 for electrons, 1 for holes
`INTEGER:: illegalPart(:)`	Logical vector of zeros and ones describing particles to be deleted
`INTEGER:: nump`	Number of active particles, `esize` if $ptype = -1$, `hsize` if $ptype = 1$
`INTEGER:: EjectNumb`	Number of ejected particles

- `mod_carrierDyns.f90` contains the flight routine

  - `SUBROUTINE flight(dev, kx, ky, kz, x, y, z, band, F, timestep, poisson, Eject)`

`CLASS(t_APD_device):: dev`	the device (e.g. the `APD` from the main program)
`REAL :: kx,ky,kz,x,y,z`	wave vector and position of particle
`REAL :: timestep`	basic timestep
`REAL :: F(3)`	The force (the electric field) acting on the particle
`CHARACTER(LEN=*):: band`	The band for electrons and the valley for holes.
`INTEGER:: poisson`	Logical variable, 1 if poisson solver is on, 0 otherwise
`INTEGER :: Eject`	A return variable, set to 1 if the particle needs to be deleted from the simulation.

- `mod_collectstats.f90` Module with functionality for collecting statistics from the run and saving to files. Contains the following public subroutines:

  - `SUBROUTINE openStatFiles` Opens all the files that the statistics should be written to

- SUBROUTINE closeStatFiles Closes the same files when the run is finsihed
- SUBROUTINE write12(step, extractor) Writes particle positions and wave vectors to file 12 times during the simulation
- SUBROUTINE writeelectronenergy Writes average electron enegries to file

- **mod_contact.f90** Procedures for handling insertion of particles at the contact. See also Section 5.
  - TYPE :: t_contact, with variables and procedures

CLASS(t_triangulation) :: surf	Surface triangulation of contact area
CHAR(LEN=1) :: ctype	Contact type (P or N)
REAL :: impdens	Impurity (doping) density of area adjacent to contact
REAL :: area	Total surface area of contact
REAL :: xstart, dx	Variables defining neutral area of contact
REAL :: ystart, dy	
REAL :: zstart, dz	
PROCEDURE :: init_contact	Initialization of contact variables from input file

    SUBROUTINE init_contact(contact, cname)

CLASS(t_contact) :: contact	The contact variable that is to be initialized
CHARACTER(LEN=*) :: cname	The name of the contact as provided in the input file.

PROCEDURE :: inNeutral	Test if particles are in neutral area
PROCEDURE :: insertParticles	Compute necessary number of new particles and insert new particles at contact surface

- **mod_device.f90** Some device specification and initialization of particles in the given device. Goal is that more device types can be added without needing to change other parts of the code.
  - TYPE :: t_APD_device, with device specifications(not listed) and procedures:

PROCEDURE :: init_APD	Specification of APD variables (length, width, impurity, etc.)
PROCEDURE :: init_particles	Initialize particles according to the different doping densities in each region, corresponing to an overall neutral devices

`PROCEDURE :: passedContact`	Calculates if a particle has passed a contact of the device

- `mod_energyflow.f90` Contains phononvariables and the following routine:
  - `SUBROUTINE writeanderase` Writes and resets phonon variables
- **`mod_FEMassembly.f90`** Contains procedures to assemble the stiffness matrix and the load vector for the linear system arising from the FEM discretization. The public routines are
  - `FUNCTION assemble_stiff(tri, base) RESULT (A)` Assembles the stiffness matrix for the given triangulation *tri* with basis functions and quadrature rule from *base*. The result A is in the CRS format (see `mod_sparsemat.f90`).
  - `FUNCTION stiffBC_fromstiff(A, tri) RESULT(A_BC)` Sets the Diriblet part of A to identity, resulting in a new matrix *A_BC* ("A with boundary conditions")
  - `FUNCTION assemble_load(tri,base,ppos,nump,c, p_elem) RESULT(f)` Assembles the load vector for the given triangulation *tri* with basis functions and quadrature rule from *base*, with the particle positions given in the $3 \times N_p$ vector, with c a constant depending on the permittivity and the particle charge, and *p_elem* the list of element positions for each particle.
  - `FUNCTION loadBC_fromload(tri, f) RESULT(f_BC)` Sets 0 Diriclet contribution to entries of f corresponing to Dirichlet nodes, resulting in a new vector *f_BC* ("f with boundary conditions")
- **`mod_FEMroutines.f90`** Additional FEM-routines;
  - `SUBROUTINE init_device_geometry(tri)` A simple routine reading the file name specification and element order from the initialization file and calls the routine initialicing the triangulation (see `mod_triangulation.f90`).
  - `FUNCTION Efield_fromPot(pot,pos,nump, tri, base, p_elem) RESULT (E)`
    Calculates the Electric field in each particle position, according to Equation (2.48). *pot* is the potential in each node as calculated by solving the linear system arising from the FEM-discretization, *pos* are the particle positions, *nump* the number of active particles, *tri*, *base*, *p_elem* as for the stiffness assemble routine.
- `mod_globalparams.f90` Holds global parameters such as physical constants and some user-specified parameters such as the number of simulation steps and basic timestep. A routine,
  - `SUBROUTINE init_globalparams` reads basic user specified parameters from the initialization file.

- mod_ImpactIonizationScattering.f90 Contains routines for performing impact ionization scattering of particles.
  - SUBROUTINE ImpactIonizationGen(x0,y0,z0,totE)
  - FUNCTION DistEnergyLoss(x,E)
  - FUNCTION EnergyLoss(idum, E)
- mod_initialize.f90 Initialization routines for flight duration (called *flightlength* in the program) and wave vectors, in addition to a routine initializing from file written in possible previous run. Other initialization routines, unused in MCFEM, is also contained in this module (not listed here, but available in the HTML documentation).
  - SUBROUTINE readinitialfromfile()
  - SUBROUTINE initializeflightlength()
  - SUBROUTINE initializekvector
- **mod_isovaryingstring.f90**[3] Contains functionality for variable string lenght, used only in mod_readintools.f90
- **mod_linalg.f90** Linear Algebra procedures:
  - FUNCTION cross(a, b) Cross product of vectors a and b (Assumed each vector has three entries)
  - FUNCTION dot(a,b) Dot product of vectors a and b (any length)
  - FUNCTION matmatT_3x3(inmat) RESULT(outmat) Matrix-matrix product of the input matrix *inmat* with itself transposed (only for 3×3 matrices).
  - FUNCTION detmat_3x3(mat) RESULT(det) Determinant of 3×3 matrix
  - FUNCTION inv_3x3(mat) RESULT(inv) Inverse of 3×3 matrix
  - FUNCTION matvecmult(mat, vec) RESULT(resvec) Matrix-vector multiplication, for non-sparse matrices, any size. This routine is an interface operator overwriting * for matrix vector input. Throws error if dimensions of matrix and vectors do not agree.
- mod_matpar.f90 Material parameters.
- mod_measurements.f90 Calculation of group velocities for particles
  - FUNCTION holevgi(dir, kx, ky, kz, band)
  - FUNCTION electronvgi(ki, k, valley)

[3]GNU Public Licsence File

- `mod_nonpcc.f90` Old module missing decent documentation. It is referenced in some case selects in the scattering modules, and therefore included, but non of its routines are explicitly called during the execution of MCFEM.

- `mod_pauli.f90` Parameters and routines for use of the Pauli principle. Requieres extensive amount of computational resources and is turned off during the execution of MCFEM, but is, as the nonpcc-module referenced elsewhere in the code and thus included.

- `mod_phononstatistics.f90` Contains routines for handling phonons (not listed, see HTML documentation for complete list).

- **`mod_PL2D_CBF.f90`** Point location routine and help functions for 2D-point location in triangulations.

  - `SUBROUTINE point_location_2D(loc,triangulation, point, init_search_elem)` As described in Algortihm 4.1 in Section 4.2.
  - `FUNCTION center_of_elem(triangulation, elem) RESULT(center)` Calculates center of the triangular element *elem* in *triangulation*.
  - `FUNCTION trajectory_to_the_left(init_point,traj, node) RESULT(L)` Calculates the TTL, i.e. the value of α_i from Equation (4.3).
  - `FUNCTION particle_to_the_left(node1,node2, point) RESULT(P2L)` Calculates the PTL, i.e. the value of β_i from Equation (4.4).

- **`mod_PL3D_CBF.f90`** Point location routine and help functions for 3D-point location in triangulations.

  - `SUBROUTINE point_location_3D(loc,triangulation, point, init_search_elem)` As described in Algortihm 4.2 in Section 4.2.
  - `FUNCTION center_of_elem(triangulation, elem) RESULT(center)` Calculates center of the tetrahedral element *elem* in *triangulation*.
  - `FUNCTION trajectory_towards_inside(init_point,traj, node1, node2) RESULT(TTI)` Calculates the FTI, i.e. the value of ω from Equation (4.6). The *traj* variable is the trajectory denoted by $\boldsymbol{S}$ in Section 4
  - `FUNCTION particle_inside(node1,node2,node3, point) RESULT(P2I)` Calculates the PTI, for a face given by the three nodes, i.e. the value of γ_F from Equation (4.9).

- **`mod_precond.f90`** Routines for handling preconditioners for sparse linear system solvers. Currently only the *ILU0* is implemented, but it should provide a descent interface for introducing new preconditioners.

- SUBROUTINE ILU0(A, LU, diagptr) Computes the LU decomposition of A and a *diagptr* pointing to the diagonal entries in the LU matrix, which stores L in its lower triangular part and U in its upper triangular part. Note: this is not the product LU but merely the two matrices saved in one to save space.
- FUNCTION solveLUz(LU, diagptr, b) RESULT(z) Solves first $Ly = b$ and then $Uz = y$ and returns z.

- mod_precpar.f90 Contains the precision parameter for reals, wp (working precision).

- **mod_readintools.f90**[3] Routines for reading from initialization file, such as GETSTR, GETINT, etc.

- **mod_refmaps.f90** Mapings between reference tetrahedron and elements and Jacobians of these.
  - FUNCTION cartesian_to_barycentric(triangulation, elem, point) RESULT(lambda)
  - FUNCTION barycentric_to_cartesian(triangulation, elem, lambda) RESULT(point))
  - FUNCTION J_ref_to_elem(triangulation, elem) RESULT(J))
  - FUNCTION J_elem_to_ref(triangulation, elem) RESULT(J)

- mod_scatter.f90 Scatter routines
  - SUBROUTINE scatter(j,o,SumRates,energydelta, ScCounter, hScCounter) Scatter routine for electrons
  - SUBROUTINE hscatter(j,o,hSumRates,energydelta, hScCounter, ScCounter) Scatter routine for holes
  - FUNCTION thetaintrabandpolaroptical(hk,hkf,betasqrd)
  - FUNCTION thetainterbandpolaroptical(hk,hkf,betasqrd)
  - FUNCTION holeenergy(hk,band)

- mod_scatteringangle.f90 Contains routine for final scattering angle after scattering.

- FUNCTION thetapolaroptical(E,Ef,alpha,betasqr,meff)

- mod_scatteringrates.f90 All taburation of scattering rates is performed in this module, with a long list of public variables and calculation routines. See HTML documentation and source code.

- **mod_sparselinalg.f90** Routines for linear algebra with sparse matrices.

- `FUNCTION matvecmult(A,v) RESULT (Av)` Matrix vector multiplication when *A* is on the sparse format CRS (see `mod_sparsemat.f90`). Overwrites the multiplication operator * for input of this type.
- `FUNCTION solveBiCGStab(A, b, x0, tol) RESULT(x)`
  A BiCG-Stab algorithm
- `FUNCTION solvePCG(A,b,x0,LU,diagptr, tol) RESULT(x)`
  Preconditoned Conjugate gradient with LU preconditioner
- `FUNCTION solveCG(A,b,x0, tol) RESULT(x)` Conjugate Gradient algorithm

- **mod_sparsemat.f90** Sparse matrix types
  - `TYPE t_spDOK` Dictionary of Keys matrix type

`INTEGER ::`	`maxnnz`	The maximum value of nonzero elements
`INTEGER ::`	`nnz`	Actual number of nonzero elements
`INTEGER ::`	`nrows`	Number of rows in the dense format
`INTEGER ::`	`ncols`	Number of cols in the dense format
`INTEGER ::`	`keys(:,:)`	Pair of (row, col) for nonzero matrix element
`REAL ::`	`vals(:)`	The nonzero values of the matrix

    `PROCEDURE :: spalloc => spalloc_DOK` Sparse allocation
    `PROCEDURE :: set => set_DOK` Set value
    `PROCEDURE :: get => get_DOK` Get value
  - `TYPE t_spCRS` Compressed Row Storage matrix type

`INTEGER ::`	`nrows`	Number of rows in the dense format
`INTEGER ::`	`ncols`	Number of cols in the dense format
`INTEGER ::`	`cols(:)`	Column index for the corresponding value in *vals*
`INTEGER ::`	`rowstart(:)`	Start index for each row in vals
`REAL ::`	`vals(:)`	The nonzero values of the matrix

    `PROCEDURE :: fromDOK => fromDOK_CRS` Convert from DOK to CRS
    `PROCEDURE :: get => get_CRS` Get value
    `PROCEDURE :: replace => replace_CRS` Replace value
    `PROCEDURE :: toFile => toFile_CRS` Write CRS matrix to file
    `PROCEDURE :: fromFile => fromFile_CRS` Read CRS matrix from file
  - `SUBROUTINE CRS_assign(A, B)` Overwrites assignment operator (= for CRS matrices)

- **mod_triangulation.f90** The triangulation module. Important for every aspect of the FEM part of the code. Contains the triangulation type and its procedures.
  - `TYPE :: t_triangulation`

`INTEGER :: d`	Dimension (2 or 3)
`INTEGER :: num_nodes`	Number of nodes
`INTEGER :: num_elems`	Number of elements
`REAL :: nodes(:,:)`	List of nodes (range; (1:Nh,1:3))
`INTEGER :: elems(:,:)`	List of elements
`INTEGER :: neighbours(:,:)`	List of neighbors
`INTEGER :: DirBound(:,:)`	(:,i) the ith list of DirNodes
`INTEGER :: NeuBound(:)`	The list of Neumann nodes
`INTEGER :: numDir(:)`	! (i) Number of DirNodes for ith list
`INTEGER :: numNeu`	Total number of neumann nodes
`PROCEDURE :: init_triangulation`	Initializer for the triangulation. Use this to initialize the triangulation for a given dimension and a *.msh file. The triangulation must be allocated before this initializer is called.
`PROCEDURE :: read_from_msh_2D`	Reads the *.msh file to make the surface triangulation for the contacts
`PROCEDURE :: read_from_msh_3D`	Reads the *.msh file to make the volume triangulation
`PROCEDURE :: read_from_msh_3D_quad`	As above, for quadratic element
`PROCEDURE :: find_neighbours_2D`	Make the neighbours list from the information saved in the elems vector in the 2D case
`PROCEDURE :: find_neighbours_3D`	Make the neighbours list form the information saved in the elems vector in the 3D case
`PROCEDURE :: read_boundary_quad`	Find boundary nodes given quadratic elements
`PROCEDURE :: read_boundary_lin`	Find boundary nodes given linear elements

- `randomgen.f90` Contains the pseudorandom number generator:
  - `FUNCTION rangen(idum)`