NTNU
Norwegian University of
Science and Technology

Self-Force Reduced Finite Element Poisson Solvers for Monte Carlo Particle Transport Simulators

David Kristian Åsen

Master of Science in Physics and Mathematics
Submission date: July 2016
Supervisor: Helge Holden, MATH

Norwegian University of Science and Technology
Department of Mathematical Sciences

Self-Force Reduced Finite Element Poisson Solvers for Monte Carlo Particle Transport Simulators

David Kristian Midtgård Åsen

Abstract

The purpose of this master's thesis is to implement a Finite Element Poisson solver for a Monte Carlo particle transport simulator without the influence of self-forces. Self-forces are unphysical forces that come into existence when symmetries are not taken into account in a particle-mesh coupling. These self-forces are handled using Kalna's self-force reduction method. A central subject for particle transport simulators are that they need a particle-mesh coupling scheme, this is done using a Dirac delta function in the nearest-element center scheme. The particles' locations in the triangulation are needed for the particle-mesh coupling, and they are found using Guiba's and Stolfi's Point location algorithm. A linear basis is used for the Finite Element method, and the contacts are dealt with using conditions of thermal equilibrium and charge neutrality. To solve the linear system in the discretization, preconditioned Conjugate Gradient method is implemented together with the preconditioners ILUT and ILU0. The usage of these methods and their parameters are discussed, and simulations of a PN-junction is performed to verify that the implementation is working. Improvements to the self-force reduction and the particle location algorithm are made, and a relationship between the background charge, the particle charge, and the element size is found. This relationship is used for the characteristic step length to generate a near-optimal mesh. It is found that the implementation is successful and can be used in the Monte Carlo particle transport simulator to simulate semiconductor devices.

Sammendrag

Hensikten med denne masteroppgaven er å implementere en Endelige Element Poisson løser for Monte Carlo partikkeltransportsimulatorer uten innflytelse av egenkrefter. Egenkrefter er fiktive krefter som følger ved bruk av partikkel-element kobling uten hensyn til symmetrier. Egenkrefter blir fjernet ved å bruke Kalnas egenkrefter-reduksjon. Et sentral tema for partikkeltransport er at de trenger en kobling mellom partikler og noder. Dette er gjort ved å bruke Dirac delta funksjonen i nærmeste element senter. Partiklenes posisjon i trianguleringen er funnet ved bruk av Guibas og Stolfis punktlokaliseringsalgoritme. En lineær basis er brukt i Endelig element metode og kontaktene er implementert ved å bruke termisk likevekt og ladningsnøytralitet som betingelser. For å løse det lineære likningssystemet i Endelig element metode brukes forbehandlet konjugerte gradienters metode sammen med forbehandlerene ILUT og ILU0. Bruken av disse metodene og deres parametere blir diskutert og

simuleringer av en PN-overgang brukes til å verifisere at implementasjonen fungerer. Det blir laget forbedringer til egenkrefter-reduksjonen og punktlokaliseringsalgoritmen, og en sammenheng mellom dopeladning, partikkelladning, og elementstørrelser blir funnet. Sammenhengen blir brukt til å lage den karakteristiske steglengden for å generere en god triangulering. Implementasjonen er god og kan brukes i Monte Carlo partikkeltransportsimulatorer til å simulere halv-ledere.

Contents

List of Figures

Acronyms

CIC Cloud-in-cell method.

CRS Compressed row storage format.

FEM Finite element method.

GMSH Gnu mesh software.

IChol Incomplete Cholesky preconditioner.

ILU0 Zero fill-in Incomplete lower upper preconditioner.

ILUT Incomplete lower upper threshold preconditioner.

MCCT Monte Carlo charge transport software.

NEC Nearest-element-center method.

NGP Nearest-grid-point method.

PCG Preconditioned conjugate gradient method.

PDE Partial differential equation.

PIC Particle-in-cell method.

PLA Point location algorithm.

PMC Particle-Mesh coupling scheme.

List of Symbols

Central Symbols

$\mathring{\phi}$ Transformed potential

ϕ Potential

$\phi_j^{p_{node},R}$ Mesh coupled reference potential at node j with a particle set in node p_{node}

ϕ^{ref} Reference potential

$\vec{E}$ Electric field

h Characteristic step length, can vary in the domain

l' Mesh coupling scheme

l_i Charge assignment scheme

q_p Particle's associated charge

z_i Electric field assignment scheme

Sets

$\mathbf{H^1(\Omega)}$ First order Sobolev space.

$\mathbf{N}$ Nodes in mesh

$\mathbf{N_D}$ Set of nodes on Dirichlet boundary

$\mathbf{N_I}$ Set of nodes not on the Dirichlet boundary

$\mathbf{P}$ Set of particles in MCCT simulation.

γ The enumeration of the sets Ω_j,Ch. 5: set of edges

$\mathbb{N}_{nodes}$ Set of nodes from GMSH.

Ω Domain

Ω_j Subdomain of Ω with varying doping densities.

τ Triangulation, Ch.6: Tolerance of PCG

τ_i Subset of triangulation with node i as a vertex

P_i Subset of particles in support of b_i

P_K Subset of particles contained in element K

Object identifiers

i_K Local enumeration of nodes in element K

K An element of the triangulation, $K \in \tau$

p Particle, $p \in P$

p_{node} Node with fictive particle, $p_{node} \in N_I$

Matrices

L Lower unit matrix

M Solution matrix

U Upper matrix

Z Preconditioner matrix

Vectors

$\vec{e_p}$ Edge origin to particle vector

$\vec{e}$ Directed edge vector

$\vec{p_i}$ Vertex positions of local element, Ch. 6: Conjugate vectors

$\vec{r_p}$ Particle position

Scalars

α Scaling factor of characteristic step length refinement equation.

ϵ Permittivity

λ_D The Debye length

ω_p Plasma frequency

ρ Charge density

ρ_{bc} Background charge density

$\rho_{particles}$ Particle charge density

τ_t Threshold parameter of ILUT

A Area, Ch. 6: Symmetric positive definite matrix.

A_i Area of region i

A_K Area of element K

b_i	Basis function
D	Dirichlet boundary value
e	The elementary charge, Ch. 5 edge number
k_B	Boltzmann constant
l	Distance
m	Mass
N	Neumann boundary value
$n(P)$	The cardinality of set P, the number of particles.
n_D	Doping density
n_D^j	Doping density, in varying regions j
q	The signed elementary charge
T	Temperature
t	Time
v	Velocity

Chapter 1

Introduction

Society today is marked by the use of electronic devices which consist of semiconductor devices. Semiconductor devices are hard to describe properly using analytical tools, which means other approaches are necessary. Nowadays, computers allow the modeling of processes inside materials. This can reduce cost immensely, as fewer prototype semiconductors need to be produced. Hence there is a demand of modeling non-equilibrium carrier distributions in semiconductor devices.

There are several models for semiconductor devices. Each model is employed depending on the scale of the device. Hydrodynamic models work well until scales are so small that electrons are too far from equilibrium. Smaller devices need to factor in quantum mechanical effects. The de facto standard is to use semiclassical Monte Carlo methods down to scales where the quantum effects are too noticeable. When the scales are really small, that is, when the characteristic dimensions become shorter than the electron phase coherence length, the non-equilibrium Green's function method is applied. The devices treated in this paper fall within the domain of the semiclassical Monte Carlo method, which works well with dimensions in between the other two methods. The timescale of the method is typically within some hundred picoseconds.

The ensemble Monte Carlo method for particle transport follows the phase space trajectories of free electrons and holes in a semiconductor device. The behavior of particles is modeled as a series of free flights and scattering events. The method is called semiclassical because it utilizes classical theory for the free flights and quantum theory for the scattering events. The scattering events are dealt with by usage of the Monte Carlo method. The duration of the free flight is dependent on the scattering rates. Hence they are both determined drawing random numbers.

The method studied in this thesis is called the self-consistent ensemble Monte Carlo method for particle transport. Building on the ensemble Monte Carlo method, the particles' contribution to the potential is taken into account with the solution of Poisson's equation. The solution to Poisson's equation can be numerically approximated in numerous ways, for instance with the finite difference method, the multigrid method, or the fast multipole method, each with their advantages and

disadvantages. One of the latest concerns is to make solvers capable of difficult geometries and extending these to three-dimensional models. The finite element method(FEM) is chosen as the numerical method in this thesis because it can address these concerns.

In the 1950s, M. J. Turner was among the first to use the finite element method in everyday practice for the airplane industry [1, 2]. It was further developed over the following decades. The variational approach was studied in the 60s, and the method was expanding into the area of civil engineering. Before the end of the 80s, the method was developed into a rigorous mathematical frame set. Today, the method divides the domain into subdomains, then in each subdomain, the partial differential equation is posed as a variational problem. A basis in a subspace of the space of functions is made and transformed into a linear system to be solved. For usage in the particle-based Monte Carlo method, there are some important aspects as to how the finite element method is implemented. The charge distribution of the particles and the resulting potential needs to be addressed in a way that gives physical meaning to the simulation.

Chapter 2

Problem definition

The application for semi-classical transport modeling of carriers in electronic devices which is studied, and will be further developed, is named "Monte Carlo Software for Charge Transport and Electro-optic Applications". This software will be referred to as the Monte Carlo charge transport software (MCCT). This chapter presents MCCT in its previous states. In Section 2.1 the program flow will be introduced. In Section 2.2 the steps of a Particle-Mesh coupling are presented. In Section 2.3 details of relevant implementations are discussed. In Section 2.4 the previous particle boundary conditions of the device are presented. In Section 2.5 the relation between chapters in this master thesis, and the program flow is shown.

2.1 Overview

Holes and electrons are modeled as particles termed superparticles. Superparticles represent a number of holes or electrons through a superparticle charge q_p and are initialized with their respective positions and velocities. Each superparticle has an associated charge density weight calculated from the particles they represent [3]. This weight is given by the doping density in a volume of donors, with a user-chosen number of superparticles. The holes get assigned the same charge density as the electrons to ensure that no charge is left over after a recombination of electrons and holes. A flow chart of a typical MCCT simulation is shown in Figure 2.1. The free flights, acceleration, and displacements are calculated using classical mechanics. The scattering events and final states are calculated using quantum mechanics[4]. The simulation controls everything without limiting the charge development of superparticle controlled areas.

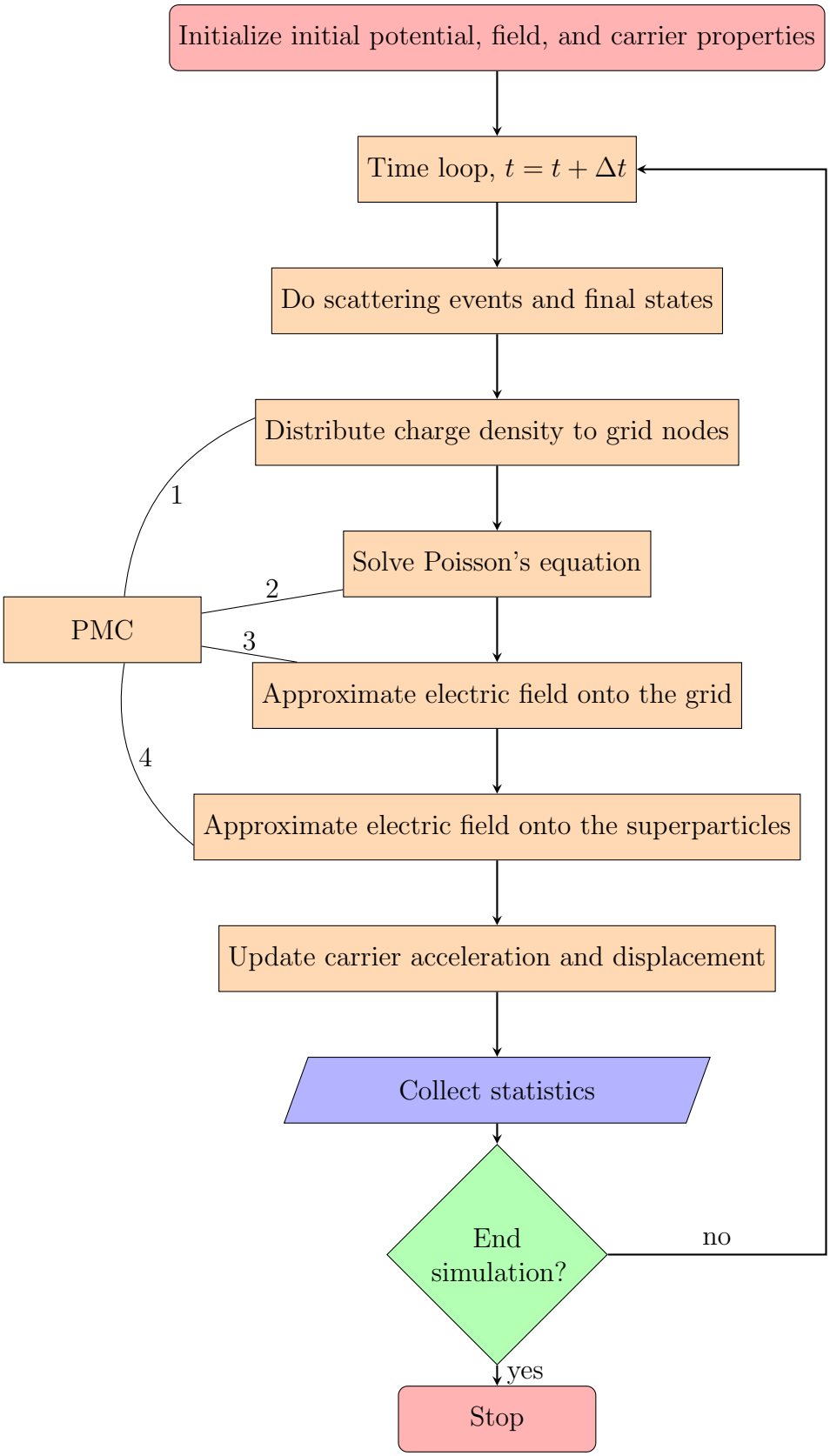


Figure 2.1: Flow chart of an MCCT run.

2.2 Particle-Mesh coupling schemes

The problem in its core can be split into what is called the Particle-Mesh coupling schemes (PMC) [5, 6], which consist of the following steps:

1. Assign particle charges to the mesh.
2. Solve Poisson's equation.
3. Calculate the electric field in mesh nodes.
4. Interpolate electric field onto particles.

Since changing one of these steps may alter what is appropriate for the others, the steps must be considered in relation to each other. The purpose of these schemes is to maximize the realism of the simulation and minimize the impact of numerical artifacts. The numerical artifacts reduced in PMC schemes takes the form of artificial forces exerted on superparticles. These are often referred to as "self-forces" [5].

2.3 Previous implementations

This section summarizes the relevant implementations in MCCT by looking at the grid requirements and each step in PMC.

2.3.1 Grid

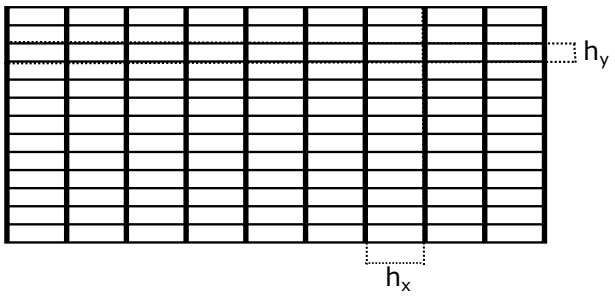


Figure 2.2: An illustrative mesh with constant step sizes.

As illustrated in Figure 2.2, the grid is structured with fixed step sizes h_x and h_y in $x-$ and $y-$ directions. The properties of the mesh are important for making sure that all relevant interactions are conserved, and therefore, there are limitations on the coarseness of the grid. The spatial limitation needs to be small enough to capture the charge variations in the grid. It has to be smaller than the smallest

wavelengths of charge variations. This smallest wavelength is given by the Debye length λ_D [7]. The Debye length is

$$\lambda_D = \sqrt{\frac{\epsilon k_B T}{e^2 n_D}}, \tag{2.1}$$

where ϵ is the permittivity, k_B is Boltzmann constant, T is the temperature, e is the elementary charge, and n_D is the doping density.

Not so important for PMC, but essential to MCCT, is the time steps for a simulation. This needs to be below the inverse plasma frequency. The plasma frequency is

$$\omega_p = \sqrt{\frac{e^2 n_D}{\epsilon m^*}}, \tag{2.2}$$

where m^* is the effective mass of an electron. The effective masses vary depending on the chosen model, for MCCT the effective masses can be found in the appendix of [3].

In addition to these criteria, the maximum distance traveled by a particle needs to be restricted[8]. This distance is

$$l_{max} = v_{max} \cdot \Delta t, \tag{2.3}$$

where v_{max} is the estimated maximum carrier velocity and Δt is the time step. The time step must be chosen so that l_{max} is smaller than the spatial mesh size. This is to ensure that the field distribution, only calculated once every Δt, can keep up with the particles.

2.3.2 Charge distribution

These algorithms are written by C.N. Kirkemo, and are described in [3, 5]. They are presented here as they introduce well the core concepts needed throughout the report.

Nearest grid point method

The simplest charge distribution algorithm, illustrated in Figure 2.3, is the nearest-grid-point(NGP) method. In the structured grid, this distribution is

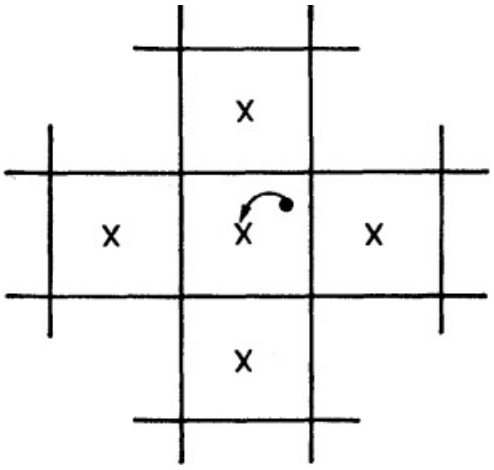


Figure 2.3: Illustration in two dimensions of NGP taken from [9]. The crosses are nodes in the grid. The particle charge is given to the closest node. The dot is the superparticle.

$$\rho_{i,j} = \rho(x_i, y_i) = \frac{en_D A_{device}}{n(P) A_{element}} = \rho_{sup}, \tag{2.4}$$

where $A_{device}, A_{element}, n_e, n(P), e$ denotes respectively the area of the device, the area of the current element, the doping density, the number of superparticles in the simulation, and the elementary charge. In this case, the area of each element is $A_{element} = h_x h_y$. This method have issues with a discontinuity in force when particles cross cells [9].

Cloud-in-cell method

The algorithm implemented in MCCT is the cloud-in-cell method(CIC) [5]. CIC approximates the charge distribution as a finite radius, constant distribution with its center in the superparticles' position. This has a smoothing effect on the electric potential since the charge is distributed to the particle's four closest nodes, instead of just the closest node. As the approximated force becomes linear instead of constant, it allows the force to be continuous across nodes [9]. This removes fluctuations and leads to increased stability compared to NGP.

The four closest nodes are given the charge densities

$$\rho_{i,j} = \rho_{sup} \left(1 - \frac{|x_k - x_i|}{h_x}\right) \left(1 - \frac{|y_k - y_j|}{h_y}\right), \tag{2.5}$$

where the mesh point is located at (x_i, y_j) and the point charge is located at (x_k, y_k) with h_x, h_y as the length between each node in respectively $x-$ and $y-$ directions. As seen in Equation (2.5), each node is given a fraction of the total charge depending on their distance from the particle. See Figure 2.4 for an illustration of CIC.

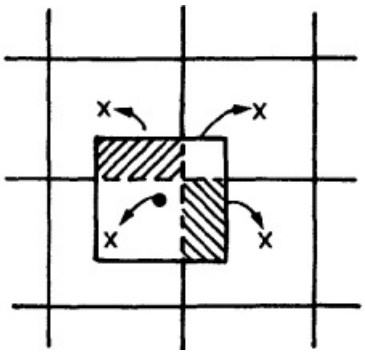


Figure 2.4: Illustration in two dimensions of CIC taken from [9]. The crosses are the four closest nodes of the grid. The dot is the superparticle.

There also exists higher order schemes for charge distribution, but the tradeoff in computational power makes them undesired [9].

2.3.3 Poisson solver

Partial differential equation formulation

The potential ϕ is given by Poisson's equation [10],

$$\nabla^2\phi = -\frac{\rho}{\epsilon}, \tag{2.6}$$

where ρ is the charge density and ϵ is the permittivity. Most of the computation time spent in MCCT is in the Poisson solver. Therefore, it is crucial that the solution is efficient.

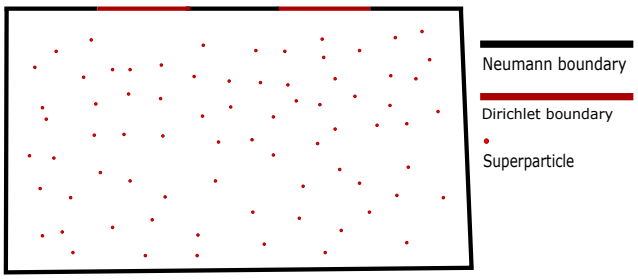


Figure 2.5: Illustration of the domain.

The typical problem is a bounded domain with constant permittivity, a charge density calculated from the cloud-in-cell algorithm, and boundary conditions. On the boundary, the electric field should be zero unless there are contacts that apply a fixed potential. Mathematically these are represented as Neumann or Dirichlet boundary conditions. An example of a problem setup for Poisson's equation is illustrated in Figure 2.5.

Let **n** denote the normal vector on the boundary, then the full partial differential equation(PDE) formulation, with Neumann and Dirichlet boundaries, is

$$\begin{aligned} \nabla^2\phi &= -\frac{\rho}{\epsilon} \text{ in } \Omega, \\ \phi &= D \text{ on } \partial\Omega|_D, \\ \frac{\partial\phi}{\partial\mathbf{n}} &= N \text{ on } \partial\Omega|_N, \end{aligned} \tag{2.7}$$

with $\partial\Omega = \partial\Omega|_D \cup \partial\Omega|_N$ and $\emptyset = \partial\Omega|_D \cap \partial\Omega|_N$, that is, the boundaries are specified as either Neumann boundaries or Dirichlet boundaries. The Neumann condition is usually zero because there is no electric field on the normal of the boundary where no potential is applied.

The finite difference method

The finite difference method [11] utilizes Taylor expansions to discretize Equation (2.7). The Neumann boundary conditions are applied using first order Taylor expansions. Dirichlet boundary conditions are applied by replacing the equations on the corresponding nodes in the resulting linear system.

The solution of the linear system is calculated with the successive over-relaxation method [3]. An alternative, the biconjugate gradient stabilized method is also implemented [12]. The first is a fast linear system solver intended for the solution of two-dimensional Poisson finite difference equations with constant step size, while the second is intended for usage in the case where the step size vary.

2.3.4 Electric field approximation onto grid

The electric field is implemented with centered difference method. Since the electric field is given by

$$\vec{E} = -\nabla\phi, \tag{2.8}$$

the centered difference method yields

$$\vec{E}_{i,j} \approx \begin{bmatrix} \frac{\phi_{i+1.j} - \phi_{i-1,j}}{2h} \\ \frac{\phi_{i.j+1} - \phi_{i,j-1}}{2h} \end{bmatrix}. \tag{2.9}$$

See appendix A.1 for a derivation of equation (2.9).

2.3.5 Electric field interpolation onto particles

The electric field interpolations onto particles are done using the NGP method. It is, in this case, given as

$$\vec{E}_{(x_p,y_p)} \approx \vec{E}_{(\lfloor \frac{x_p}{h_x}+\frac{1}{2} \rfloor, \lfloor \frac{y_p}{h_y}+\frac{1}{2} \rfloor)}, \tag{2.10}$$

where $\lfloor \cdot \rfloor$ denotes the round down to nearest integer function and $\vec{E}_{(x_p,y_p)}$ denotes the electric field on particle p. In other words, the particle is treated as if it is in the closest node in the grid.

2.4 Particle boundary conditions

Semiconductor devices typically consist of two contacts where carriers can pass into or out of the device. The types of contacts are Schottky contacts or ohmic contacts. In Schottky contacts, there is a potential barrier for carriers to break through to get into the device. For ohmic contacts, carriers can freely enter because there is no such potential barrier. MCCT treats the case of ohmic contacts. For more details on the study of contacts see [13, 14].

The contact implementation is a pivotal factor in MCCT since it controls the number of particles entering and exiting the device, and thereby controls the current through the device. Even though it is a crucial component in the model, it is not the study of contacts that is of interest. Therefore, among the most popular models of contacts, one was chosen which was simple and had an intuitive physical interpretation. The contact model implemented in MCCT uses the approach of Fischetti and Laux described in [15].

This model imposes conditions of charge neutrality and thermal equilibrium in a small region adjacent to the contacts. If there is a net deficit of majority carriers, injection happens until charge neutrality is maintained. The contact region is extended into the domain and is typically a few mesh cells wide. This net charge is calculated by counting free and immobile charges within the region. The injected particles are assigned a k-vector drawn from a thermal distribution to ensure thermal equilibrium.

The contacts absorb particles that will hit the contact surface during the next time step. If the estimated particle positions fall at or beyond the contact surface, the particles are absorbed and removed from the simulation. What types of superparticles are injected and absorbed is dictated by the simulation. Depending on the substance of the materials either contact can inject only holes, electrons, or both. Absorption, on the other hand, is available for both electrons and holes in both contacts. The contacts will get a surface charge on the metal because of the fields from the electrostatic potential, given by Poisson's equation.

2.5 New program flow

The goal is to implement the new FEM solver as an alternative. There are many considerations to cover in such an implementation, and therefore this section is made as a guidance to the following chapters. There are four main categories of objectives in the FEM implementation.

1. Discretize the potential using FEM
2. Calculate the charge distribution
3. Calculate the potential
4. Calculate the electric field

However, they are all intricately connected and depend on other problems that need to be solved. Finding the charge distribution relies on being able to locate where in the mesh a particle is located. Therefore an algorithm for particle localization is implemented in Chapter 5. With a straightforward implementation, the electric fields will have an error associated with them called self-forces. These have to be removed, and this is done in Chapter 4 using reference potentials. The calculation of potentials relies on a linear system solver, in this case, implemented as Preconditioned Conjugate Gradient method in Chapter 6.

The FEM implementation can now be categorized into two sections. The first is the initialization and reference potential calculations. The program flow of this part is depicted in figure 2.6. The first row is the setup of the FEM discretization, all of its contents is described in Chapter 3. The second row makes the reference potentials needed for self-force reduction. This row is mostly described in Chapter 4, which in turn relies on Chapter 3. The linear system solver in this row is described in its entirety in Chapter 6. The third row initializes the needs of the second category, the PMC scheme, and also covers the structures needed for point location. This row is described in Chapter 3 and 5.

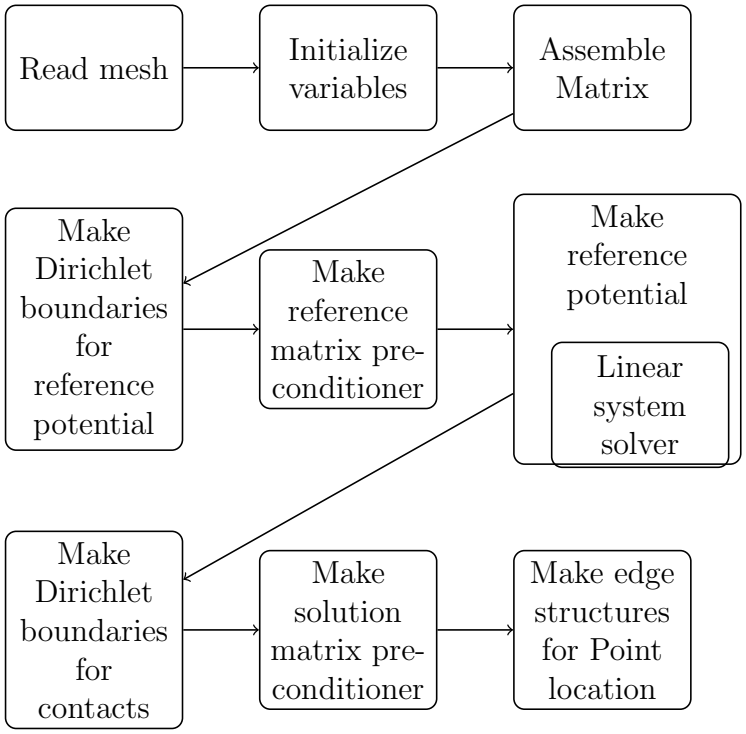


Figure 2.6: Initialization of FEM solution. First row is the basic initialization of the FEM. Second row is the reference potential calculations. The third row is the preparation for the time loop.

The program flow depicted in figure 2.7 happens in place of the PMC scheme in figure 2.1. The PMC scheme is implied in the calculation of the charge distribution and the electric field. For this master's thesis, the updating of particle element positions is described in its entirety in Chapter 5. The calculation of the charge distribution is described in Chapter 3, in particular, in Section 3.5. The linear system solver is described in Chapter 6. The calculation of the electric field is described in Chapter 3, but it depends on the self-force reduction, described in Chapter 4.

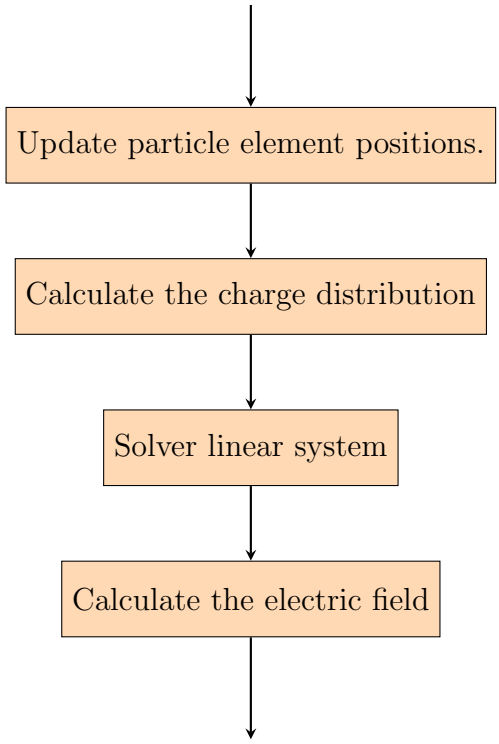


Figure 2.7: FEM flowchart inside MCCT time loop.

Chapter 3

FEM

As analytical solutions to partial differential equations are hard to obtain there is a need for a numerical method. The previous Poisson solver utilized a finite difference scheme. However, as the complexity of the domain increases and three-dimensional solutions are needed, the finite difference scheme meets its limitations. An alternative approach for obtaining an approximate solution will be presented, the finite element method. In this master's thesis, only the two-dimensional solution is presented and implemented.

FEM divides the domain into finite elements, then upon each of these domains derives the weak formulation of the PDE. In the sense of distributions, strong and weak formulations are equivalent [16]. The weak formulation is discretized to form a solvable linear system. The Dirichlet boundary conditions are implemented with the technique of lifting functions. For a more rigorous derivation see [16] which has heavily influenced this chapter.

Here is a listing of the steps in FEM:

- Deriving the weak formulation.
- Discretizing the weak formulation.
- Choosing a subspace of functions.
- Choosing a basis and transforming to a linear system.
- Solving the linear system.

In Section 3.1 the weak formulation is derived. In Section 3.2 the discretization is performed both in the sense of the discretization of the domain and the basis of functions. In Section 3.3 the area coordinates are introduced and used for analytical integration. In Section 3.4 the Dirichlet boundary implementation is described. In Section 3.5 the charge distribution is calculated and presented in a general framework. In Section 3.6 the new contact treatment is shown. In Section 3.7 a result

is derived for guidance in the choice of parameters. In Section 3.8 the assembly algorithm of the solution matrix and the load vector is presented. In Section 3.9 electric field approximations are discussed, and in Section 3.10 the chosen electric field approximation is presented.

3.1 Weak formulation

For the derivation of the weak formulation of Poisson's equation, the following definition is needed.

Definition 1. *$H^1(\Omega)$ is the set of functions such that their first order distributional derivatives are square integrable on the domain Ω.*

Introducing the transformation $\mathring{\phi} = \phi - D_g$, where $D_g \in H^1(\Omega)$ is some function on the domain with $D_g = D$ on the Dirichlet boundary, yields the PDE formulation:

$$\begin{aligned} \nabla^2 \mathring{\phi} &= -\frac{\rho}{\epsilon} \text{ in } \Omega, \\ \mathring{\phi} &= 0 \text{ on } \partial\Omega|_D, \\ \frac{\partial \mathring{\phi}}{\partial \mathbf{n}} &= N \text{ on } \partial\Omega|_N. \end{aligned} \tag{3.1}$$

Multiplying with an arbitrary function $f \in H^1(\Omega)$, integrating over Ω using Green's theorem, and expanding the transformation yields

$$\int_\Omega f \nabla^2 \phi d\Omega = -\int_\Omega f \frac{\rho}{\epsilon} d\Omega, \tag{3.2}$$

$$\int_\Omega \nabla f^\top \nabla \mathring{\phi} d\Omega = \int_\Omega f \frac{\rho}{\epsilon} d\Omega + \int_{\partial\Omega} f \nabla \phi d\vec{S} - \int_\Omega \nabla D_g^\top \cdot \nabla f d\Omega. \tag{3.3}$$

At this point, there is a need to change the space of functions such that $f|_{\partial\Omega_D} = 0$ because $\nabla\phi$ is unknown at the Dirichlet boundaries. This modifies Equation (3.3) to

$$\int_\Omega \nabla f^\top \cdot \nabla \mathring{\phi} d\Omega = \int_\Omega f \frac{\rho}{\epsilon} d\Omega + \int_{\partial\Omega|_N} f N d\vec{S} - \int_\Omega \nabla D_g^\top \cdot \nabla f d\Omega. \tag{3.4}$$

Equation (3.4) is the weak formulation, and with proper boundary conditions Lax-Milgram's lemma [17] ensures that there always exists a unique weak solution. The solution is now given by $\phi = \mathring{\phi} + D_g$.

3.2 Discretization

In any numerical discretization, the mesh is an essential part. In this case, a triangulation of the domain is used. The Delaunay algorithm is applied through the usage of the free mesh generation software GMSH [18]. There are several commercial or free mesh generating programs available, whereas GMSH is chosen because it is free and widely used. A discussion on GMSH and alternatives is found in Appendix A.4. The mesh generated has the Delaunay property, which is the property that no circumcircle of any element contains a node of the mesh. This property leads to many favorable results for the mesh. For instance, in the plane, it maximizes the minimum angle of triangles, which is good concerning finite precision geometry. It is known that, on average, each node is surrounded by six triangles. By denoting this triangulation as τ, the empty set as $\emptyset$, and the domain as Ω, the following relations are true.

$$\Omega = \underset{K\in\tau}{\cup} K \ , \ \emptyset = \underset{K\in\tau}{\cap} K. \tag{3.5}$$

Equation (3.5) states that each K is an element within the mesh τ, the elements are disjoint, and their union is the space Ω. It should be noted that GMSH uses a characteristic step length, denoted h, to decide the size of elements. Unfortunately, the documentation[18] is poor on how it is used, but it will be assumed that it defines the wanted average edge length. The average edge length is dependent on the domain, and therefore the characteristic step length will not always be the actual average edge length.

Introducing the space of finite elements, with the linear approximation approach as follows:

$$X^1 = \{v \in C^0(\bar{\Omega}) : v|_K \in \mathcal{P}_1 \forall K \in \tau\}, \tag{3.6}$$
$$V = \{v \in X^1 : v|_{\partial\Omega_D} = 0\}. \tag{3.7}$$

In other words, V is the space of continuous functions that are linear on each element and zero at the Dirichlet boundary. Redefining variables so that they are now elements of the space $V(\Omega) \subset H^1(\Omega)$ gives the approximation to equation (3.4): Find $\mathring{\phi} \in V$ such that

$$\int_\Omega \nabla f^\top \cdot \nabla \mathring{\phi} d\Omega = \int_\Omega f \cdot \frac{\rho}{\epsilon} d\Omega + \int_{\partial\Omega|_N} f \cdot N d\vec{S} - \int_\Omega \nabla D_g^\top \cdot \nabla f d\Omega, \forall f \in V. \tag{3.8}$$

The functions $f, \mathring{\phi} \in V, D_g \in X^1$ are characterized by their values taken at nodes N_i in the grid. Therefore, the basis of V is chosen to be the linear function defined in each node $\mathbf{N_i}$ as

$$b_j(\mathbf{N_i}) = \delta_{\mathbf{ij}} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j \end{cases}, \tag{3.9}$$

as illustrated in Figure 3.1.

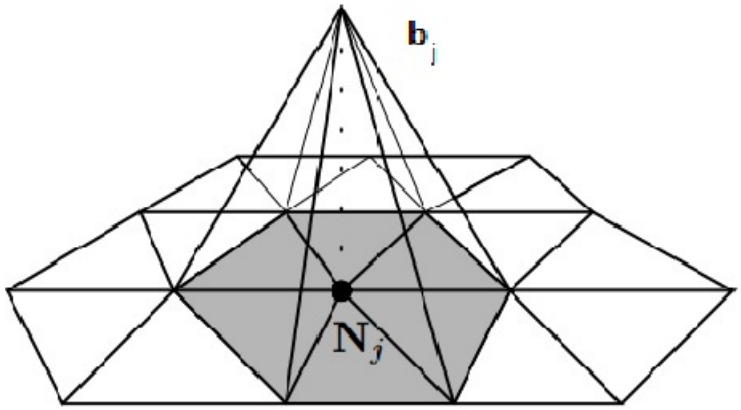


Figure 3.1: The basis function $b_j \in V$ and its support, taken from [16]

Each function in V can now be expressed from the basis as

$$f(\vec{x}) = \sum_{i \in \mathbf{N}} f_i b_i(\vec{x}) \ , \forall \vec{x} \in \Omega \text{ with } f_i = f(\mathbf{N}_i). \tag{3.10}$$

Now, all functions are expressed in this basis. $\mathbf{N}_D$ is defined to be the set of nodes on the Dirichlet boundary and $\mathbf{N}_I$ to be the rest of the nodes in the mesh, both sets represented as integers. Then, function (3.8) is reformulated as

$$\begin{aligned} \sum_{j \in \mathbf{N}_I} \mathring{\phi}_j \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega \ &= \ \int_\Omega b_i \cdot \frac{\rho}{\epsilon} d\Omega + \int_{\partial\Omega|_N} b_i \cdot \nabla\phi d\vec{S} \\ &\quad - \sum_{k \in \mathbf{N}_D} D_k \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \\ &\quad \forall i \in \mathbf{N}_I. \end{aligned} \tag{3.11}$$

Defining,

$$\begin{aligned} M &= [m_{ij}], \\ m_{ij} &= \textstyle\int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega, \\ \vec{u} &= [u_j], \\ u_j &= \mathring{\phi}_h(\mathbf{N}_j), \\ \vec{f} &= [f_i], \\ f_i &= \textstyle\int_\Omega b_i \cdot \frac{\rho}{\epsilon} d\Omega + \int_{\partial\Omega|_N} b_i \cdot \nabla\phi d\vec{S} - \sum_{k \in \mathbf{N}_D} D_k \int_\Omega \nabla b_i \cdot \nabla b_k d\Omega, \end{aligned} \tag{3.12}$$

the linear system becomes

$$M\vec{u} = \vec{f}. \tag{3.13}$$

M is referred to as the solution matrix and $\vec{f}$ is referred to as the load vector. The approximation to equation (3.1) is now given by $\vec{u}$ in the nodes N_I and D_k in the nodes N_D. The approximation to ϕ in any position $\vec{r} \in \Omega$ is given by the basis as

$$\phi(\vec{r}) \approx \sum_{i \in N_I} u_i b_i(\vec{r}) + \sum_{k \in N_D} D_k b_k(\vec{r}). \tag{3.14}$$

For ease, Equation (3.14) is used re-enumerated with ϕ_i as

$$\phi(\vec{r}) \approx \sum_{i \in N} \phi_i b_i(\vec{r}). \tag{3.15}$$

3.3 Analytical integration

The next few sections will be about calculating the solution matrix and each term in the load vector. Since the Neumann boundary condition always is zero for MCCT, it vanishes, and there is no need to handle it. In this section, the terms of the solution matrix are solved analytically.

For this linear basis, the natural coordinate system to use is area coordinates [19]. The advantage of area coordinates over the Cartesian coordinate system is that each element will not have to be transformed to a reference element for calculations. The coordinates are defined as shown in Figure 3.2. This coordinate system is not linearly independent, and it is reflected in the fact that the total area of a triangle K is $A_K = A_1 + A_2 + A_3$. For ease, A_K will be denoted as A as long as there can be no confusion from this.

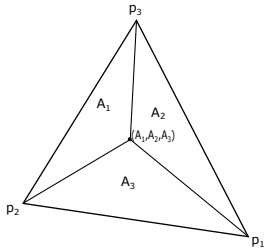


Figure 3.2: Definition of area coordinates.

The basis used in section 3.2 can be expressed through area coordinates as

$$\begin{aligned} b_1 &= \frac{A_1}{A}, \\ b_2 &= \frac{A_2}{A}, \\ b_3 &= \frac{A_3}{A}. \end{aligned} \tag{3.16}$$

Because of Equation (3.5) one can integrate over each element. For ease of calculations it is useful to define

$$m_{ij} = \int_\Omega \nabla b_i^\top \nabla b_j d\Omega = \sum_{K \in \tau} \int_K \nabla b_i^\top \nabla b_j dK. \tag{3.17}$$

M^K is defined with elements

$$m^K_{i_K j_K} = \int_K \nabla b_i^\top \nabla b_j dK, \tag{3.18}$$

where there is an implied transformation $(i_K, j_K, K) = (i, j)$ for the local element indices i_K and j_K, so that the solution matrix can be obtained as

$$m_{ij} = \sum_{K \in \tau} m^K_{i_K j_K}. \tag{3.19}$$

The solution is obtained for each 3-by-3 matrix M^K analytically. Since the basis functions are linear, differentiation gives a constant. Hence, local area coordinates yield

$$m^K_{i_K j_K} = \int_K \nabla b_l^\top \nabla b_q dK = \nabla b_l^\top \nabla b_q \int_K dK = \frac{\nabla A_{i_K}^\top \nabla A_{j_K}}{A_K}. \tag{3.20}$$

For vector cross products with the z-axis constant the triangle area spanned by the vectors u, v is given by $\frac{||u \times v||}{2}$, where $||\cdot||$ denotes the Euclidean norm. Each area A_i can, therefore, be represented as

$$\begin{aligned} A_1 &= \frac{||\overrightarrow{(p_3 - p_2)} \times \overrightarrow{(p - p_2)}||}{2}, \\ A_2 &= \frac{||\overrightarrow{(p_1 - p_3)} \times \overrightarrow{(p - p_3)}||}{2}, \\ A_3 &= \frac{||\overrightarrow{(p_2 - p_1)} \times \overrightarrow{(p - p_1)}||}{2}, \end{aligned} \tag{3.21}$$

where p_i is defined in Figure 3.2. These coordinates are specified in the Cartesian coordinate system by GMSH. Now the gradient can be taken on each basis as

shown in appendix A.2. Taking the dot product of each integrand $\nabla A_i^T \nabla A_i$, the local element matrix becomes

$$M^K = \frac{1}{4A_K} \begin{bmatrix} ||\overrightarrow{p_3 - p_2}||^2 & \overrightarrow{(p_3 - p_2)} \cdot \overrightarrow{(p_1 - p_3)} & \overrightarrow{(p_3 - p_2)} \cdot \overrightarrow{(p_2 - p_1)} \\ \overrightarrow{(p_3 - p_2)} \cdot \overrightarrow{(p_1 - p_3)} & ||\overrightarrow{p_1 - p_3}||^2 & \overrightarrow{(p_1 - p_3)} \cdot \overrightarrow{(p_2 - p_1)} \\ \overrightarrow{(p_3 - p_2)} \cdot \overrightarrow{(p_2 - p_1)} & \overrightarrow{(p_1 - p_3)} \cdot \overrightarrow{(p_2 - p_1)} & ||\overrightarrow{p_2 - p_1}^2|| \end{bmatrix}. \quad (3.22)$$

3.4 Dirichlet implementation

In this section the term,

$$\sum_{k \in \mathbf{N}_D} \int_\Omega \nabla b_i \cdot \nabla b_k d\Omega, \quad (3.23)$$

is handled. Since this term is identical to the M matrix term, instead of calculating it by itself, the solution matrix M is calculated assuming no Dirichlet boundary conditions. Then the Dirichlet nodes are removed from M and inserted on the right-hand side, multiplied by D_i. The boundary node positions $\vec{r_i}$ are elements in one of the sets $\partial\Omega|_D$ or $\partial\Omega|_N$, so that one can define

$$\vec{u} = \begin{bmatrix} \vec{u}_D \\ \vec{u}_N \\ \vec{u}_I \end{bmatrix}, \quad (3.24)$$

where $\vec{u}_D$ corresponds to Dirichlet nodes in the grid, $\vec{u}_N$ corresponds to Neumann nodes, and $\vec{u}_I$ corresponds to interior nodes. Now M can be partitioned such that $M_{i,j}$ have rows i and columns j corresponding to either Dirichlet, Neumann or interior nodes. For instance, by re-enumerating nodes appropriately, it can be put in the following form,

$$M = \begin{bmatrix} M_{DD} & M_{DN} & M_{DI} \\ M_{ND} & M_{NN} & M_{NI} \\ M_{ID} & M_{IN} & M_{II} \end{bmatrix}, \quad (3.25)$$

where M_{ID} means rows corresponding to interior nodes, and columns corresponding to Dirichlet nodes. Since M_{ND} and M_{ID} are multiplied by Dirichlet nodes, their values are already known, and they can be removed and set on the right-hand side:

$$\begin{bmatrix} M_{DD} & M_{DN} & M_{DI} \\ 0 & M_{NN} & M_{NI} \\ 0 & M_{IN} & M_{II} \end{bmatrix} \vec{u} = \vec{f} - \begin{bmatrix} M_{ND} \\ M_{ID} \end{bmatrix} \vec{u}_D, \tag{3.26}$$

where $\vec{f}$ denotes the right-hand side terms except the Dirichlet conditions in Equation (3.11). Since $\vec{u}_D$ is known, the corresponding rows can be removed, and the final linear system is reduced to

$$\begin{bmatrix} M_{NN} & M_{NI} \\ M_{IN} & M_{II} \end{bmatrix} \vec{u} = \vec{f} - \begin{bmatrix} M_{ND} \\ M_{ID} \end{bmatrix} \vec{u}_D \tag{3.27}$$

Now the systems dimensions have been reduced by the number of Dirichlet nodes, and since Equation (3.25) was symmetric, Equation (3.27) is symmetric. This representation is a simplification of the actual situation. In reality, the matrix has several boundary partitions. However, it can always be partitioned in a similar sense and yield a similar result. In theory, one can always re-enumerate the nodes to achieve the above representations, but in practice, the implementation deals with the number of partitions at hand without additional re-enumerating of nodes. In any case, it will always end up as a symmetric positive definite matrix. It is also important to note that, in the above calculations, there is an implied re-enumeration of nodes due to the dimensional reduction. For instance, the first element $M_{(1,1)}$ is now a Neumann node and not a Dirichlet node. It should be clear to the reader that this is an equivalent, but more practical approach to the system defined in Equation (3.11). That is, the term

$$\sum_{k \in \mathbf{N}_D} \int_\Omega \nabla b_i \cdot \nabla b_k d\Omega, \tag{3.28}$$

has been replaced with the vector product

$$\begin{bmatrix} M_{ND} \\ M_{ID} \end{bmatrix} \vec{u}_D. \tag{3.29}$$

3.5 Charge distribution

In this section the charge distribution term in Equation (3.11) is handled. That is, the calculation of

$$\int_\Omega b_i \cdot \frac{\rho}{\epsilon} d\Omega. \tag{3.30}$$

There are two contributions to this charge distribution,

$$\rho = \rho_{particles} + \rho_{bc}, \tag{3.31}$$

where $\rho_{particles}$ is the distribution from the moving superparticles and ρ_{bc} is the contribution from the immovable charges in the device, denoted as the background charge.

3.5.1 Background charge

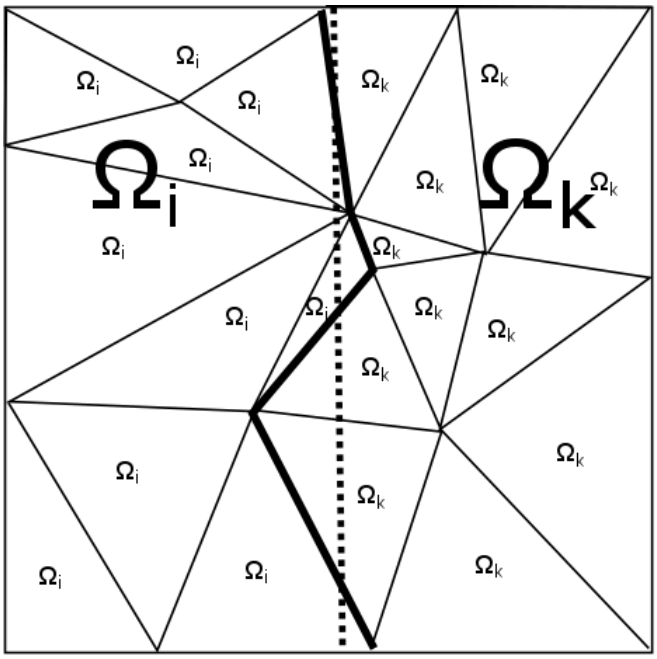


Figure 3.3: Illustration of redefined boundary between Ω_i and Ω_k. The dotted line is the actual boundary line between the sets, the bold line is the redefined line. The small Ω_k and Ω_i denote which one the elements are defined as subsets of.

The background charge is considered constant in regions of the device, given by the doping in the material. Let Ω_j denote the regions of different charge densities. Figure 3.3 shows an example, the dotted line is the boundary between the regions Ω_i and Ω_k. Let ρ_{bc}^j denote the specific density in each region Ω_j then

$$\rho_{bc}^j = qn_D^j, \tag{3.32}$$

where q is the signed elementary charge, n_D^j is the constant doping density in region Ω_j and the sign of q depends on whether the background charge originates from donors or acceptors. Since ρ_{bc}^j is constant in each region, the integral on each element $K \in \tau$ within a region Ω_j, that is $K \subset \Omega_j$, is

$$\int_K b_i \cdot \frac{\rho_{bc}^j}{\epsilon} dK = \frac{A_K}{3} \frac{\rho_{bc}^j}{\epsilon}, \tag{3.33}$$

so that each node of a triangle gets an equal contribution of one-third of the charge. When an element is on the boundary between regions Ω_j, the above integral needs to be split. Denote γ as the enumeration j so that $\Omega = \cup_{j\in\gamma}\Omega_j$, then

$$\sum_{j\in\gamma} \int_{K\cap\Omega_j} b_i \cdot \frac{\rho_{bc}^j}{\epsilon} dK = \sum_{j\in\gamma} \frac{\lambda(K\cap\Omega_j)}{3} \frac{\rho_{bc}^j}{\epsilon}, \tag{3.34}$$

where $\lambda(\cdot)$ denotes the Lebesgue measure of the argument set, which means the area of this intersection of sets. However, in practice, this is cumbersome and therefore the implementation uses only equation (3.33). The boundaries are redefined to coincide with some edge in between elements. With finer grids, this will give approximately the same solution. Each element is well-defined to be in an Ω_i by defining that its element center must be in Ω_i as shown in Figure 3.3.

Since the equations are for each node and not for each element, the contribution to each node is written as

$$\int_\Omega b_i \cdot \frac{\rho_{bc}}{\epsilon} = \sum_{K\in\tau_i} \frac{A_K}{3} \frac{\rho_{bc}}{\epsilon}, \tag{3.35}$$

where τ_i is defined as the elements which are in the support of b_i. Since the regions Ω_j are disjoint, ρ_{bc} is well defined as $\rho_{bc} = \rho_{bc}^j$ in any region of Ω and therefore the index is omitted.

3.5.2 Particle-mesh coupling

For $\rho_{particles}$, the integration becomes different depending on the chosen charge distribution. It is time to introduce the assignment scheme function. Denoting the assignment scheme function to be l_i, it is given as

$$l_i = \int_\Omega b_i l'(r_p) d\Omega, \tag{3.36}$$

where $l'(r_p)$ approximates $\frac{\rho}{q_p}$ as a mesh coupling of the superparticle located at r_p, with the charge q_p. q_p is either positive or negative depending on if the superparticle

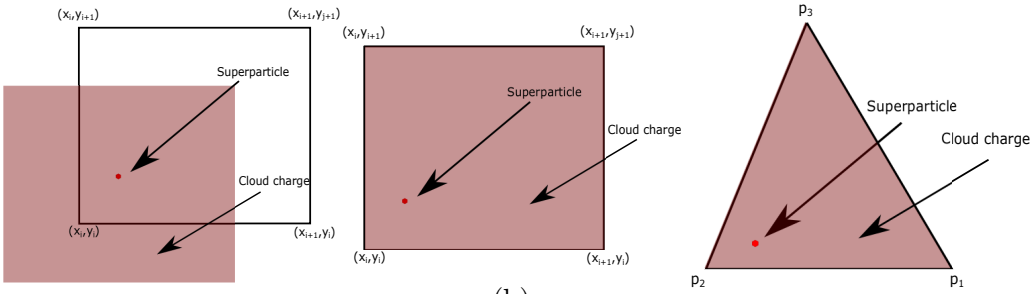


Figure 3.4: Illustrating the relationship, between (a) CIC in a rectangular grid (b) NEC in a quadrilateral grid (c) NEC in a triangular grid, by showing the grid points, the cloud charge and the superparticle in one element.

represents electrons or holes. q_p is explicitly given as an equal charge distributed to all the superparticles in the simulation:

$$q_p = \frac{qn_D A_{device}}{n(P)}, \tag{3.37}$$

where q is the signed elementary charge, $n(P)$ is the number of superparticles, and n_D is the doping density. Equation (3.37) is made so that each particle has an equal amount of charge distributed to it, and it is the charge equivalent of ρ_{sup} from Equation (2.4), where the needed quantity was the charge distribution of each particle, and not the charge itself. This difference in what quantity is needed is mainly due to the integrals in FEM, which in a sense can be interpreted as counting the charges in each area. Taking the Neumann boundary as zero, Equation (3.11) becomes,

$$\begin{aligned}\sum_{j\in\mathbf{N}_I} \mathring{\phi}_j \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega \;=\; & -\sum_{p\in P} \frac{q_p}{\epsilon} l_i(r_p) + \sum_{K\in\tau_i} \frac{A_K}{3}\frac{\rho_{bc}}{\epsilon} \\ & - \sum_{k\in\mathbf{N}_D} \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \\ & \forall i \in \mathbf{N}_I.\end{aligned} \tag{3.38}$$

where l_i is a charge assignment scheme that could in principle be any scheme.

One example for a charge distribution scheme is the nearest-element-center scheme(NEC) which was first proposed by Laux in [5]. This scheme takes a similar approach to CIC which was described in Section 2.3.2. Before making a cloud, the superparticle's position is moved to the center of its element. This way the NEC scheme keeps the charge density locally within an element as shown in Figure 3.4c. Just like with CIC, the charge density uses ρ_{supp} from NGP.This yields an equal contribution to each node within an element, but not an equal amount to each node across elements.

In this case $l_i = \frac{A}{3}\frac{\rho_{supp}}{q_p \epsilon}$. Denote P_i to be the set of which particles are in the support of b_i and A_{K_p} to be the area of the element this particle is in. Then, Equation (3.11) becomes

$$\begin{aligned}\sum_{j \in \mathbf{N}_I} \mathring{\phi}_j \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega &= -\sum_{p \in P_i} \frac{A_{K_p}}{3}\frac{\rho_{supp}}{\epsilon} + \sum_{K \in \tau_i} \frac{A_K}{3}\frac{\rho_{bc}}{\epsilon} \\ &\quad - \sum_{k \in \mathbf{N}_D} D_k \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \\ &\quad \forall i \in \mathbf{N}_I.\end{aligned} \tag{3.39}$$

Another example of an assignment scheme is $l_i = b_i(r_p)$, that is the scheme where the source is modelled as a Dirac delta. $l' = \delta(\vec{r} - \vec{r_p})$ yields that

$$\begin{aligned}\sum_{j \in \mathbf{N}_I} \mathring{\phi}_j \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega &= -\sum_{p \in P} \frac{q_p}{\epsilon} b_i(r_p) + \sum_{K \in \tau_i} \frac{A_K}{3}\frac{\rho_{bc}}{\epsilon} \\ &\quad - \sum_{k \in \mathbf{N}_D} \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \\ &\quad \forall i \in \mathbf{N}_I,\end{aligned} \tag{3.40}$$

where P denotes the set of all superparticles. The implemented choice has been a merge of these two methods; it uses the Dirac delta function where the charge is set in the center of an element. The reason for this choice is that the Dirac delta function is most appropriate with the self-force reduction scheme which will be presented in Chapter 4, and the electric field approximation presented in Section 3.10 will be constant within an element. In that case, it is unlikely for the position of the particle within an element to have much influence on the result. In addition to this, it will be seen in Chapter 4 that additional possibilities for improvement show itself using this scheme. With the scheme chosen, the equation to be solved is

$$\begin{aligned}\sum_{j \in \mathbf{N}_I} \mathring{\phi}_j \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega &= -\sum_{p \in P_i} \frac{1}{3}\frac{q_p}{\epsilon} + \sum_{K \in \tau_i} \frac{A_K}{3}\frac{\rho_{bc}}{\epsilon} \\ &\quad - \sum_{k \in \mathbf{N}_D} \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \\ &\quad \forall i \in \mathbf{N}_I,\end{aligned} \tag{3.41}$$

where P_i and τ_i denote respectively, particles and elements in the support of b_i.

3.6 Contact treatment

As described in Section 2.4, the model must have ohmic contacts. Unfortunately, the previous implementation does not cope well with FEM. Therefore, the approach has been altered. The old contact model imposed conditions of charge neutrality and thermal equilibrium, and therefore this is chosen for FEM as well. The thermal equilibrium is still kept by assigning a k-vector drawn from a thermal distribution. The charge neutrality is now done by checking the charge of elements in the region. There are alternative approaches to check if the contacts become interesting or important at some stage, for instance, [20] is using tetrahedral or triangular elements.

From Equation (3.31), the background charge needs to equal the contribution of moving charges in each element. From Equation (3.41) it means that for each element, it is in theory a neutral charge element if

$$\sum_{p \in P_K} \frac{1}{3} \frac{q_p}{\epsilon} = \frac{A_K}{3} \frac{\rho_{bc}}{\epsilon}, \tag{3.42}$$

where the sum over $p \in P_K$ is taken over particles in the support of b_i. Now this yields that a charge neutral element needs $n(P_K)$ particles. That is,

$$n(P_K) = \frac{\rho_{bc} A_K}{q_p}, \tag{3.43}$$

which means that the number of particles needed in an element is equal to the total charge of an element per particle charge.

The contacts are dealt with by checking how many particles are needed in total in the contact region. Then particles are injected to a randomly chosen element in this region with a bias towards elements which are far from charge neutral. This approach is naive, but as has been stated, the contacts are not of interest in the simulator, and that is the reason it has been done in a rough and simple manner. This approach assumes that there will be elements that are large enough to accommodate particles in the contact region, restricting the characteristic step length. The issue can be resolved by considering merged elements for charge neutrality.

3.7 Mesh considerations

The contacts treatment in the previous section can be taken even further. By continuing the calculations, one can get insight into the relationship between superparticles' charge, and the elements' background charge. The particles' charge is

given by Equation (3.37) when using the average doping. For a device of varying doping, denote each region Ω_j as the region with n_D^j constant doping, and γ as the enumeration j so that $\Omega = \cup_{j\in\gamma}\Omega_j$, then the particles' charge is given as

$$q_p = \frac{q \cdot \sum\limits_{i\in\gamma} n_D^i A_i}{n(P)}, \tag{3.44}$$

where q is the signed elementary charge, A_i is the regions area, and $n(P)$ is the number of superparticles. The charge associated with an element from the background charge is

$$q_K = qA_K n_D^j. \tag{3.45}$$

where j indicates the region Ω_j containing K. When there is an equal contribution from the background charge and a particle in the element, it means that the area of an element should be

$$A_K = \frac{\sum\limits_{i\in\gamma} n_D^i A_i}{n_D^j n(P)}. \tag{3.46}$$

This result is interesting as it gives an estimate of what area the average element should have to be of neutral charge with one particle in it. Also, since it is in terms of properties of the domain and the particles, it is suggested that it also gives an indicator of the particles' ability to interact with each other and the domain. Note that, Equation (3.46) is the same as Equation (3.43) when the latter uses the average doping density as the only density in the domain. These equations should be thought of as functions of the number of superparticles $n(P)$ or functions of doping densities n_D^i. This means that for a set device and mesh, this equation can tell how many particles must be used, or with a set number of particles viable, it can be used to generate an appropriate mesh.

Although there is no proof, it will here be suggested that Equation (3.46) should be used as an upper bound for the size of the elements in areas of interest. Unfortunately, for the mesh generation used in this master's thesis, there is no simple way to configure the areas of elements. Therefore, to get an estimate for the characteristic length, which configures element size in GMSH, an assumption will be made. Since the mesh is Delaunay, the minimum angle of triangles are maximized, therefore a reasonable assumption is that, on average, the areas of the elements can be approximated as equilateral triangles. Equation (3.46) can then be approximated as

$$h = \alpha \sqrt{\frac{4}{\sqrt{3}} \cdot \frac{\sum_{i \in \gamma} n_D^i A_i}{n_D^j n(P)}}. \tag{3.47}$$

where $\alpha(\vec{r})$ is a scaling factor which decides the degree of refinement in an area. This relationship is used in the PN-junction in Chapter 7, but except for that, there is no further testing of the relationship. Note that due to the contact implementation, $\alpha \geq 1$ near the boundaries. Except for that, the user is free to vary α as needed in the domain. The following list is a suggestion of how the parameter $\alpha \geq 0$ should vary in a simulation.

- In contact region, $\alpha \geq 1$.
- Regions of few or no particles, $\alpha \gg 1$.
- Regions of many particles and important particle interactions, $\alpha \ll 1$.

It is important to note that this estimate is a suggestion, and there is no requirement to use it for the solver to work. Often, the user of FEM implementations needs to test the mesh and see what works in different circumstances.

3.8 Assembly of stiffness matrix and load vector

As is clear by now, the assembly can be done element-wise. The algorithm for assembly is presented here for completeness. The load vector has contributions only from individual particles and is therefore assembled by iterating over particles in elements.

The algorithm needs a corresponding index transformation, $(i_K, j_K, K) \rightarrow (i, j)$, which depends on the numbering of nodes in the triangulation, which is essentially the connectivity of nodes in the grid. The GMSH triangulation contains this mapping. For a thorough discussion of the assembly process when using other triangulation methods see [21]. Since the matrix, and other matrices involved, are sparse, they are stored in the compressed row storage format (CRS) which essentially stores only the nonzero values, their column position and the number of non-zero values in each row. This format is described in Appendix B.1

3.9 Electric field approximation

The goal of the potential approximation is to find the electric field to make MCCT self-consistent by applying the forces of the electric field onto the particles. Therefore, an approximation based on the potential is needed to get the electric field

Algorithm 1 Matrix and vector assembly

input Triangulation τ, superparticles.
ouput global stiffness matrix M and global load vector f.
$\forall K \in \tau$ calculate locally $M^K_{i_K j_K}$
$M_{ij} = M_{ij} + M^K_{i_K j_K}$
Remove Dirichlet boundaries by redefining matrix as
$M := \begin{bmatrix} M_{NN} & M_{NI} \\ M_{IN} & M_{II} \end{bmatrix}$
Insert dirichlet conditions as
$f = f + \begin{bmatrix} M_{ND} \\ M_{ID} \end{bmatrix} \vec{u}_D.$
$\forall K \in \tau$ calculate locally the charge distribution f^k_i
$f_i = f_i + f^K_{i_K}$
In each time iteration calculate moving charges contribution:
$\forall$ superparticle in K calculate locally f^K_i
$f_i = f_i + f^K_{i_K}$

doing work on each particle. No matter how this is done, the approximation can be written as,

$$\vec{E_p} \approx z(\phi, \vec{r_p}) \tag{3.48}$$

where $z(\phi, \vec{r_p})$ approximate the electric field in some way dependent on the potential field calculated and the position of the particle. There are several possibilities for doing this, however in the particular case of finite element method, there is especially one which is natural to use and which will be used here. This is because FEM does not approximate the function just in the nodes, it approximates the function using a basis, in this case, a linear basis. Therefore, it is approximated as a function in terms of this basis. This approximated function has its properties in the domain, and not just at each grid node.

The choice of approximation for $\vec{E}_p$ matters for the properties of the method. However, the problems which stem from these properties are handled through other means than finding the most appropriate scheme. Therefore it is wise to use a simple scheme to make calculations few and quick.

3.10 Natural FEM electric field approximation

The electric field is given by

$$\vec{E} = -\nabla\phi. \tag{2.8}$$

From Equation (3.10), ϕ at a particle position $\vec{r_p} = (x, y)$ can be represented locally in its element as

$$\phi(\vec{r_p}) = \sum_{i \in N} \phi_i b_i(\vec{r_p}). \tag{3.49}$$

Using the above equations, the linearity of the gradient function, and the support of the basis functions, the electric field approximation becomes

$$z(\phi, \vec{r_p}) = \vec{E}^{K_p} = -\nabla\phi = -\sum_{i_k=1}^{3} \phi_{i_K} \nabla b_{i_K}(\vec{r_p}), \tag{3.50}$$

where $\vec{E}^{K_p}$ denotes the E field approximation on the element containing the particle p and ϕ_{i_K} denotes the approximated potentials at nodes belonging to the element with their respective basis functions b_{i_K} .

The area coordinates from Section 3.3 are used and the gradients can be found in Appendix A.2. From these gradients, and from Equation (3.50), the final electric field approximation in $x-$ and $y-$directions on a particle is

$$E^x = \frac{1}{2A_{K_p}}(\phi_1(p_3^y - p_2^y) + \phi_2(p_1^y - p_3^y)) + \phi_3(p_2^y - p_1^y)), \tag{3.51}$$

$$E^y = \frac{-1}{2A_{K_p}}(\phi_1(p_3^x - p_2^x) + \phi_2(p_1^x - p_3^x)) + \phi_3(p_2^x - p_1^x)), \tag{3.52}$$

where p_i^x and p_i^y respectively denotes the x- and y- coordinate of vertex p_i in element K which the superparticle is contained in.

Notice that due to the chosen basis and approximation function, z, the electric field becomes constant on each element and therefore is discontinuous when particles cross boundaries. As has been stated in [5] this lead to issues with stability for other methods than FEM. However, the finite element method deals best with rapid variations in the potential field using linear elements. Therefore, this issue has been ignored to be able to use the self-force reduction scheme presented in the next chapter. If there are issues with stability, then the method can be improved either by using some of the electric field interpolations presented in [22] to interpolate electric fields of elements onto nodes or the basis can be upgraded to use higher order polynomials.

Chapter 4

Self-force reduction

Self-forces are forces which stem from the properties of the mesh used in a discretization, and it is defined as the force which a particle exerts on itself during simulations. Self-forces has been a topic of discussion within particle simulations for a long time. Mostly, the NGP and CIC schemes were used with structured meshes to resolve the issue. In 1996 Laux published improved versions of NGP and CIC[5], which could be used for non-uniformly spaced tensor-product meshes but also showed that a simple PMC scheme of the same form as NGP and CIC would not produce zero self-force for unstructured triangular meshes. However, it did show that for equilateral triangular meshes, there is zero self-force if the NEC scheme is used.

Since FEM's main advantage is to be able to solve functions on arbitrary domains, and triangular elements enhance this advantage, many studies have been done to try to make this work. Among these, the most important paper on the theoretical aspect is [22]. This paper introduced the theoretical conditions needed to produce zero self-force for an unstructured triangular mesh. However, it was unable to produce such a scheme, like all before, and it drew the same conclusion as Laux, that this is only easily obtained if the elements are equilateral. However, they did show that, in practice, the self-force was greatly reduced when using interpolation of electric fields in elements onto nodes.

In 2015, Kalna et al. presented a method[23] to remove the self-force in PMC schemes by removing each particle's contribution to the grid. The method was presented as a PMC scheme using the Dirac delta representation and the natural PMC coupling that arise from this. By doing the calculations this way, the Coulomb potential was used to remove any self-force on each particle. A side-effect of this approach is that any PMC scheme can be utilized. As this method gave results which worked for arbitrary meshes, it has been chosen for the implementation in MCCT.

In this chapter, the self-force reduction is carried out and analyzed. In Section 4.1 the concept of self-forces is explained. In Section 4.2 the self-force reduction is introduced. In Section, 4.3 the self-force to grid coupling is introduced. In Section 4.4 the method is illustrated and discussed. In Section 4.5 the method is tested for

a single particle in an infinite domain.

4.1 Self-forces in PMC

Denote $\vec{r_p}$ as the position of a superparticle, henceforth any variable denoted as f_p will be considered as $f_p = f(\vec{r_p})$. In PMC schemes the potential a superparticle p experiences can be split up as follows:

$$\phi_p = \phi_{self} + \phi_{othersources}, \tag{4.1}$$

where $\phi_{othersources}$ denotes the potential which would be if the carrier wasn't present, and ϕ_{self} denotes the potential which the particle itself contributes. Because of linearity the electric field becomes

$$\vec{E_p} = -\nabla\phi_p = -\nabla\phi_{self} - \nabla\phi_{other} = \vec{E_{self}} + \vec{E_{other}} \tag{4.2}$$

Since the force is q_p multiplied by this electric field, and a particle does not exert a force on itself, it is apparent that $\vec{E_{self}} = 0$. However, this is not the case in PMC schemes in general. When the discretization is unable to conserve symmetries, the self-force becomes a factor.

4.2 Self-force reduction

The idea behind self-force reduction is to remove the charges own contribution to the electric field when calculating the electric field on this particle. This is done by placing a charge in infinite space,$\mathbb{R}^2$, so that it should have zero forces working on it. The self-force will then be the force that is doing work on the particle, $-q_p\nabla\phi_{self}$. The reference potential is defined to be equal to this self-potential, $-q_p\nabla\phi^{ref} = -q_p\nabla\phi_{self}$, so that it can be removed for the particle to produce zero self-force. This reference potential depends on the charge assignment scheme, $l'(\vec{r_p})$, and the electric field approximation scheme, $z(\phi, \vec{r_p})$. Both of these need to be chosen the same as in Chapter 3. In any case of scheme choices, the reference potential with an electron is given by

$$\nabla^2\phi^{ref'} = -\frac{q_p}{\epsilon}l'(\vec{r_p}) \tag{4.3}$$

Introducing scaling, define $\phi^{ref'}\frac{\epsilon}{q_p} = \phi^{ref}$, such that

$$\nabla^2\phi^{ref'}\frac{\epsilon}{q_p} = \nabla^2\phi^{ref} = -l'(\vec{r_p}) \tag{4.4}$$

Now the solution can be obtained in terms of unit charge and unit permittivity, then scaled for the simulator. The solution to (4.3) is now given as

$$\phi^{ref'} = \frac{q_p}{\epsilon}\phi^{ref} \tag{4.5}$$

When approximating the charge as an infinitely small particle, in infinite space, $\Omega_{inf} = \mathbb{R}^2$, the solution is the Coulomb potential:

$$\phi^{ref'} = \phi^{Coloumb} = \frac{-\ln(|\vec{r} - \vec{r_p}|)}{2\pi}. \tag{4.6}$$

Equation (4.6) is interpreted as the two-dimensional version of the Coulomb potential given that the charge is distributed with unit charge along the third axis. It is important that it is interpreted in this sense since the boundary condition is that the potential goes to zero as $\vec{r}$ tends to infinity in $\mathbb{R}^3$. This will be denoted as the infinite boundary condition. The actual domain for simulations is limited to the domain of the mesh Ω. To simulate Ω_{inf} with the finite Ω, the boundary nodes are approximated as the Coulomb potential. For a unit charge in a 2-dimensional domain, the boundary is therefore given by

$$\phi^{ref}|_{\partial\Omega_D} = \frac{-\ln(|\vec{r} - \vec{r_p}|)}{2\pi}. \tag{4.7}$$

If another scheme is used for $l'(\cdot)$, and not the Dirac delta function, then as the number of elements increase, the charge assignment gets closer to the Dirac delta function as long as the scheme used takes into account the size of each element. This means that the boundaries of a simulated infinite domain can at least be approximated by the Coulomb potential as Equation (4.6) on the boundaries. It is important to emphasize that in the case of the reference potentials, the boundary is this Dirichlet condition, and there are no Neumann conditions. This means that for this chapter, $\mathbf{N}_I^{ref}$ are the interior nodes and $\mathbf{N}_D^{ref}$ are the boundary nodes.

Before moving on, the goals of this section should be summarized. The steps that will follow are shown in the following list:

- Calculate the reference potential for a particle in the particle position $\vec{r_p}$ in the infinite domain Ω_{inf}
- Calculate the reference electric field in the particle position $\vec{r_p}$ in the infinite domain Ω_{inf}

- Remove the self-force from a particle in the position $\vec{r_p}$ in the simulation domain Ω

Following the finite element approach analogously to Chapter 3, the approximation to the reference potential is, for a single particle,

$$\begin{aligned} \sum_{j \in \mathbf{N}_I^{ref}} \mathring{\phi}_j^{ref} \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega &= -l(\vec{r_p}) \\ &\quad - \sum_{k \in \mathbf{N}_D^{ref}} \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \\ &\quad \forall i \in \mathbf{N}_I^{ref}, \end{aligned} \tag{4.8}$$

where D_k is given by Equation (4.6) with $\vec{r} = \vec{r_k}$ at boundary nodes k. Equation (4.8) solves the first step in this section.

The reference potentials role is to reduce the self-force of a particle, and therefore it is natural to take a look at the electric field before looking at the next step for the potential. The second step in this section is given by using the same approximation for the reference electric field, as for the electric field of particles. That is,

$$\vec{E_{ref}} \approx z(\phi, \vec{r_p}) = -\sum_{i_k=1}^{3} \phi_{i_K}^{ref} \nabla b_{i_K}(\vec{r_p}), \tag{4.9}$$

with i_K denoting nodes in an element K in which the particle's position, $\vec{r_p}$, is contained.

The idea is to reduce the electric field in the particle's position using the reference potential. At this time, the potential for the particle itself is given by (3.41) at each time step. The electric field, from equation (3.50) is given as

$$\vec{E_p} \approx z(\phi, \vec{r_p}) = -\sum_{i_k=1}^{3} \phi_{i_K} \nabla b_{i_K}(\vec{r_p}), \tag{4.10}$$

Since the particle's position is the same for MCCT and the reference potential particle, Equation (4.2) is used to remove the particle's own contribution to the electric field.

$$\vec{E}_p^{corrected} = \vec{E_p} - \vec{E}_{ref}. \tag{4.11}$$

Using equal assignment schemes for the potential and the reference potential, together with the scaling in Equation (4.5), yields that

$$E_p^{\vec{corrected}} = \sum_{i_k=1}^{3} \phi_{i_K} \nabla b_{i_K}(\vec{r_p}) - \frac{q_p}{\epsilon} \sum_{i_k=1}^{3} \phi_{i_K}^{ref} \nabla b_{i_K}(\vec{r_p}). \tag{4.12}$$

Since the electric field approximation is linear in its arguments one can define the solution as

$$\vec{E_p}^{\,corrected} = z(\phi, \vec{r_p}) - z(\frac{q_p}{\epsilon}\phi^{ref}, \vec{r_p}) = z(\phi - \frac{q_p}{\epsilon}\phi^{ref}, \vec{r_p}). \tag{4.13}$$

By defining

$$\phi^{corrected} = \phi - \frac{q_p}{\epsilon}\phi^{ref} \tag{4.14}$$

the corrected electric field potential at a particle's position is given by

$$\vec{E_p}^{\,corrected} = z(\phi^{corrected}, \vec{r_p}) \tag{4.15}$$

and the corrected potential for the particle's position is

$$\phi^{corrected}(\vec{r_p}) = \sum_{i_k=1}^{3} \phi_{i_K} b_{i_K}(\vec{r_p}) - \frac{q_p}{\epsilon} \sum_{i_k=1}^{3} \phi_{i_K}^{ref} b_{i_K}(\vec{r_p}) \tag{4.16}$$

However, it should be noted that $\phi^{corrected}$ is not an error corrected potential, it removes all the contribution from the particle in an infinite domain. Therefore, it can only be used as a definition to correct the electric field for that particular particle's contribution to itself in the grid.

4.3 Reference potential to grid

The above calculations are sufficient to get rid of the particles' self-forces, but it requires that the reference potential is calculated once for each superparticle at each time iteration. This is computationally expensive and is therefore not a manageable approach. It is important that the reference potentials are connected to the grid so that they can be calculated beforehand. The idea is to calculate the potential in each node or element, then use the superposition property as a tool to get the reference potential at any particle position. Since the particles are all positioned inside the center of each element in the used assignment scheme, the positions are discrete. Thus it is possible in this case to calculate the reference potential element-wise. However, there are far more elements than there are nodes in the grid, and for this

reason, the nodal approach is used. If accuracy is needed, the elemental approach does a better job since the positions of the reference potentials are the same as for the MCCT run. Also, the elemental approach has the added advantage of being able to calculate the reference potential in all elements. The nodal approach cannot reduce the self-force in elements that have a node on the boundary since these nodes cannot be calculated. The goal of this section is to do the reduction of the self-force by calculating the reference potentials before knowing the particle's position. The steps required to do so, with the nodal approach, are listed below:

- Calculate ϕ^{ref} with a particle set in each mesh node $p_{node} \in \mathbf{N}_I$, denoted as $\phi^{p_{node},R}$,
- Approximate ϕ^{ref} for a particle in $\vec{r_p} \in \Omega$ during the MCCT time loop.

Here, the approach of Kalna et al. [23] will be used. This approach calculates the reference potential for each grid node using a Dirac delta function for the assignment scheme function, that is

$$l'(r_p) = \delta(\vec{r} - \vec{r_p}) \tag{4.17}$$

and the electric field approximation is the natural one,

$$z(\phi, \vec{r_p}) = \sum_{i_K=1}^{3} \phi_{i_K} \nabla b_{i_K}(\vec{r_p}). \tag{4.18}$$

Now, for each internal node p_{node}, the reference potential is calculated and denoted as $\phi^{p_{node},R}$. That is, the discretized potential $\phi_j^{p_{node},R}$ is the reference potential in node j from a particle set in node p_{node}. Inserting l' and z into equations (4.8) and (4.15) leads to the reference potential calculations being done as

$$\sum_{j \in \mathbf{N}_I} \mathring{\phi}_j^{p_{node},R} \int_\Omega \nabla b_j \nabla b_i d\Omega = -\delta_{ip_{node}} - \sum_{k \in \mathbf{N}_D} D_k \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \, \forall i \in \mathbf{N}_I. \tag{4.19}$$

for each node p_{node} in the grid, where $\delta_{ip_{node}}$ is zero for $i \neq p_{node}$ and one for $i = p_{node}$. It is important to understand that $p_{node} \in \mathbf{N}_I$, it is in the non-Dirichlet nodes. D_k is given by Equation (4.6) with $\vec{r} = \vec{r_i}$ and $\vec{r_p} = \vec{r}_{p_{node}}$. Then from equation (3.40) the potential at each time step is calculated with FEM as

$$\sum_{j\in\mathbf{N}_I} \mathring{\phi}_j \int_\Omega \nabla b_j^\top \cdot \nabla b_i d\Omega = -\sum_{p\in P} b_i(\vec{r_p})\frac{q_p}{\epsilon} + \int_\Omega \frac{\rho_{bc}}{\epsilon} d\Omega - \sum_{k\in\mathbf{N}_D} D_k \int_\Omega \nabla b_i^\top \cdot \nabla b_k d\Omega, \quad \forall i \in \mathbf{N}_I. \tag{4.20}$$

Now that the reference potential is known for a particle located in each mesh node, the reference potential ϕ^{ref} for a particle located in $\vec{r_p}$ must be approximated. At this point, this report will diverge from the method of Kalna et al. Their approach is to use the reference potential directly as the contribution of the self-force to the particle, by doing the reduction as,

$$\phi^{corrected}(\vec{r_p}) = \sum_{i_k=1}^{3} \phi_{i_K} b_{i_K}(\vec{r_p}) - \frac{q_p}{\epsilon} \sum_{i_K=1}^{3} \phi_{i_K}^{i_K,R} b_{i_K}(\vec{r_p}). \tag{4.21}$$

However, even though it is not mentioned in their report, this is merely an approximation of the true value of the reduction term. To see this, remember that the reduction is done as Equation (4.16),

$$\phi^{corrected}(\vec{r_p}) = \sum_{i_k=1}^{3} \phi i_K b_{i_K}(\vec{r_p}) - \frac{q_p}{\epsilon} \sum_{i_K=1}^{3} \phi_{i_K}^{ref} b_{i_K}(\vec{r_p}), \tag{4.16}$$

where ϕ^{ref} is calculated from the particle's position. Because all the charge is given to only one node in $|\phi_i^{i,R}|$, it is strictly larger than $|\phi_i^{ref}|$, as long as the particle is not in a node position. Since the two equations (4.21) and (4.16) have the same terms in them, Equation (4.21) has to be an approximation at best. The approximation is that one treats the particle as if it was in the nodes themselves, and when the particle is close to a node, this will be a good approximation. However, it is only a good approximation as long as it is the potential which is in question. Since the weights are the bases, they become constants for the electric field, and the approximation does not take into account the distance to the nodes farther away. Fortunately, with some care taken, one can make a better approximation which captures the interaction between nodes within an element.

Here, the usage of the superpositioning principle and the usage of the bases will be done in great detail to develop a more suitable approximation. If one wants to represent the particle in an element using particles in the three nodes of said element, then the superposition principle says that the particle can be represented by using the distance-weighted particles' contributions, which can be written in terms of the bases as

$$\phi^{ref}(\vec{r}) = \sum_{j_K=1}^{3} \phi^{j_K,R}(\vec{r}) b_{j_K}(\vec{r_p}). \tag{4.22}$$

Equation (4.22) is the potential at any position $\vec{r}$ of a particle charge located at $\vec{r_p}$ in the domain in terms of the three reference charges located in r_{j_K} in the mesh using superpositioning. It is tempting to say that since the basis function is zero in all nodes except the one which the particle is placed in, the potential at the particle's position becomes

$$\phi^{ref}(\vec{r_p}) = \sum_{j_K=1}^{3} \phi_{j_K}^{j_K,R} b_{j_K}(\vec{r_p}), \tag{4.23}$$

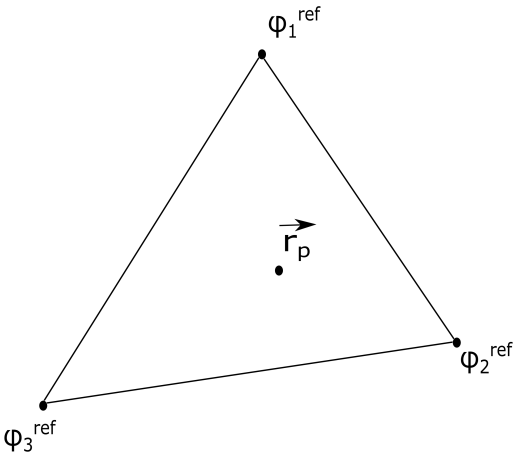
which is Kalna's approximation. However this assumption that the particle's location is in each node is untrue. The particle's location in Equation (4.22) is inside the element, where each of the three nodes has support. Since it is the response at the particle's position which obeys the superposition principle, and the basis is merely a representation of this response, the expression has to be made with respect to each basis. This can be done by using Equation (4.22) with $\vec{r} = \vec{r_{i_K}}$ in the elements nodes. Then the result of each basis contribution is that,

$$\phi_{i_K}^{ref} = \sum_{j_K=1}^{3} \phi_{i_K}^{j_K,R} b_{j_K}(\vec{r_p}). \tag{4.24}$$

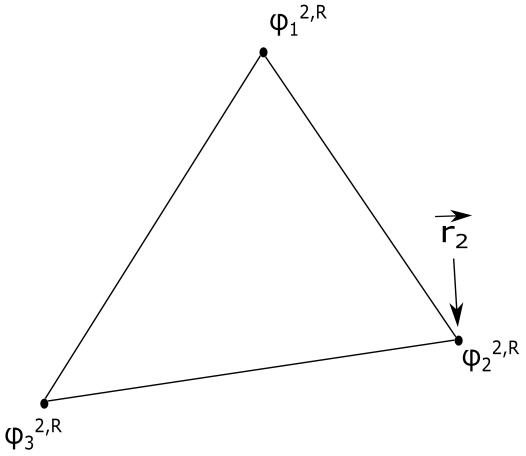
That is, each ϕ_i^{ref} is the distance-weighted average of $\phi_i^{;R}$ from the calculation of a particle in each node in this element. With ϕ_i^{ref} found, one can see that in the particle's position, the reference potential has to be

$$\phi^{ref}(\vec{r_p}) = \sum_{i_K=1}^{3} \sum_{j_K=1}^{3} (\phi_{i_K}^{j_K,R} b_{j_K}(\vec{r_p})) b_{i_K}(\vec{r_p}). \tag{4.25}$$

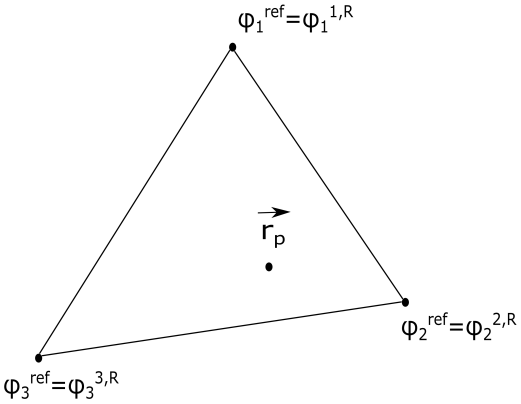
There is a test one can do on this equation. As stated, Kalna's approximation should be good enough when the particle is close to a node. This means that Equation (4.25) should be equal to Equation (4.23) for a particle in a node position. This is certainly true, take for instance a particle in node number 1, so that $b_j = 0$ for any $j \neq 1$



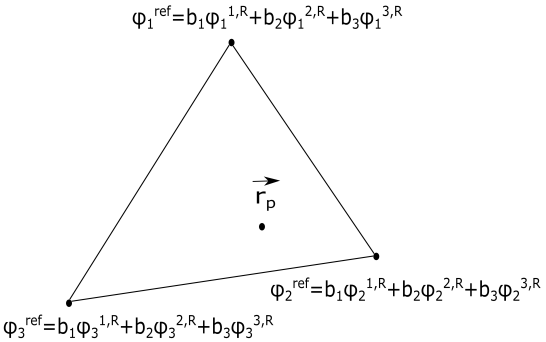
(a) Reference potential for a particle in an element



(b) Grid reference potential for an element where the particle is set in node 2.



(c) Kalna approximation reference potential for a particle in an element



(d) New approximation reference potential for a particle in an element

Figure 4.1: The illustrated usage of reference potential approximations.

$$\begin{aligned}
\phi^{ref}(\vec{r_1}) &= \sum_{i_K=1}^{3}\sum_{j_K=1}^{3}(\phi_{i_K}^{j_K,R} b_{j_K}(\vec{r_1}))b_{i_K}(\vec{r_1}) \\
&= \sum_{i_K=1}^{3}(\phi_{i_K}^{1,R}\cdot 1))b_{i_K}(\vec{r_1}) \\
&= \phi_1^{1,R},
\end{aligned}$$

which is the expected result. The reference potential for a particle in a node is equal to the reference potential calculated in that node, which is the same answer as Kalna's approach has.

Since the discussion above may be somewhat hard to understand, the methods are illustrated in Figure 4.1. Figure 4.1a illustrates the reference potential calculated from the particle's position, that is the reference potential talked about in Section

4.2. Figure 4.1b shows how the grid reference potential is calculated, in this illustration the potential is calculated as if a particle is positioned in grid node number 2. In figure 4.1c the way Kalna approximates the function from Figure 4.1a using the grid potential from figure 4.1b is shown. This is not to be mistaken to be the approximation to the reference potential situated at the particle's position; it is the approximation to the nodes in the grid when the particle is in $\vec{r_p}$. In Figure 4.1d the new approximation just presented is shown.

Now that the new method has been explained, it should be clear that the corrected potential is given as

$$\phi^{corrected} = \sum_{i_K=1}^{3} \phi_{i_K} b_{i_K}(\vec{r_p}) - \frac{q_p}{\epsilon} \sum_{i_K=1}^{3} \sum_{j_K=1}^{3} (\phi_{i_K}^{j_K,R} b_{j_K}(\vec{r_p})) b_{i_K}(\vec{r_p}) \tag{4.26}$$

Using this correction, the corrected electric field is given by

$$\vec{E} = -\sum_{i_K=1}^{3} \phi(\vec{r_{i_K}}) \nabla b_{i_K}(\vec{r_p}) + \frac{q_p}{\epsilon} \sum_{i_K=1}^{3} \sum_{j_K=1}^{3} (\phi_{j_K}^{i_K,R} b_{j_K}(\vec{r_p})) \nabla b_{i_K}(\vec{r_p}) \tag{4.27}$$

4.4 Illustrations of the reference potential

This section will show the results of the method. For this demonstration, a coarse grid has been employed to make it possible to see the effects and details. The grid employed has 417 nodes and 832 elements. It is employed on a rectangular domain where the length in x- and y-directions are respectively 3 µm and 1 µm, the characteristic length used for the generation of this mesh is $h = 0.1$ µm.

In Figure 4.2 a scaled reference potential is shown as a contour plot. This is the solution to Equation (4.3), $\phi^{i,R}$, where the particle is set in a node i in the grid, with the infinite boundary condition. That is, this is the Coulomb potential for the discretized mesh. The mesh is transparent in the figure to give some insight into why the equipotential lines are not perfect circles. It is important to understand that this is merely one of the calculations in the reference potentials. There is such a calculation for each interior grid node. The resulting electric field from this reference potential is shown in Figure 4.3. As can be seen, the electric field points outwards from the particle in all elements. The elements which are stored for usage in MCCT are the ones which the node is a part of, depicted in Figure 4.4.

Since the reference potential is not unique in each mesh node, the full reference potential is not shown. As mentioned before, this is because the reference potential stores each neighbor of its calculated node i. It needs to store several mesh nodes for each calculation, not just the one node i. Therefore, the best representation is the electric field coming from the reference potential. In figure 4.5 the reference electric field is shown for each element. This means that, if one wants to imagine what this

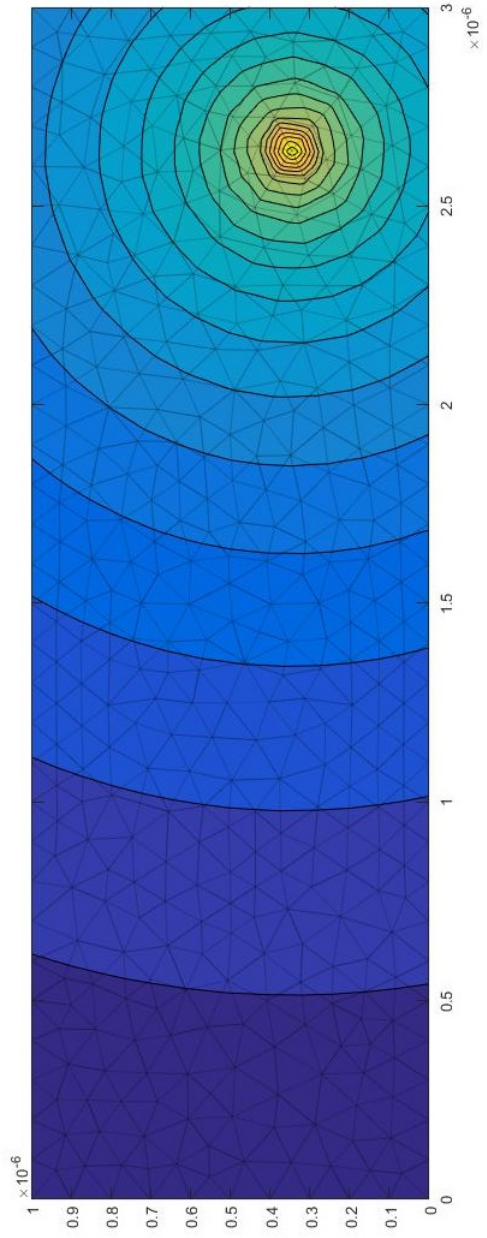


Figure 4.2: The approximated potential plot for $\phi^{p_{node},R}$ for node p_{node}.

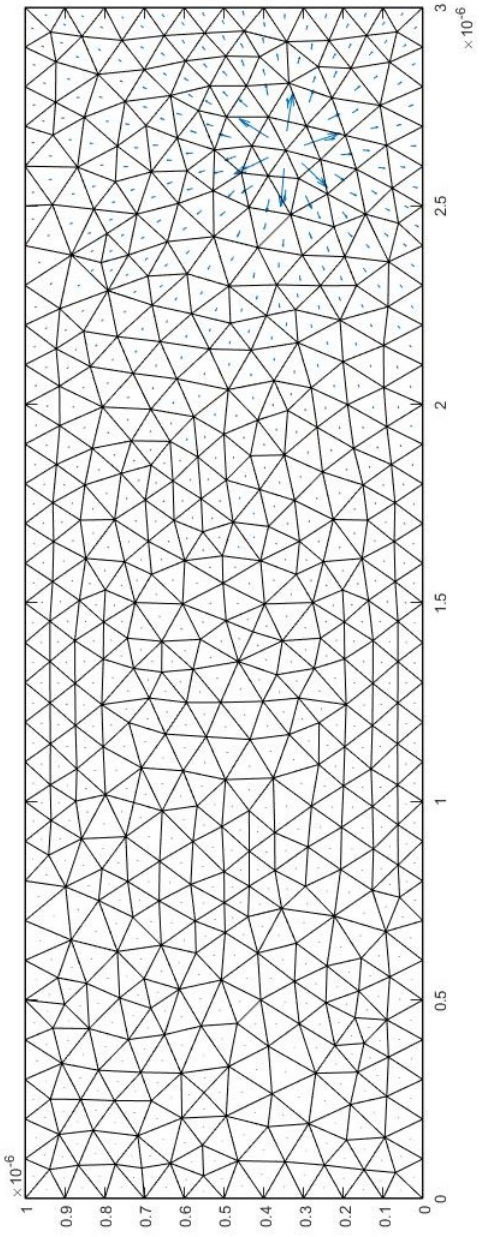


Figure 4.3: The approximated electric field coming from the potential $\phi^{p_{node},R}$ for node p_{node}.

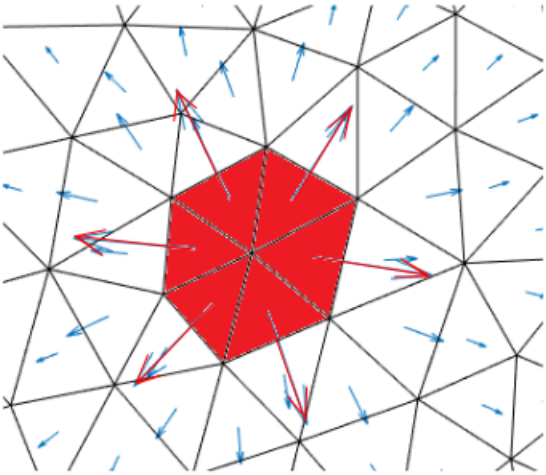


Figure 4.4: The electric field vectors used for self-force reduction.

figure shows, it is the electric field in each element if a hole was in that one element in an infinite domain without any other particles.

It may seem like, from the Figure 4.5 , that the electric field is many orders larger in elements at the boundary. However, this is a side effect of the fact that it is only possible to calculate the reference potentials in the interior nodes, and the actual self-force may not be as large. Therefore, these elements only have contributions from one or two nodes, which gives the large reference electric field here. Unfortunately, this means that these elements cannot be reduced with this method, at least not in a simple way. But one can see the tendency of the electric field pointing towards the nearest boundary when close to one boundary. But on the contrary, when not close to a boundary, the electric field does seem arbitrary and shows that finding an analytical expression for the self-force, or using statistical analysis for an approximation, may be to no avail. However, the most dangerous type of self-force are the systematic ones, and therefore, it might be enough in some circumstances only to get rid of these that have a tendency to move particles towards, or away from the boundary. But this is mere speculation, and the most rigorous method would be to remove the self-force with this reference electric field. It should be noted that the fields in the interior are barely visible, but they are present. The reason they are barely visible is that they are much smaller than the ones closer to the boundaries. However, they are still large, the 2 norm of the reference electric field varies from $632\,\mathrm{V\,m^{-1}}$ to $4312\,\mathrm{V\,m^{-1}}$.

Fortunately, the fact that reference potentials can't be calculated for elements with boundary nodes is not very impactful. In fact, the boundary conditions need to take priority over the reduction of self-forces. Since the boundary condition is Neumann zero condition, except for the contacts, they cannot be reduced in the same manner as other elements. That is because doing the reduction after calculating the potential with zero Neumann condition will introduce a nonzero electric field on the normal of the boundary. Therefore, if the self-force is to be reduced near boundaries, it should only be calculated parallel to the boundary. This is possible to do for elements which have only one node on the boundary, but not for the ones with two nodes

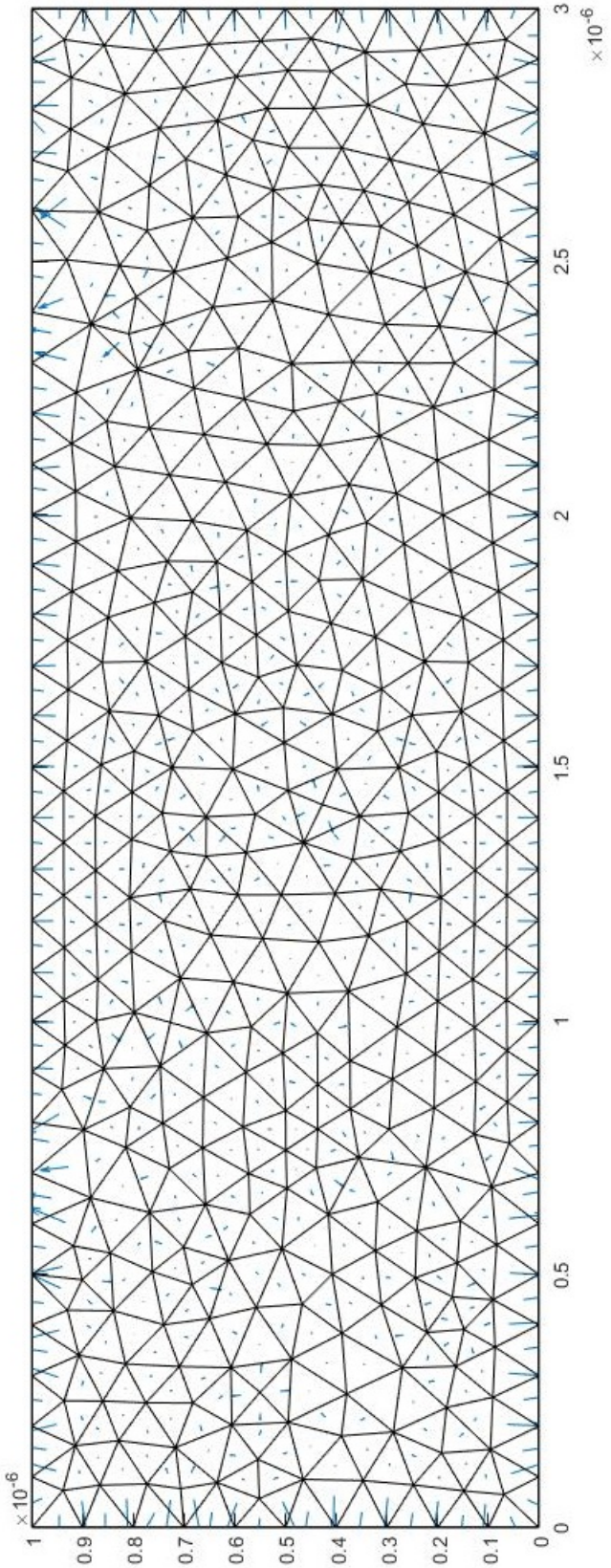


Figure 4.5: The resulting reference electric field calculated on each element $K \in \tau$.

on the boundary. The reason for the Neumann zero boundary condition is that the interesting results should be well into the domain, and the boundaries should have as small an influence as possible on these results. This has usually been achieved by making the electric field as small as possible. Mathematically, this is expressed as the Neumann zero boundary condition. A concern is whether or not the self-force will alter the electric field to be of importance. Looking at the component of $\vec{E}$ normal to the boundary, near the boundary, it is often said that this is the component which has the largest magnitude for the self-force, but in the case of Neumann zero boundaries, this component is essentially removed since the condition is to set it to zero. On the parallel component of $\vec{E}$, the self-force is usually much lower near the boundary. Since this is the case, although no self-force reduction is done near the boundary, the problem is handled by making sure that the Neumann zero condition is enforced. There are ways to try to remove the remaining self-force in the parallel direction. For the boundary elements with only one node on the boundary, the parallel self-force can be approximated from the potential in the two interior nodes. For the elements with two boundary nodes, the self-force must be approximated in a different way. For instance, the element-wise reference potential approach where the reference potential is calculated for each element is a possibility. Since the boundaries are employed on nodes, all of the elements, including boundary elements, can be reduced in this approach.

4.5 Self-force test

In this section, the self-force is tested. To do this, a superparticle is set in an infinite domain, just as with the reference potential. That is, the same equations are used as in the reference potential calculations. The difference is that the particle is not set in nodes, and therefore the calculation is done with the particle in some element, at some position $\vec{r_p}$ and without scaling. That is, here the equations

$$\nabla^2 \phi^{ref'} = -\frac{q_p}{\epsilon}\delta(\vec{r} - \vec{r_p}) \tag{4.28}$$

with dirichlet boundaries simulating an infinite domain as explained in 4.2,

$$\phi^{ref}|_{\partial\Omega_D} = \frac{-1}{2\pi}\ln(|\vec{r} - \vec{r_p}|), \tag{4.29}$$

are calculated using FEM as described so far. That is, firstly, the reference potentials are calculated; secondly, the particle's potential is calculated, and finally, the reduced potential and the electric field are calculated. The particle has been set in $x = 1.5$ and $y = 0.5$. The resulting potential from an electron superparticle is shown in Figure 4.6. From this figure, it is easy to spot why the self-force is such a big problem. Since the potential is so large at the particle's position, and quickly descends in its near vicinity, a very small error either in the particle's location or the potentials

peak will make it so that the particle essentially is in the near vicinity where the electric field becomes large. Since the grid is coarse, it illustrates the non-symmetry from the triangular elements. The calculation results in an electric field that is

$$\vec{E} = \begin{bmatrix} -175.9 \\ 1936 \end{bmatrix} \mathrm{V\,m^{-1}}. \tag{4.30}$$

The reference electric field is calculated as nearly identical with opposite direction and since the superparticle is an electron, the sign of the reference potential is taken into account so that the negative reference potential is subtracted to yield the reduced electric field:

$$\vec{E}^{reduced} = \begin{bmatrix} 1.282 \cdot 10^{-9} \\ 8.001 \cdot 10^{-8} \end{bmatrix} \mathrm{V\,m^{-1}}. \tag{4.31}$$

These vectors are shown in Figure 4.7. The 2-norm now gives a relative error of $4.1 \cdot 10^{-11}$ for the method. This number makes sense as the linear system solver used has been set to a tolerance of $1 \cdot 10^{-10}$. So the method is exact up to the tolerance of the solver.

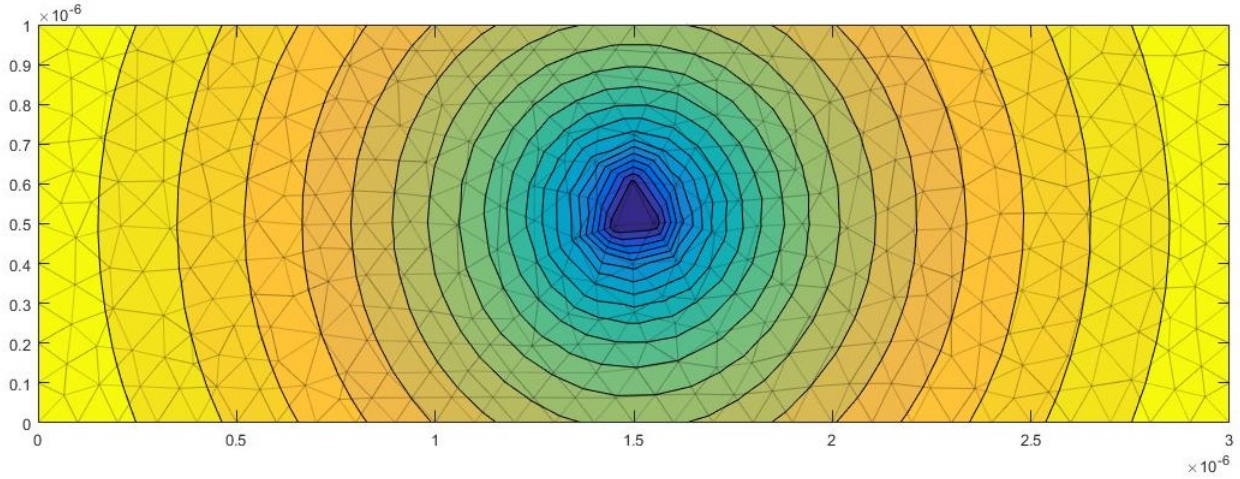


Figure 4.6: Approximated potential from a superparticle in an infinite domain.

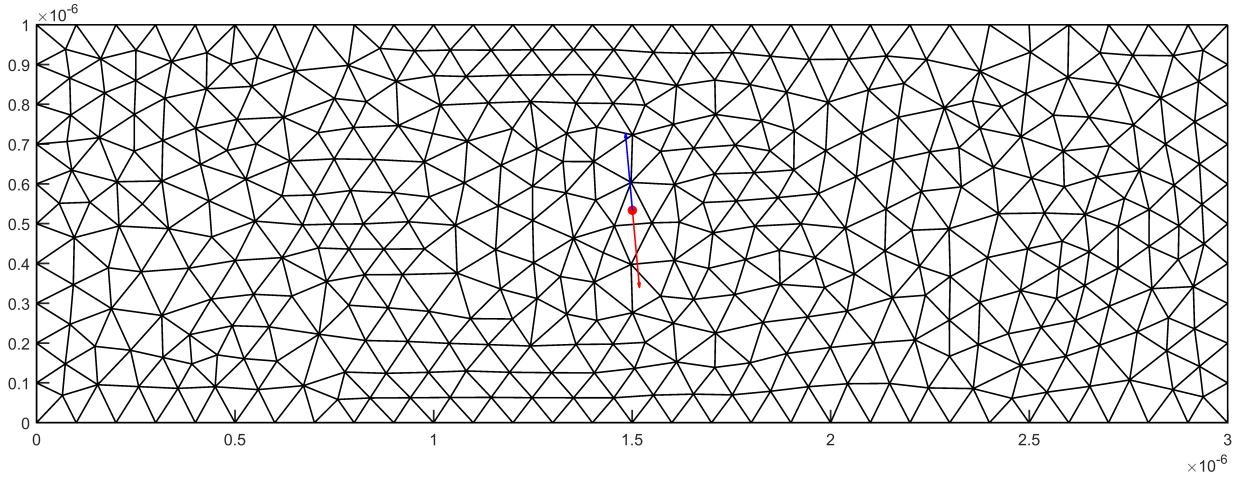


Figure 4.7: The electric field reduction, the blue vector denotes the electric field from the particle, the red vector denotes the reference electric field in the corresponding element.

Chapter 5

Particle locating

When there is a partitioning of a domain into subdomains, the problem of finding which subdomain an arbitrary point in the domain is within, is what is called the point locating problem. Point location is a topic of computational geometry and has seen much research from back in the late 70s [24] until recent years. In the 80s studies of Voronoi diagrams led to advancements for Delaunay meshes, in particular, [25] was published by L. Guibas and J. Stolfi, which contained a robust algorithm for Delaunay meshes. In the 90s further steps were taken to make the method robust for non-Delaunay meshes in [26], this method is built on Guibas' algorithm and patched the algorithm by inserting a distance to position measure which is forced to be strictly decreasing. In the start of the 21st century point location for particles was developed further [27][28]. In particular, a particle tracking algorithm able to work well with particle boundary conditions was published in [29]. In this master's thesis, the choice has fallen on using Guibas' and Stolfi's algorithm as it suits the purpose and the Delaunay property is satisfied for the mesh.

In this chapter, the particle locating problem will be discussed. In Section 5.1 the problem is defined and possible algorithms are discussed. In Section 5.2 the chosen algorithm is presented. In Section 5.3 the structures needed for an efficient implementation is discussed. In Section 5.4 the implementation is altered for speed. In Section 5.5 the run time of the algorithm is tested.

5.1 General problem description

The general problem to find which element a particle resides within, can be stated as follows: given a coordinate (x, y) in the domain Ω find which element $K \in \tau$ it lies within. See figure 5.1 for an illustration. For a rectangular grid, this problem is trivial since one can simply calculate which four nodes are closest to the coordinate, immediately giving the location in the grid. However for a general mesh, the problem can be hard. Several algorithms such as [30, 31] exists. The choice is dependent upon what kind of mesh has been generated. Here, the choice will be dependent

on the Delaunay property of the grid. However, if another type of grid is generated in the future, there exist methods that can deal with non-Delaunay meshes, for instance, [26].

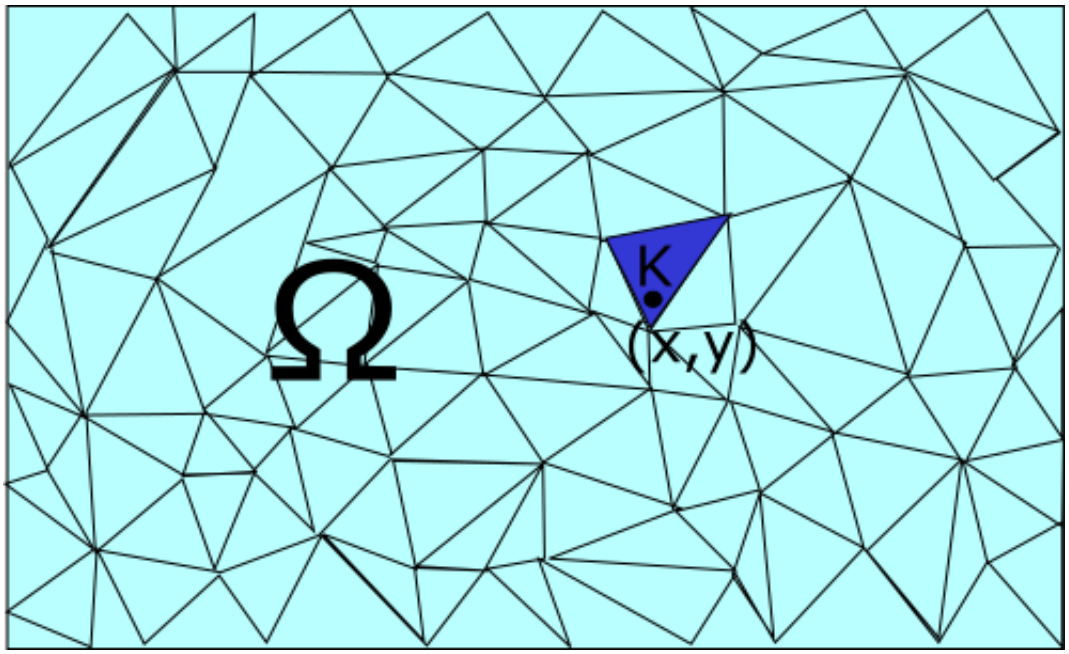


Figure 5.1: An illustration of a point (x, y) in a domain Ω with a triangulation τ, where $K \in \tau$ is the element containing the point (x, y).

There is an algorithm made in particular for particle-in-cell (PIC) methods [29]. This algorithm should be employed when the mesh is unstructured and used in combination with tracking of particles since it assumes that the particle's position was known at the previous time step. A vector is made from the previous to the new location. This vector will intersect all faces in the shortest route to the new location. Then, the length of this vector and its intersections with faces decides which element the point is within. If hybrid elements in 3D are used [32] is a valid choice. This method stores triangular faces of all elements. While searching through elements, the triangular faces are used to combine existing methods for different element types to find the points location in the mesh. Here the focus is only on triangular meshes since that is what has been used in the finite element method, and therefore simpler methods can be employed.

Even for triangular meshes, there are many algorithms that can be chosen. However, since these algorithms are worthy a study of their own, a simple algorithm, which does well enough has been implemented. Even so, there are criteria for the algorithm. The criteria are:

- Take into account previous particle location.
- Take into account that the mesh is Delaunay.
- Be able to detect particle positions outside of the domain.

The first and second criteria are advantages of the mesh, which should be exploited for speed or rigor. By taking into account the previous particle location, with a small time step, the new location should be in an element close to the previous one, and therefore the algorithm should only need to check elements close to the previous location. The Delaunay property is a favorable property for FEM, and in general, for cases of geometrical computations, it is a property that should be exploited. The last criterion, being able to detect positions being outside of the domain, is required to make the particle boundary conditions implementable on an arbitrary domain. It is sufficient to be able to detect that it is outside, and not knowing the closest element to the particle. However, it is positive if the algorithm can identify which boundary edge the particle crossed to get outside of the domain.

5.2 Guibas' and Stolfi's Point Location Algorithm

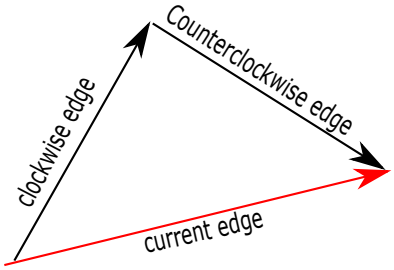


Figure 5.2: An example element with directed edges.

Since Guibas' and Stolfi's Point Location Algorithm (PLA) [25] is designed to locate an element containing a coordinate in a Delaunay triangulation, it is a perfect match for the mesh generated by GMSH. It has been chosen for its simplicity both in implementation and concept and also for being in agreement with the three criteria laid down in the previous section. Guibas' and Stolfi's Point Location Algorithm, shown in Algorithm 2, starts at some random edge, then traverses the mesh in the general direction of the particle to be located. This is done by letting each edge be a directed edge, keeping the particle on the left of its directed edge at all times. Then it will arrive at the correct element K when no such edge exists. The directions for each edge is shown in Figure 5.2. The illustration shows that if the particle to be found is contained in the element only the current edge have the particle to its left. Notice that since counterclockwise rotation is checked first, the algorithm has a bias for searching in a counterclockwise direction, which implies that it does not necessarily take the shortest path to the correct element. Therefore, there is potential to speed up the algorithm by alternating which edge is checked first. This has not been applied since when the particles travel a short distance, the algorithm will approximately take the shortest path. An example run through a triangulation from an arbitrary edge is shown in Figure 5.3. This figure also illustrates that the algorithm has a bias for searching in a counterclockwise direction.

To determine the side of a directed edge, denote $\vec{e}_p$ as the vector between the origin of the current edge and the particle p to be located in space and $\vec{e}$ as the current edge vector. Then the side of $\vec{e}$ which the particle lies on is shown by the sign of

Algorithm 2 Particle locating in a Delaunay triangulation

input $\vec{r_p}$ = coordinate of particle to find location of. τ = triangulation to find particle in.
ouput K = element which point is within.
set K = element with edge e
if $\vec{r_p}$ is on the right of the current edge **then**
    Edge e direction is inverted.
end if
while element not found **do**
    if $\vec{r_p}$ = one of the endpoints of edge e **then**
        $e = e$ is the found edge with K as the found element.
    else
        if $\vec{r_p}$ is to the left of counterclockwise edge **then**
            e = counterclockwise edge, update K to opposite element of e
        else
            if $\vec{r_p}$ is to the left of clockwise **then**
                e = clockwise edge, update K to opposite element of e
            else
                e is the found edge with K as the found element.
            end if
        end if
    end if
end while

$$|S| = |\begin{bmatrix}\vec{e} & \vec{e_p}\end{bmatrix}|, \tag{5.1}$$

where $|\cdot|$ denotes the determinant. The change of sign on the left or right of a directed edge follows from the interpretation of the determinant as a dot product. It should be noted that point location algorithms often have issues related to using finite precision to calculate geometrical concepts. This issue is discussed in [29]. In this algorithm, the issue shows up in Equation (5.1). That is, because of finite precision, this algorithm may in some instances calculate the point in space to be on the right or left side of a directed edge when in reality it is on the opposite side. With one exception, this has not had any effect on the results as far as what the author has noticed. The exception is that in some cases the algorithm concludes that a particle may be outside of the grid when it is not. However, the algorithm has been implemented with this in mind. Whenever it concludes that a particle is outside the grid, it will stay in the same element but have the boundary edge as its current edge. If the particle is inside this element, it will not find any new edge or element. If it is in another element, it will rotate the edge correspondingly. Since the new iteration will have redefined what is clockrotate and counterclockrotate, the next test will decide whether or not the particle is in this element or not. This means that some care has to be taken to be able to find out whether or not a particle is about to cross the boundary of the domain. The algorithm has to be allowed to find an element which it believes the particle is within, and if this element is on the boundary, a particle-in-cell test must be used to check if it is actually outside the

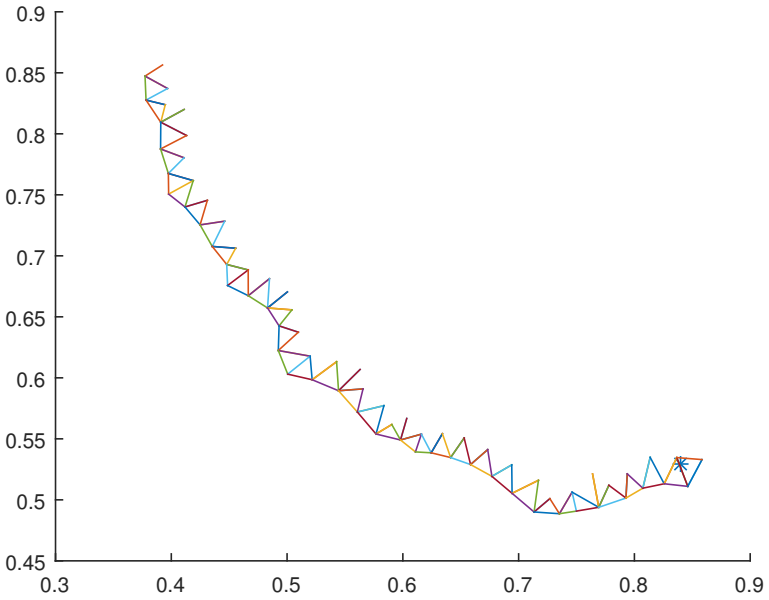


Figure 5.3: Example of a path of the particle location algorithm. Starting at a random edge, finding the element containing the particle in $(0.84, 0.54)$.

domain. Therefore, this algorithm can find particles that are outside the domain, but, unfortunately, it is inefficient.

5.3 Particle locating structures

The output from GMSH is two lists, or functions, which relate the elements to the nodes, and the nodes to the x- and y-coordinates in the domain. By defining the sets $\mathbb{N}_{nodes}$ and τ, as respectively the set of nodes and elements, the output of GMSH can be represented as the two functions

$$N : \mathbb{N}_{nodes} \to \mathbb{R}^2 \tag{5.2}$$

and

$$E_{nodes} : \tau \to \mathbb{N}^3_{nodes}. \tag{5.3}$$

That is, E_{nodes} is a list of which nodes each element consist of, and N is a list of the node coordinates. However, it is apparent from Algorithm 2 that the edges are important. Edges between nodes can, in theory, be found using only E_{nodes} by searching for elements sharing the nodes which are part of an edge, but this search will make the algorithm slower than it needs to be. Therefore, some more mappings

are constructed during initialization to resolve these speed issues. The lists needed are:

1. A mapping telling which two elements share an edge.
2. A mapping telling which two nodes make an edge.
3. A mapping which says which three edges make an element.

By defining that γ is the set of edges,

$$e_{elements} : \gamma \to \tau^2, \tag{5.4}$$

denotes which two elements share an edge. In the case of boundary elements, the edges at the boundary are seen as 1-dimensional elements and are therefore included in this list as both edges and elements. This is handy to be able to identify boundary edges.

$$e_{nodes} : \gamma \to \mathbb{N}^2_{nodes}, \tag{5.5}$$

is a list denoting which edge consist of which two nodes.

$$E_{edges} : \tau \to \gamma^3, \tag{5.6}$$

is a list denoting which elements consists of which edges.

Using the three new lists, the operation of finding opposite element can be reduced in complexity. Denote e^i_{nodes} as the ith output of e_{nodes}. Then the opposite element, given a current edge $e \in \gamma$, can be stated as

$$\begin{aligned} K_{oppositeelement} &= e^j_{elements}(e),\ j = 1, 2 \text{ such that,} \\ e^j_{elements}(e) &\neq K_{currentelement}. \end{aligned} \tag{5.7}$$

The reason e_{nodes} is needed, is because it defines the edges. The directed edges are found in E_{nodes}, then the edge number e can be found in the list e_{nodes} by comparing with E_{nodes}. Notice this approach only needs to check the neighboring edges, which gives speedup from the naive approach of matching elements by searching in E_{nodes}.

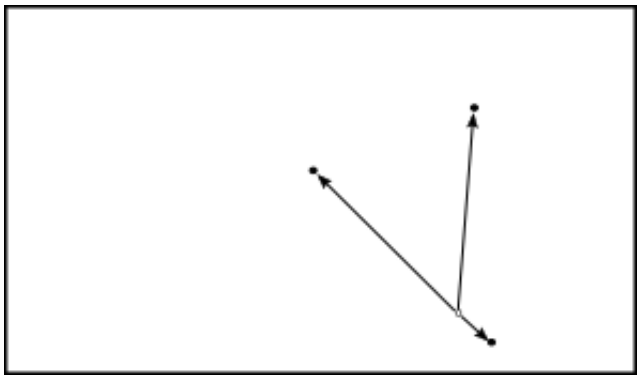


Figure 5.4: Naive approach of initialization

5.4 Reducing search time

In MCCT all the particles need to be updated once for each iteration. Each particles search is proportional to the number of elements the algorithm needs to pass through to get to the correct element. Therefore, the algorithm starts from the previous location to reduce the length from the initial element and the particle's element. With this initial element, the time spent in the particle location algorithm is obviously close to proportional to the length the particle traveled and inversely proportional to the size of each mesh element. That is, for $n(P)$ number of particles,

$$T_{searchtime} \propto n(P) \cdot \frac{l}{h}, \tag{5.8}$$

where h is an average element size. This equation is only true for the ensemble of particle searches. It is not true for only one particle since h is a step length on average, and the particle may be going through elements larger, or smaller, than the average element. Equations (2.3) and (5.8) yields that

$$T_{searchtime} \propto n(P) \cdot \frac{\Delta t}{h}. \tag{5.9}$$

From section 2.3 it is assumed that a finer grid needs a shorter time step, but since the triangulation is not structured, another limitation is put on the time step from Equation (5.9). This limitation is of a kind which has more to do with practicality rather than stability or other issues. A large Δt will make the algorithm slow, which can in some instances be an issue.

It is worth mentioning that since this search is done for each particle, the algorithm is parallelizable by giving particles to different processes. However, load balance is an issue since it is unknown prior to the search which searches will take a long time. Considering the number of particles and that they have an equal time step, this may not be an issue. If it is an issue, it can be resolved by using distance traveled as an estimate for load balance. Note that this is only an approximation since the number

of elements traversed is not necessarily proportional to the distance traveled for a particle in an unstructured grid.

At initialization the particles' positions are randomly generated and therefore the first search will take extensive time. However, it can be sped up by making some considerations. The naive approach is to make all the particles search from an arbitrary element which is inside the domain. This could take a very long time since most particles will be far away from this element. A smarter way to initialize is to distribute some ghost particles that are imagined on the grid and use the particle location algorithm on these ghost particles. Then for each superparticle, check which ghost particle is the closest and start the algorithm from this ghost particle's element. With this idea in mind, each device can be initialized efficiently depending on where particles are likely to be. For a PN-junction, which will be presented in Chapter 7, a simple approach has been used. The approach is to use 4 ghost nodes which are in the center of each of the regions P^+, N^+, P, N. This is done in this way since the centers for P and N are central for most particles, and because the regions P^+, and N^+ are initialized with more particles than P and N regions. An illustration of this strategy is shown in Figure 5.5c.

The idea of ghost particles extends beyond just initialization. The concept can be used to get speedup for large time steps. Usually this is not needed, but in some cases it might be useful not to be restricted to a small time step, and therefore another approach has been implemented. The idea is to use a domain filling strategy using a Cartesian coordinate system. By inserting the ghost particles at each node in the Cartesian coordinate system, the domain can be filled to the extent where, often, the closest particle will be in the same element as the actual superparticle. This means that the search time can, in theory, be reduced to be close to some chosen constant that must be larger than in the case of all the particles being in their initial element. Some step size is chosen low enough to fill the domain sufficiently. Then the set of ghost nodes is instead viewed as a rectangular grid in the xy- plane. For each particle, the nearest grid point in this grid is used as the initial search element. Using

$$(i, j) = ([\frac{x}{h_{gn}}], [\frac{y}{h_{gn}}]), \tag{5.10}$$

where h_{gn} is the step size between ghost nodes, a look-up table gives the approximated nearest element in this grid. For a given h_{gn} this method is linear with respect to the number of elements. However, h_{gn} can always be decreased until near constant search time is achieved. This can require a huge amount of memory, but only integers need to be stored, so this is efficient and can be of help in some cases. This method is depicted in Figure 5.5b. Although the figures shown in this section are only for rectangular domains, the approach is just as useful within an arbitrary domain. This is because any arbitrary domain can be put into a rectangle of a larger area. It just means that these domains will have ghost nodes for which there are no elements. However, there can be trouble with the size the grid would need, and in that regard, the amount of memory needed for an arbitrary domain. It should be noted that the implementation in MCCT, when using the domain-fill method, uses

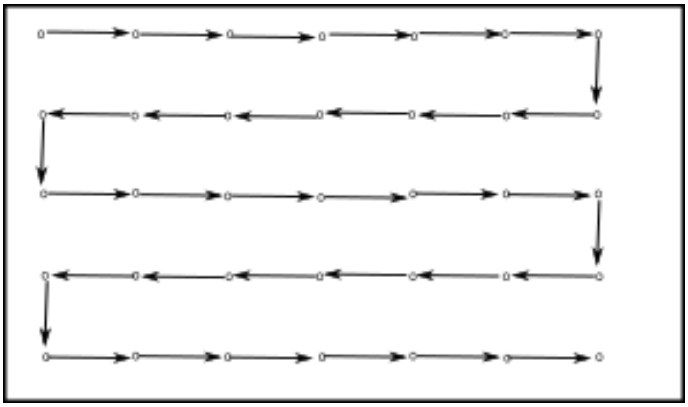
the following relation to choose the initial element:

$$\begin{aligned} \vec{r_{gn}} = (x_{gn}, y_{gn}) &= (i \cdot h_{gn}, j \cdot h_{gn}) \\ K_{initial} &= \begin{cases} K_{previous} & \text{if } ||\vec{r_p} - \vec{r}_{prev}|| < ||\vec{r}_{gn} - \vec{r}_{prev}||, \\ K_{gn} & \text{if } ||\vec{r_p} - \vec{r}_{prev}|| \geq ||\vec{r}_{gn} - \vec{r}_{prev}||, \end{cases} \end{aligned} \tag{5.11}$$

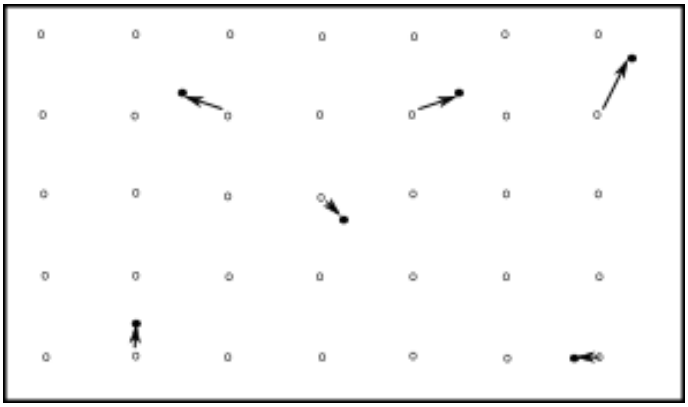
where $K_{initial}$, K_{gn}, K_{prev}, denotes the initial element to search from in this iteration, the ghost node's element, and the particle's previous element respectively. The same notation is used for the particles' positions where $\vec{r_p}$ is the new particle position. Equation (5.11) describes that the new initial search element is either the previous location or the closest grid node depending on which of these two is closest to the new position. This is to ensure that a too coarse ghost grid will not make the algorithm slower. For fine grids, relative to the distance traveled by the particle, this check can be neglected. It is tempting to fill the space and never use the search algorithm. However, this is not recommended since as long as h_{coord} is finitely small, there is a chance that a superparticle p will be put into a neighboring element. This approach is only used as an initial guess and will always need a point location algorithm to find the real element.

The ghost particles' elements are found in an orderly fashion by making sure the distance between successive ghost particles is close, as illustrated in Figure 5.5a. Each search is iterated onwards from the previous search. This approach is linear in search time since more ghost particles mean less distance between each and the total number of edges traversed is approximately constant. That is, up until the number of ghost particles exceeds the number of elements in the grid by a large amount. When this happens, the time spent making the grid becomes dominated by the time it takes to check if the ghost particle is contained in the starting element. Therefore, with such a high degree of resolution, the initialization time of the ghost grid is proportional to the number of ghost particles.

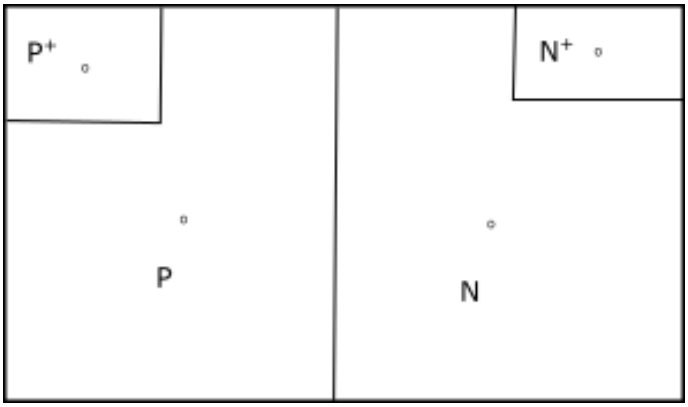
The domain filling method resembles what is presented in [33]. In that article, it is shown that this kind of search can give great speedups, and gives a reason to suspect the method presented herein also performs well. The difference between the approach here and the one given in the article is that the grid on top is not used as an initial guess for another algorithm, but rather checks which element the particle is contained in by iterating over each element with a subset of an element inside the Cartesian coordinate location, using a partial volume based method, or outer normal vector based method, both described in the paper. Depending on in what order these tests are carried out, it is possible that the method in the article outperforms the method described here, or the other way around.



(a) The initialization of the domain-fill strategy.



(b) The usage of the domain fill strategy.



(c) The initialization of a $P+, P, N, N+$ device.

Figure 5.5: Illustrations of strategies using ghost nodes. The circles are ghost nodes, the black dots are superparticles, and the arrows are approximate search direction when using the point location algorithm.

5.5 Results

In this section, run times of the algorithm are presented. The domain is a rectangle with sides $L_x = 3\,\mu m$ and $L_y = 1\,\mu m$. The characteristic length in the area where this test takes place is close to 0.016µm and the total number of elements is 203872. Further details of the mesh are not important, but can be found in Section 7.3. The mesh is the same as will be used in Chapter 7.

For this test, a particle is set in position $(x, y) = (0.1, 0.1)\mu m$ inside a known element. Then the particle travels to a new location, where the current element is unknown. The particle-location algorithm is employed to find the new element. Since an MCCT run consists of typically $100,000$ particles, the process is repeated $100,000$ times to show the time spent for a typical run. Since the mesh is unstructured, the paths taken are not the same for different lengths. This means that there is some additional variation in the time taken. To minimize this fact, the directions to the end locations are the same. In fact, the particle positions are all set on the line $y = x$.

In Table 5.1 the end locations, length traveled, and the time spent in the particle locating algorithm is shown. The number of lengths tested is not enough to check Equation (5.8). A detailed study could be performed to test the relationship, but the relationship will vary much depending on the mesh, and the test case, and therefore such a study has not been done. It is important to verify, that the time step chosen dictates the time spent in the algorithm. This is clearly seen from Table 5.1, as a shorter travel distance gives a shorter time spent in the point location algorithm. Analyzing the data given, one can see that the time spent in PLA is reduced faster than linear with respect to the distance traveled. This behavior is explained by the fact that the particle location algorithm does not take the shortest path to the destination. It has a large influence on the path taken and the number of elements traversed when traveling a long distance.

l(µm)	t (s)	x(µm)	y(µm)
0.99	2.44	0.8	0.8
0.14	0.27	0.2	0.2
0.02	0.14	0.12	0.12
0.0014	0.016	0.101	0.101

Table 5.1: Run times for the particle location algorithm. The particle starts in $(x, y) = (0.1, 0.1)\mu m$ and ends in the positions listed in columns 3 and 4. The traveled distance is listed in the first column, and the time spent in PLA is listed in the second column.

The improved version of the point-location algorithm is tested without varying the position of the destination. The destination is set to $(x, y) = (0.8, 0.8)\mu m$ and the distance h_{gn} between ghost nodes is varied instead. To fill the domain sufficiently, a good estimate of h_{gn} is needed. In this particular case, the used estimate is:

$$h_{gn} = [\frac{L_y}{\sqrt{N_\tau}}] \tag{5.12}$$

where N_τ is the number of elements. This estimate treats the problem as if $L_x = L_y$, and the elements were inside this area. Since $L_x = 3 \cdot L_y$, this approach makes a much finer step size than the average step size between elements. With this approximation, and letting $N_y = \sqrt{N_\tau}$, $N_x = \frac{L_x}{h_{gn}}$ the different tests have been done by scaling the result. The result gained using this estimate is $N_y = 452$, $N_x = 1506$, and $h_{gn} = 1.99\text{e}{-9}\text{m}$. For MCCT, the most natural approach is to use the average distance travelled as an upper bound for h_{gn}, then use a stricter bound than this value to get speedups.

As can be seen in Table 5.2, this estimate was a success since further dividing the domain into more ghost nodes does not give any speedup from the estimate. It looks like the estimate is rough and the result can be gained with fewer ghost nodes, but as should be made clear, this is a method which needs to be tested over time to see what works for a problem. In any case, if there is an issue with the estimate not being good enough, one can make a finer ghost mesh. The goal of this test is to show that the method can give the desired result of being fast when the distance traveled is large. Even for a very coarse ghost grid with $N_x = 376$ and $N_y = 113$, one can see that the time has been reduced from 2.44s to 1.77s. But these results are, as stated, not to be taken too literal. It should be stated that 0.008s is the time it takes when the initial guess of PLA is the same as the end destination.

N_x	N_y	h_{gn}(µm)	t (s)
3013	904	$9.4 \cdot 10^{-4}$	0.008s
1506	452	$1.99 \cdot 10^{-3}$	0.008s
753	226	$4.01 \cdot 10^{-3}$	0.008s
376	113	$8.02 \cdot 10^{-3}$	1.772s
186	56	$1.6 \cdot 10^{-3}$	5.23s

Table 5.2: Run times for particle location algorithm using ghost nodes.

Memory consumption has been stated as a concern for the method. For Table 5.2 the worst memory consumption is very low. 4-byte integers are sufficient to represent the mesh. Calculating the storage needed then yields that 11MB was enough for the finest grid tested in this case. Therefore, at this time, this is not a concern.

Chapter 6

Preconditioned Conjugate Gradient Method

A linear system solver is needed to calculate both the reference potentials and the potential in MCCT. Since the linear systems are large in these calculations, exact solvers are unsuitable. Exact solvers spend a large amount of memory to obtain the solution, especially for large matrices the memory requirements grow substantially. These arguments and the fact that iterative solvers have an initial guess, which can be close to the new solution, are the reasons that an iterative solver is implemented.

There exists a vast amount of iterative solvers that may be applied to the problem, knowing which one is most suitable is hard. However, an indication is that a specialized solver is more efficient in obtaining a solution. Therefore, a solver should be based on the properties of the system. In this case, the matrix is sparse, symmetric, and positive definite. With these properties, the obvious choice is the Conjugate gradient method, which was first published in [34] by Hestenes and Stiefel.

In addition to having a solver that works, it needs to be quick. Any preparations that can be done to speed up the solver can have a large impact, traditionally done with the use of preconditioners. Preconditioners alter the system to make its properties more favorable for convergence. They can be expensive to calculate, but are favorable in circumstances where the same matrix is used for several calculations, as in MCCT. Included in these convergence properties are not just speed of convergence, but also stability. A preconditioner is often needed for the iterative solver to converge at all.

The preconditioned Conjugate Gradient method is chosen to solve the linear system arising from the problem. This choice is a consequence of the fact that the linear system is symmetric and positive definite. For such systems, this method is among the most tested and stable algorithms that exist. The method of conjugate gradients is an exact solver which uses a finite number of iterations to find the solution. However in practice, it is used as an iterative solver by stopping the algorithm after reaching the desired tolerance.

This chapter will introduce the preconditioned Conjugate Gradient Method. In Section 6.1, the theory of standard Conjugate Gradient method is introduced. In Section 6.2, the preconditioned version is presented. In Section 6.3, the choice of preconditioner class is discussed. In Section 6.4 the first preconditioner is shown, and in Section 6.5 , the second preconditioner is shown. Section 6.6 shows the convergence and efficiency of the method. The notation in this chapter differs from previous chapters because of lack of new intuitive characters to employ, so the notation is self-contained in this chapter.

6.1 The Conjugate Gradient method

The goal of CG is to solve the linear system,

$$A\vec{x} = \vec{b}. \tag{6.1}$$

$A \in \mathbb{R}^{n\times n}$ needs to be a symmetric positive definite matrix, and therefore, equation (6.1) is equivalent to minimizing

$$\phi(\vec{x}) = \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x}, \tag{6.2}$$

which has the property that the residual is equal to the gradient:

$$r(\vec{x}) \equiv A\vec{x} - \vec{b} = \nabla\phi. \tag{6.3}$$

CG belongs to the class of Conjugate direction methods. The idea behind these methods is to generate a set of vectors which are conjugate and utilize this property to make an iterative method.

Definition 2. *A set of vectors* $\{\vec{p}_0, \vec{p}_1, ..., \vec{p}_n\}$ *is conjugate with respect to the matrix* A *if and only if*

$$\vec{p}_i^T A\vec{p}_j = 0, \forall i \neq j.$$

Since A is symmetric positive definite, and therefore nondegenerate, Definition 2 implies that conjugate vectors are linearly independent, which means that $\phi(\vec{x})$ can be minimized in n steps by minimizing $\phi(\vec{x})$ along $\vec{p}_i$. This can be stated as, minimize $\phi(\vec{x})$ along the n directions

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k\vec{p}_k. \tag{6.4}$$

The directional minimizing α can be found by inserting Equation (6.4) in Equation (6.2), explicitly calculated as

$$\phi(\vec{x} + \alpha\vec{p_k}) = \frac{1}{2}(\vec{x} + \alpha\vec{p_k})^T A(\vec{x} + \alpha\vec{p_k}) - \vec{b}^T(\vec{x} + \alpha_k p_k), \tag{6.5}$$

$$= \frac{1}{2}\vec{x}^T A\vec{x} - \vec{b}^T\vec{x} + \frac{1}{2}\alpha^2(\vec{p_k})^T A(\vec{p_k}) + (\alpha\vec{p_k})^T A(\vec{x_k}) - \alpha_k\vec{b}^T(p_k). \tag{6.6}$$

Then differentiating with respect to the scalar α yields

$$\frac{d\phi}{d\alpha} = \frac{1}{2}2\alpha(\vec{p_k})^T A(\vec{p_k}) + (\vec{p_k})^T A(\vec{x_k}) - \vec{b}^T(p_k), \tag{6.7}$$

which in turn, by setting the derivative to zero, gives the minimizer α along the line $x_k + \alpha_k\vec{p_k}$ as

$$\alpha_k = \frac{-r_k^T p_k}{p_k^T A p_k}, \tag{6.8}$$

where the residual $r_k = Ax_k - b$ has been inserted into the equation.

The main selling point of the Conjugate Gradient method is that it can generate a set of conjugate vectors in succession using only the previous vector, instead of the whole sequence of conjugate vectors. That is p_k can be generated from p_{k-1}. As explained above, using successive one-dimensional minimizations for each direction yields the final result in n iterations. It is important to note that, in practice, computers carry round-off errors which make the vectors not truly conjugate and therefore CG may never terminate as an exact solution.

The idea for generation of $\vec{p_k}$ is to start off with the steepest descent direction for $\vec{p_0}$, then add a term which will make the new direction conjugate with respect to the old direction, that is by setting

$$p_k = -r_k + \beta_k p_{k-1}, \tag{6.9}$$

where β is chosen such that p_k becomes conjugate with respect to A. That is by multiplying Equation (6.9) with $p_{k-1}^T A$,

$$p_{k-1}^T A p_k = -p_{k-1}^T A r_k + p_{k-1}^T A\beta_k p_{k-1}, \tag{6.10}$$

the left-hand side must be zero because of conjugacy, then solving for β yields that

$$\beta_k = \frac{r_k^T A p_{k-1}}{p_{k-1}^T A p_{k-1}} \tag{6.11}$$

At this point, the complete conjugate gradients algorithm has been derived, and it is stated in the implemented form in Algorithm 3.

Algorithm 3 Conjugate Gradient

$r_0 = b - Ax_0$
for $i = 1, 2, 3, ..., n$ **do**
  $\rho_i = r_{i-1}^T r_{i-1}$
  if i=1 **then**
    $p_1 = r_0$
  else
    $\beta_i = \frac{\rho_i}{\rho_{i-1}}$
    $p_i = z_i + \beta_i p_{i-1}$
  end if
  $q_i = Ap_i$
  $\alpha_i = \frac{\rho_i}{p_i^T q_i}$
  $x_i = x_{i-1} + \alpha_i p_i$
  $r_i = r_{i-1} - \alpha_i q_i$
end for

The algorithm described iterates over n directions to find the exact solution but is now altered to stop further calculations when the residual r_i is below some tolerance to make it an iterative method. When doing this, the following error estimate is of interest.This error estimate is well-known for CG and can be found in for instance [35].

$$\frac{||x_k - x^*||_A}{||x_0 - x^*||_A} \leq 2 \cdot \left(\frac{\sqrt{\kappa}(A) - 1}{\sqrt{\kappa}(A) + 1} \right)^k, \tag{6.12}$$

Equation (6.12) can be used to estimate the convergence of the method as a numerical method and implies that the conditioning number of A, $\kappa(A)$ dictates the convergence of the solver. In particular, $\kappa(A) = 1$ gives the fastest possible convergence. It makes sense to alter the conditioning number for a more favorable convergence rate, and this is what the preconditioners do. This is verified in many papers and the convergence of CG and preconditioned CG has been well-studied [36][37][38].

6.2 Preconditioned Conjugate Gradient

The Conjugate Gradient method does not converge exceptionally fast, however, coupled with a preconditioner it has proven itself to be useful for many problems [39][40]. When a preconditioner is used, it dictates most of the algorithms properties, the convergence rate and stability. The idea behind a preconditioner is to alter the linear system with some matrix M by multiplying the linear system with M^{-1}. The resulting linear system is

$$M^{-1}Ax = M^{-1}b, \tag{6.13}$$

where $M = LU$, and L and U are easily solvable matrices. From [41] it is known that this has the same convergence properties as multiplying by the left- and right-hand side as $U^{-1}AL^{-1}$ for symmetric positive definite matrices. The convergence rate now depends on the condition number of $U^{-1}AL^{-1}$ instead of A, which means that the preconditioner can be chosen to alter the convergence rate. These matrix products are not explicitly calculated, as this would lead to an abundance of issues, but rather included by solving linear systems of the form $Uy = z$.

Algorithm 4 is the algorithm used in MCCT, which is the preconditioned conjugate gradient method (PCG). The algorithm takes as input the linear system A, the right-hand side b and the preconditioner in the form of an upper matrix U and a lower matrix L. These two matrices have to be solved once each for each iteration. The solutions of these linear systems are acquired through backward and forward substitutions. These are exact quick solvers which the interested reader can find out more about in Appendix A.3.

Algorithm 4 Preconditioned Conjugate Gradient

$r_0 = b - Ax_0$
tolerance= $tol \cdot norm(b)$
for $i = 1, 2, 3, ...,$ max iterations **do**
  solve $Ly_i = r_{i-1}$
  solve $Uz_i = y_i$
  $\rho_i = r_{i-1}^T z_i$
  if i=1 **then**
    $p_1 = z_1$
  else
    $\beta_i = \frac{\rho_i}{\rho_{i-1}}$
    $p_i = z_i + \beta_i p_{i-1}$
  end if
  $\alpha_i = \frac{\rho_i}{p_i^T A p_i}$
  $x_i = x_{i-1} + \alpha_i p_i$
  $r_i = r_{i-1} - \alpha_i q_i$
  if $norm(r_i)$ <tolerance **then**
    answer found, exit loop
  end if
end for

The stopping criterion is that the 2−norm of the residual needs to be less than the scaled tolerance, which is scaled by multiplying with the 2−norm of the right-hand side. That is

$$\frac{||r_i||_2}{||b||_2} < \tau. \tag{6.14}$$

6.3 Preconditioners

As have been stated earlier, the convergence rate is dictated by the choice of preconditioners in the conjugate gradient method. Here has been chosen the methods which are in the category of incomplete LU preconditioners. The LU decomposition of a matrix is a decomposition where $A = LU$, where L is a lower triangular matrix and U is an upper triangular matrix. In principle a full LU factorization can yield the exact solution to $Ax = b$ by solving the triangular equations, $Ly = b$ and $Ux = y$. The algorithms presented here are taken from [42].

The LU decomposition performs Gaussian elimination to convert a matrix into the product of a lower and an upper matrix. The ikj version of Gaussian elimination is what is used as a basis for the algorithms; it is depicted in Algorithm 5,

Algorithm 5 Gaussian elimination ikj variant

for $i = 2, 3, ..., n$ **do**
  for $k = 1, ..., i-1$ **do**
    $l_{ik} = \frac{a(i,k)}{a(k,k)}$
    for $j = k+1, ..., n$ **do**
      $a(i,j) = a(i,j) - l_{ik} \cdot a(k,j)$
    end for
    $a(i,k) = l_{ik}$
  end for
end for

Doing an LU factorization and leaving out entries of some criteria makes an ILU preconditioner. In principle, the incomplete preconditioner can be written as

$$LU + E = A, \tag{6.15}$$

where E is an error term indicating the difference between the complete and incomplete factorization. L, and U are respectively the incomplete lower and upper factorizations. Such a preconditioner have the property that

$$L^{-1}AU^{-1} \approx I \tag{6.16}$$

where I is the identity matrix. Equation (6.16) has the condition number $\kappa(L^{-1}AU^{-1}) \approx 1$. It is worthwhile to mention that for an identity matrix, CG will find the solution in exactly 1 step, which is an indicator of good performance for this preconditioner. One problem these methods may have is that the resulting matrix may not be sufficiently symmetric positive definite, another issue is that the factorization may introduce instabilities or breakdown. These issues are resolved by using additional fill-in.

Different choices in what entries to leave out lead to different ILU methods. The

choice has been to implement two different methods which have different advantages and disadvantages. One of these is called the ILU0 preconditioner, and the other is called the ILUT preconditioner.

As for the choice of this preconditioner, there is another class of preconditioners which is more suitable for this particular setup. The incomplete Cholesky (ICHOL) decomposition is regarded as a better choice for positive definite symmetric matrices because it has the form $A = LL^T$. This preconditioner can guarantee that $L^{-1}AU^{-1}$ is symmetric since A is symmetric:

$$(L^{-1}AL^{-T})^T = L^{-1}A^TL^{-T} = L^{-1}AL^{-T}. \tag{6.17}$$

The same result does not apply for a general LU factorization, which may lead to issues for preconditioned CG. However in practice, the matrices produced have shown no issues. Therefore, the author has opted to use ILU instead of incomplete Cholesky since other solvers in MCCT can use ILU. Ignoring this important factor is justified by the above and since the identity matrix is symmetric and positive definite the approximation should tend to have similar properties as better approximations are used. Preconditioners were tested in MATLAB before implementation, and since incomplete LU and incomplete Cholesky performed the same in these tests, incomplete LU was chosen as it can be used in other solvers in MCCT.

6.4 ILU0 preconditioner

The (ILU0) preconditioner, given in Algorithm 6, is made by allowing only nonzero entries in positions also nonzero for the original matrix A. That is $L_{i,j} + U_{i,j} \neq 0$ if and only if $A_{i,j} \neq 0$. With this approach, the memory consumption, the number of calculations done, and the time to generate the preconditioner is kept as low as possible. Only the matrix entries are needed for L and U since the matrices are stored in the CRS format described in Appendix B.1, and the row pointer and column index are the same for the matrices $L + U$ and A. ILU0 is, therefore, a method which requires few resources but still gives a reasonable speedup in convergence. The zero in the name indicates that no fill-in is used. Fill-in is the related method where more entries, according to a level of fill-in rule, are allowed. Traditionally the fill-in rule is to accept entries where A^{k+1} have nonzero entries and k denotes the level of fill-in, so for the implementation here, $k = 0$ gives that the used entries are the nonzero entries of A.

Although this preconditioner does a decent job in making CG converge faster, it is worthwhile to invest more resources on the preconditioner since the number of calls to the solver is extremely high both in the reference-potential calculations and in MCCT main loop. Therefore, another suitable approach is implemented called (ILUT).

Algorithm 6 ILU0

for $i = 2, 3, 4, ..., n$ **do**
  for $k = 1, 2, , i-1$ and $A_{i,k} \neq 0$ **do**
    $l_{ik} = \frac{A_{i,k}}{A_{k,k}}$
    for $j = k+1, k+2, ..., n$ and $A_{i,j} \neq 0$ **do**
      $A_{i,j} = A_{i,j} - A_{i,k}A_{k,j}$
    end for
    $A_{i,k} = l_{ik}$
  end for
end for
Set L = unit lower matrix of A
Set U = upper matrix of A

6.5 Incomplete LU threshold preconditioner

The strategy of the ILUT preconditioner, shown in Algorithm 7, is to regard any substantial entries in the calculation as important. The idea is to decide what size of entries in the matrix is large enough to be substantial and include these entries in the factorization. To do this, ILUT takes as an additional parameter some value τ_t which is a threshold. Any entries in a full LU factorization with a magnitude less than this threshold is set to zero while the rest of the entries are kept. Many variations of this preconditioner exist depending on how the threshold is implemented and used. In this version, the entries which would also be in ILU0 is kept regardless of the size of these entries. Also, a condition to limit the memory consumption is applied. That is, despite being larger than the threshold, only a certain number of nonzero entries will be allowed in each row, removing the smallest entries.

The threshold,τ_t, checks the following relation,

$$\frac{|r(k)|}{||A_{(k,\cdot)}||} < \tau_t, \tag{6.18}$$

and if it is true, the entry is set to zero. Otherwise, it is included in the preconditioner as a nonzero entry. For each row, the last condition removes the smallest entries until the row has an acceptable number of entries.

Although ILUT does well in accelerating the convergence of CG, it is important to note that the prime interest is not how many iterations, but how long time it takes to converge. When using preconditioners, additional calculation cost is the price to pay. The more non-zero entries that are allowed into the preconditioners, the more costly it is to run an iteration. Therefore, the threshold parameter is key in controlling both the number of calculations and the number of iterations. In addition, calculating the preconditioner is costly for large matrices. Fortunately, it is, within reason, worth spending some time on calculating the preconditioner since the solver is extensively used in MCCT. Another limiting factor is that the memory

Algorithm 7 ILUT

$U_{1,\cdot} = A_{1,\cdot}$
for $i = 2, 3, 4, ..., n$ **do**
  copy the current row $r = A_{i,\cdot}$
  for $k = 1, 2, , i-1$ and $r(k) \neq 0$ **do**
    $r(k) = \frac{r(k)}{A(k,k)}$
    if $r(k) < thresholdcondition$ **then**
      $r(k) = 0$
    end if
    if $r(k) \neq 0$ **then**
      for $j = k+1, k+2, ..., n$ **do**
        $r_j = r_j - r_k U_{k,j}$
      end for
    end if
  end for
  if $r(k) < thresholdcondition$ **then**
    $r(k) = 0$
  end if
  while number of entries in r>allowed number of entries in each row **do**
    Set the smallest entry of row to zero
  end while
  $L(i, 1 : i-1) = r(1 : i-1)$
  $U(i, i : n) = r(i : n)$
  $r = 0$
end for

consumption can become vast for large matrices since the complete LU factorization can potentially have as much as n^2 nonzero entries. The limiting factor of the number of nonzero entries in each row is key to controlling memory consumption.

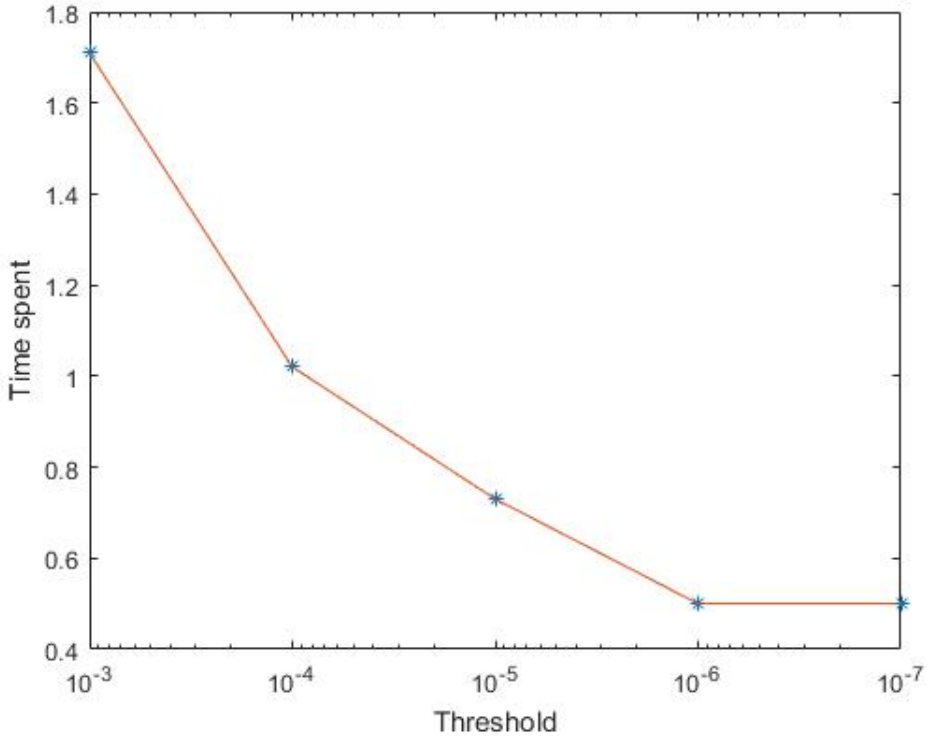
6.6 Convergence of CG

In this section the solver and preconditioners are tested, the test example is on a grid of 142,538 nodes and 285,074 elements, the characteristic step length is constant 0.005µm in the domain. It solves Equation (4.19) for node number $p_{node} = 70469$ which is at position $x = 2.44\mu m, y = 0.677\mu m$. This is on a rectangular grid of length $L_x = 3\,\mu m, L_y = 1\,\mu m$. The initial guess is $x_0 = 0$. The solver is set to stop at $\tau = 1 \cdot 10^{-6}$ residual.

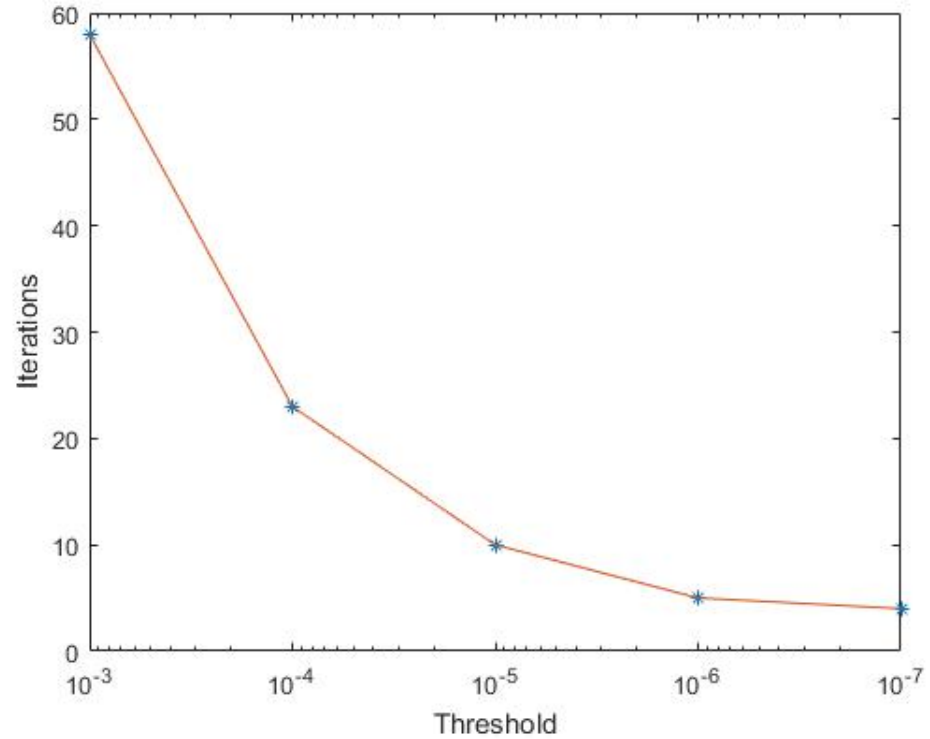
Without a preconditioner, the solver never converges. For ILU0, the solution converges in 366 iterations, and spends a total of 4.04s to obtain the solution. Figures 6.1a and 6.1b shows the time spent to obtain a solution and the convergence for the variation of the threshold parameter in ILUT. The number of non-zero entries allowed in each row was set to 1,000. The greatest speedup, in terms of time and iterations, is gained when only a few matrix entries are kept. As the threshold approaches zero, the effects are less remarkable. ILUT outperforms ILU0 for all thresholds tested. It should be kept in mind that the two preconditioners solve different issues, and the results are as expected. ILU0 requires much less memory but has more than double the time spent to calculate the solution, even for a very large threshold in ILUT. For a threshold $\tau_t \geq 10^{-2}$ the solution did not converge within 10,000 steps and is therefore not included. The convergence could be very slow for a large threshold, or more likely, the resulting preconditioned matrix is not sufficiently symmetric positive definite. As expected, a smaller threshold fixes this issue.

Figures 6.2a and 6.2b shows a smaller variation in the threshold. These figures also verify that the solution converges faster for a lower threshold, which corresponds to accepting more entries of the LU decomposition, but as can be seen from Figure 6.2a, this does not necessarily imply a speedup. The reason for this is that the more entries that are allowed into the matrix, the heavier the computations are at each iteration. For instance, at the threshold $\tau_t = 7.0 \cdot 10^{-6}$ the desired accuracy is achieved at 9 iterations, which is the same number of iterations as for $6.0 \cdot 10^{-6}$. However, the time spent is more for the latter because each calculation involves more entries in the preconditioner.

The purpose of this section is not only to show the convergence and show that it is working, but also to get some insight into what kinds of thresholds are useful in ILUT. Finding the best threshold for each matrix can be hard and needs to be done using user experience. The figures suggest that the lower the threshold, the better the preconditioner. Also, the most gains are found when the number of iterations is initially high for a threshold. If some threshold gives a low number of iterations, the speedup expected from lowering the threshold further is small. With respect to

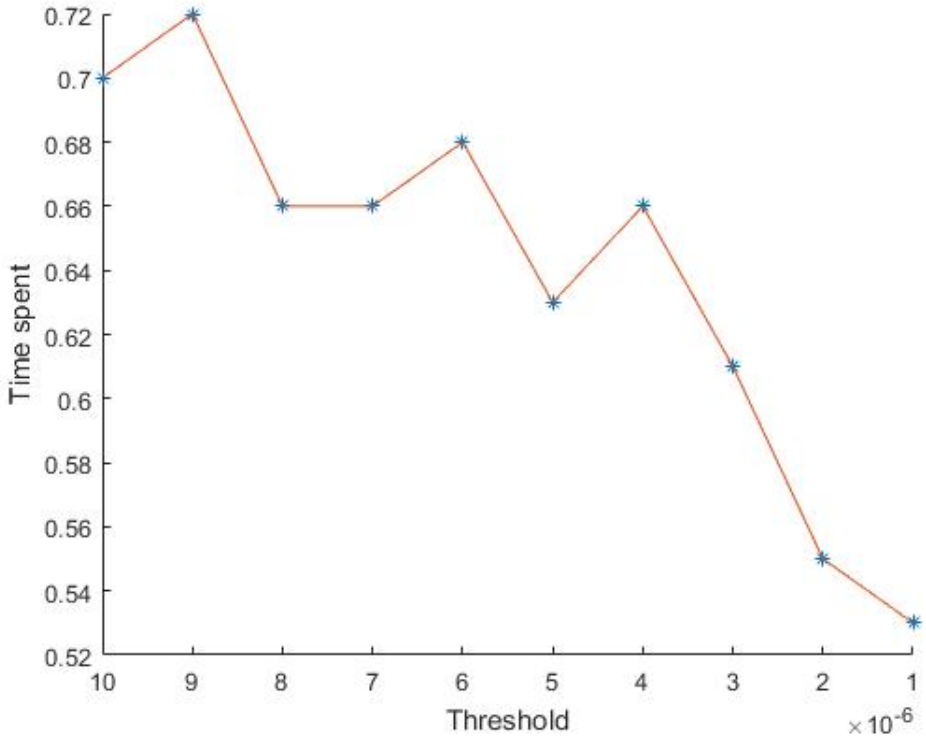


(a) Plot of time spent in seconds calculating the Poisson solver for varying threshold. Log(x)-Axis is inverted to show the variation as the threshold approaches zero.

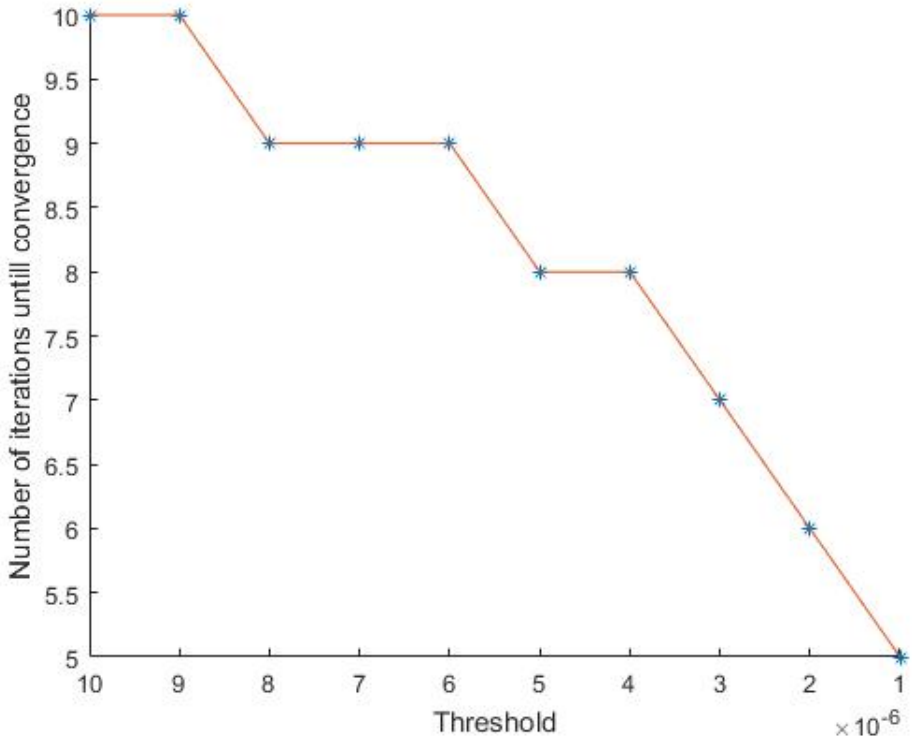


(b) Plot of iterations needed for convergence for varying threshold. Log(x)-Axis is inverted to show the convergence as threshold approaches zero.

Figure 6.1: Convergence of ILUT for $\tau_t \in [10^{-3}, 10^{-7}]$.



(a) Plot of time spent in seconds calculating the poisson solver for varying threshold. The x-axis is inverted to show the variation as the threshold approaches zero.



(b) Plot of iterations needed for convergence for varying threshold. The x-axis is inverted to show the convergence as threshold approaches zero.

Figure 6.2: Convergence of ILUT for $\tau_t \in [10^{-6}, 10^{-5}]$.

the time step loop in MCCT, it is even harder to come with a good estimate of the threshold parameter. The reason is that the initial guess depends on the previous iteration, so when the time step is low, the initial guess should be close to the new solution. This means that the number of iterations is even lower for convergence during the time step loop. Since the solver always must do at least one iteration, it is important that the allowed number of non-zero entries in the preconditioner is not too large. If it is made able to converge in only one iteration, it is quite likely that a preconditioner with fewer entries will be faster on average, since there will always be variation in the difficulty to obtain the solution in each time step.

Chapter 7

PN junction

PN-junctions got attention in the 1940s [43] and were well studied by the 1970s when the theory of PN-junctions was at a stage of standard textbook material. Here the theory has been found in [44], and the reader is referred to this book for further details.

Since PN-junctions are well described using analytical models, they are usually not modeled in simulators such as MCCT, except for the studies of [3] and [45]. However, since they are well described, they are good candidates for testing the FEM solver. Since PN-junctions need the electric field, they are great for catching mistakes in the calculation of the electric field or the potential.

In this chapter, the PN junction theory is introduced, then modeled using the FEM Poisson solver. In Section 7.1, the PN-junction is introduced and defined. In Section 7.2, the one-dimensional theory is introduced. In Section 7.3, the model used in MCCT is defined. In Section 7.4, the test cases are defined, and the near unbiased case is found. In Section 7.5, the behavior of particles is shown. In Section 7.6, the potential and electric field calculated are presented and discussed. In Section 7.7, the performance of the solution is discussed.

7.1 Introduction

Essentially, semiconductors are materials that have a resistance between metals and isolators. Their resistance can be altered by using different doping densities or temperatures. Doping is the act of introducing impurities to the semiconductor, which leads to the introduction of free electrons or holes in the material. If the doping introduces electron type carriers then it is called an N-type extrinsic semiconductor, if it introduces hole type carriers it is called a P-type extrinsic semiconductor. The doping level is quantitatively denoted through the donor and acceptor densities N_D and N_A. It is important to note that whenever a carrier is made, there is also a fixed charge of the opposite type left. That is, in a P-type semiconductor, there are left

negatively charged ions. Likewise, in an N-type semiconductor, there are positively charge ions. This is described as the background charge in MCCT.

A PN-junction is, as the name suggests, a junction of a P-type semiconductor and an N-type semiconductor. As the two types come in contact, the diffusion current of electrons and holes, from respectively the N-side and the P-side over to the opposite side will make the particles recombine and leave a carrier empty region. This region is referred to as the depletion zone. In this zone, there are no carriers left, and therefore only the background charge will yield a contribution to the potential. This results in a positive and a negatively charged side from which an electric field appears. This electric field gives a drift current in the opposite direction of the diffusion current. As time passes, an equilibrium enters in which the net currents of electrons and holes respectively are zero.

7.2 One dimensional theory

The theory of PN-junctions is well made for the one-dimensional case. Therefore, it is presented in this section for usage in discussing the simulation from MCCT. Only the results which can be used as tests are presented, and not the derivation of the results. The derivation can be found in an elementary textbook on the subject of semiconductors.

The easiest available test is to check the width of the depletion zone. When assuming a discontinuous transition from the depletion layer to the P and N side, the built-in voltage has to be $\phi_0 = \phi(x_N) - \phi(x_P)$, where x_N and x_P denotes the distance from the junction and the discontinuous transition on the N and P size. It can then be derived that

$$x_n = \sqrt{\frac{2\epsilon}{eN_D}\frac{N_A}{N_A + N_D}}\sqrt{\phi_0}, \tag{7.1}$$

and on the P side,

$$x_p = -\sqrt{\frac{2\epsilon}{eN_A}\frac{N_D}{N_A + N_D}}\sqrt{\phi_0} \tag{7.2}$$

In these equations, ϵ is the permittivity, e is the elementary charge, and N_D, N_A denotes the donor and acceptor densities respectively. Also, the electric field in the cross section between regions N and P is given as

$$E = -\sqrt{\frac{2e}{\epsilon}\frac{N_A N_D}{N_A + N_D}}\sqrt{\phi_0}. \tag{7.3}$$

The built-in voltage mentioned, which can also be compared to the simulator, is given by

$$\phi_0 = \frac{k_B T}{\epsilon} ln(\frac{N_A N_D}{n_i^2}), \tag{7.4}$$

where k_B is Boltzmann's constant, T is the temperature, and n_i is the intrinsic carrier density. The intrinsic carrier density depends on the material used. In MCCT the material simulated is $Cd_xHg_{(1-x)}Te$ where $x = 0.28$ denotes the cadmium fraction of the material. An approximation to n_i is found in [46], and is given explicitly as:

$$n_i = (5.585 - 3.82x + 0.001753T - 0.001364xT) \cdot 10^{14} \cdot E_g^{0.75} \cdot T^{1.5} \cdot e^{\frac{-E_g q}{(2kT)}} \tag{7.5}$$

where the energy gap E_g is found in [47] to be approximated by

$$E_g = -0.302 + 1.93x - 0.81x^2 + 0.832x^3 + 5.35 \cdot 10^{-4}(1 - 2x)T. \tag{7.6}$$

Unfortunately, it will be seen in this chapter that the analytical theory does not match every result. The theory is mostly applicable for PN-junctions with a built-in voltage in the range $0.2 - 1.0V$ while the built-in voltage calculated for the test case is 0.0287. Thus, the evaluation of quantities must not be awarded too much significance, and instead the qualitative behavior is emphasized.

7.3 MCCT model description

The chosen model is a P^+, P, N, N^+ model. The domain is heavier doped in the P^+ and N^+ region than in the P and N regions. The domain is depicted in Figure 7.1. The device is chosen to be of length $L_x = 3\,\mu m, L_y = 1\,\mu m$ with a small contact region of the same length as the P^+ and N^+ regions in x-direction, and the length $L_y^{cr} = 0.02\,\mu m$ in the y-direction. The doping factors have been chosen to be $10^{16} cm^{-3}$ for the N and P regions, and $10^{17} cm^{-3}$ for the N^+ and P^+ regions. The parameters used for the model are summarized in Table 7.1.

The simulations are done with a simple recombination implementation. Whenever an electron is in the proximity of a hole, they will recombine. The theory states that the recombination is the reason for depletion zones. However, it should not alter the results much since, without recombination, the particles are still able to be removed from the simulation through the contacts. Since the theory is applicable only to the steady state, it is not important how the steady state is achieved.

The variables are always the ones in Table 7.1, and the only varying factor is the Dirichlet condition of the left contact, the bias of external voltage applied to the

Variable	Value
L_x	$3\,\mu m$
L_y	$1\,\mu m$
$L_x^{P^+}$	$0.75\,\mu m$
$L_y^{P^+}$	$0.5\,\mu m$
L_x^{cr}	$0.5\,\mu m$
L_y^{cr}	$0.02\,\mu m$
Δt	$1\,fs$
N_D	$10^{16}\ cm^{-3}$
N_D^+	$10^{17}\ cm^{-3}$
N_P	$10^{16}\ cm^{-3}$
N_P^+	$10^{17}\ cm^{-3}$
T	$300\,K$
x	0.28
h	$1.6 \cdot 10^{-2}\mu m$
h^+	$4.01 \cdot 10^{-3}\mu m$

Table 7.1: Parameters in MCCT run.

simulation. Simulations were first conducted using a mesh with equal characteristic step length in the domain, but it was found that simulations could be improved by varying the characteristic step length. Figure 7.2 is a picture of the drawn device in GMSH, and the black dots indicate where the characteristic step length is defined. The arrows point at which characteristic step length is chosen in each node. Equation (3.47) is used with $\alpha = 1$ for both the heavy doped zone and the light doped zone yielding $h^+ = 4.01 \cdot 10^{-3}\mu m$ and $h = 1.6 \cdot 10^{-2}\mu m$. The arrows in the figure show which of the two characteristic step lengths are used for each node. The mesh generated in GMSH and used in this chapter is shown in Figure 7.3. As can be seen, the characteristic step length is used by doing a linear interpolation so that the element sizes vary linearly. With this mesh, the P^+ and N^+ regions are kept constant, near the suggested optimal step size, while the variation in element size happens in the lightly doped region. This mesh performed better than a mesh with equal step size in the whole domain of $h = 5 \cdot 10^{-3}\mu m$ and gained an advantage in speed of calculations. The choice of $\alpha = 1$ is important in the P^+ and N^+ regions considering these need to be large enough for the contact implementation. Since the background charge has less weight in P and N regions, these regions can have a less detailed mesh, which is taken into account just by using Equation (3.47). The mesh can further be altered for improved efficiency, but the focus here is just to demonstrate that it works and not to get the best model possible.

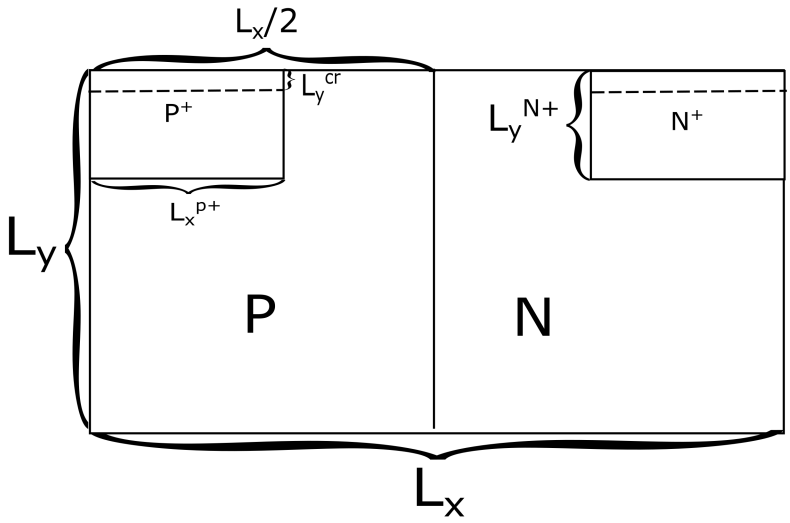


Figure 7.1: Illustration of the domain of a P^+, P, N, N^+ device. To keep the illustration readable, all lengths are not shown.

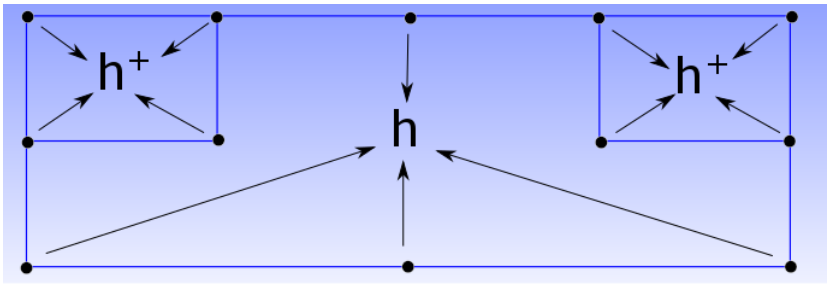


Figure 7.2: Picture of GMSH drawing of the domain, where the characteristic step length is defined in the black circles. The arrows denote which characteristic step length is chosen.

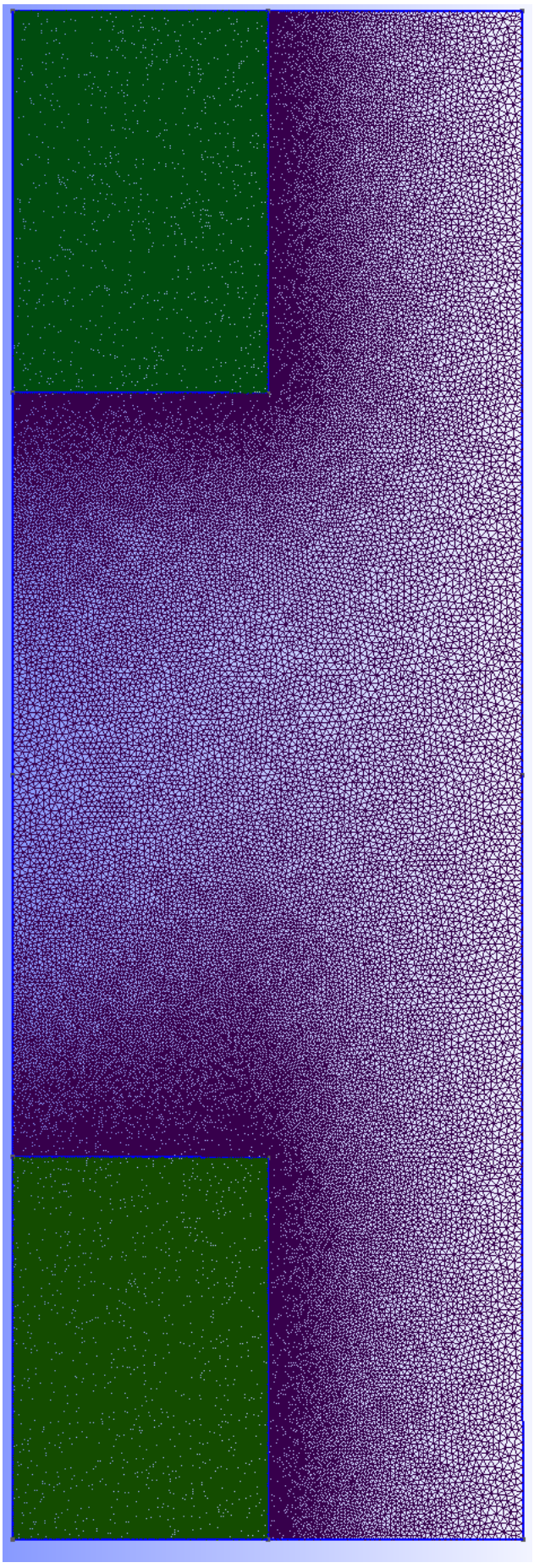


Figure 7.3: A picture of the mesh generated in GMSH and used in the PN-junction simulations, green areas have element size too small for the resolution to capture but are nearly of constant size $h^+ = 4.01 \cdot 10^{-}3\mu m$. The purple elements vary in step length between $h^+ = 4.01 \cdot 10^{-}3\mu m$ and $h = 1.6 \cdot 10^{-}2\mu m$.

All runs shown here has been started from the initial positions, which is to distribute the superparticles evenly as if they were just introduced via doping. An example is shown in Figure 7.4. The potential at the first time step is then calculated using this distribution of particles. The initial potential is dominated by the contacts since the distribution of particles creates charge free regions, but they contribute some noise to the solver through the discretization error and particle-mesh coupling. The initial potential of a PN-junction is shown in Figure 7.5 with Dirichlet left contact having applied -0.3V and right contact 0V. Since every calculation here is done in the two-dimensional domain, the third dimension is ignored in this chapter. This means that all results can be seen as per unit length, m, in the z-axis.

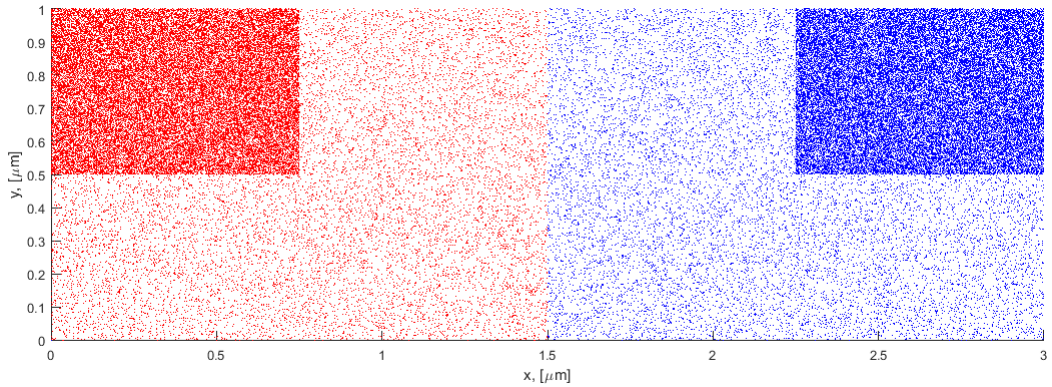


Figure 7.4: The particles' positions at initialization. Holes are indicated as red particles and electrons are indicated as blue particles.

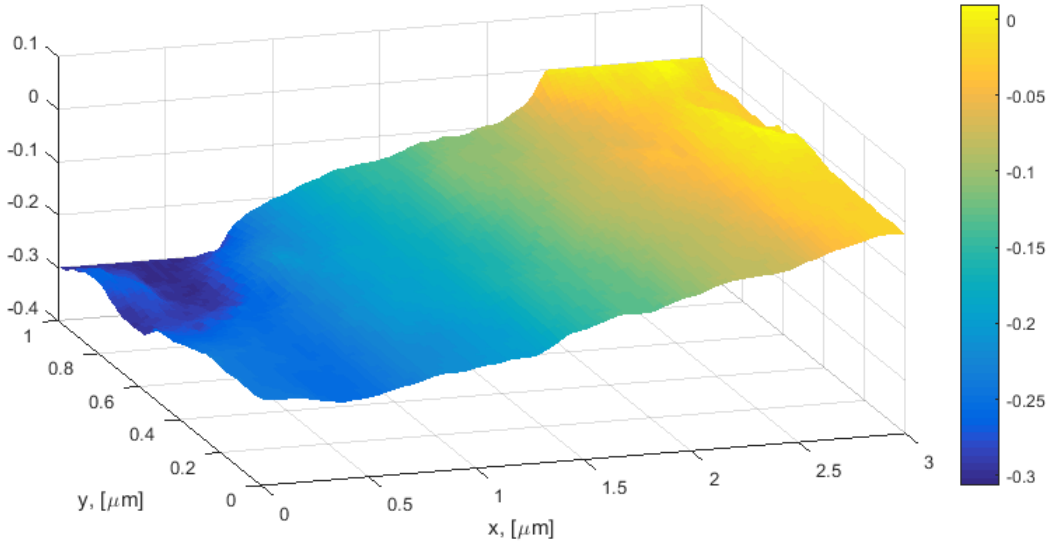


Figure 7.5: Initial potential for a PN-junction. The contacts have Dirichlet conditions of -0.3V and 0V on respectively the left and right contact.

7.4 Current

To be able to classify the simulation cases, which will be presented here, the bias of the cases must be found. One way to find which voltage yields zero bias is to measure the current in the PN-junction. Since the immediate current is dominated by noise, all figures in this section show the accumulated charge entering or exiting through the contacts. The currents are given by the slope of the accumulated charge, and the positive current is defined as the forward current of the PN-junction. The currents run parallel for the two contacts since there is no accumulation of charge inside the device. The different cases tested in this chapter are summarized in table 7.2, where D_P and D_N denotes the Dirichlet condition at the contacts respectively for the P side and the N side.

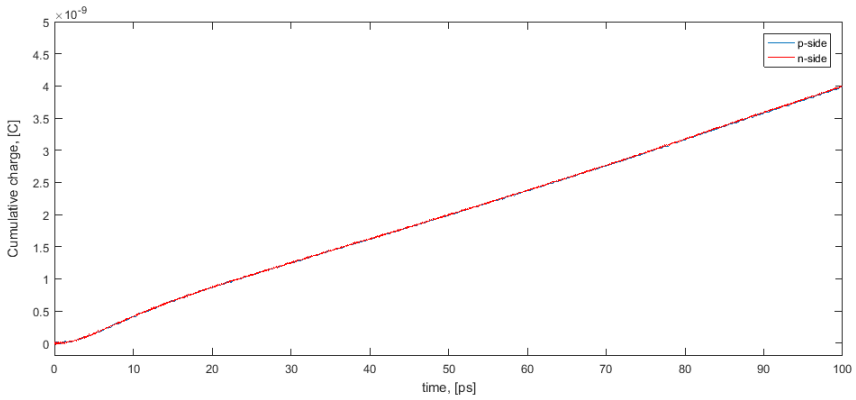
Case	D_P	D_N
1	-0.1	0
2	-0.2	0
3	-0.3	0

Table 7.2: Voltage at contacts for test cases.

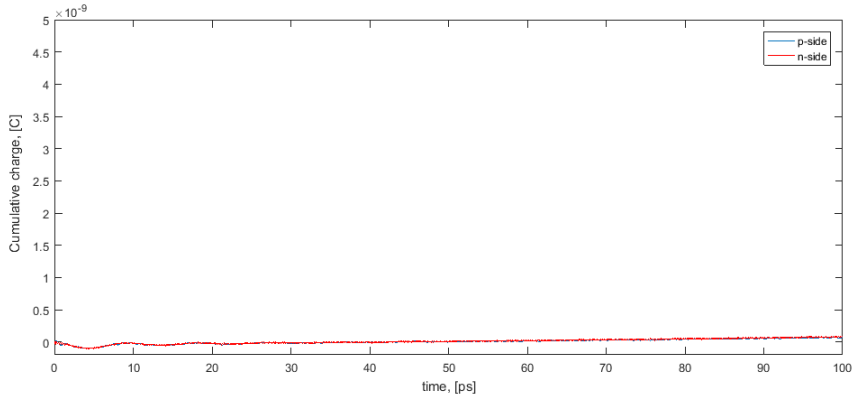
Figures 7.6a to 7.6c show the accumulated charge in the three cases. In Figure 7.6a, there is a current running through the device. With the direction of this current, case 1 is in forward bias, and its average current, after $40ps$ has passed, is $0.04\text{mA}\,\mu\text{m}^{-1}$. In Figure 7.6b there is also a small current. Since this current is so small, it is not calculated, and the case is treated as the near zero bias voltage. The third case, in Figure 7.6c shows that the current running through the device is approaching zero as the simulation goes on. The backward bias should have a small current from a generation of particles; however a generation has not been implemented, and the current becomes zero. Therefore, the near unbiased voltage is chosen as the lowest forward bias current seen, and case 3 is seen as the backward bias voltage case. Note that, in the first $10ps$ the backward bias has a current running in the negative direction since there are more particles than there should be, and therefore, the generation of particles is not needed during this time for the backward bias to get a current. When the simulation keeps going, the current becomes zero.

Figure 7.6c, illustrates the best that the current is alternating during the initialization phase. An alternating current can lead to instabilities in the simulator, which has to be taken into account when setting up the model. Figure 7.7 shows what happens when such instabilities are allowed to develop for a near zero bias PN-junction. To resolve this issue, the time step between each call to the Poisson solver has been made lower. As stated in Section 2.3.1, the time step is essential to keep the simulations stable. The simulator was unstable when the Poisson solver was called every second time step. The stable solutions have the Poisson solver called at every time step. From the figures, there is a reason to suspect that the instabilities are connected with the applied bias since the fluctuations in the current are less for the forward bias than the backward bias PN-junction. However, this has not been tested further, and it is unknown at this stage if it is the case.

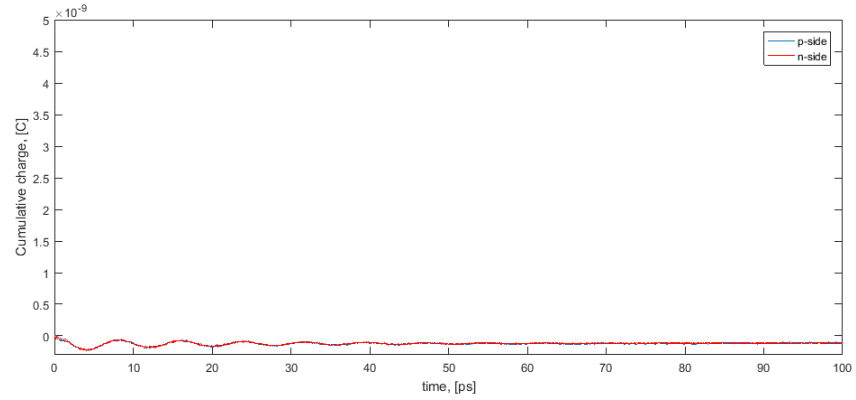
It should be stated that even though a lower timestep was needed, this did not create a long runtime. The run time was the same for the stable and unstable runs, even with twice as many calls to the solver. One reason for this is that the particle location algorithm had to travel less distance, and the initial guess for the potential for each time iteration was closer to the new solution, but the real culprit is that there is a known bug with losing track of particles when some particles leave through contacts. This does not make the solver less rigorous, but it makes some of the particles not search from their previous location. Since half the time step between each update means that on average this bug has half the impact, the speedup can mostly be accounted for by the bug. Some speedup is still because of the less travel distance and the closer initial guess in the solver.



(a) The accumulated charge over 100ps in Case 1.



(b) The accumulated charge over 100ps in Case 2.



(c) The accumulated charge over 100ps in Case 3.

Figure 7.6: The accumulated charge over 100 picoseonds measured in Coloumbs.

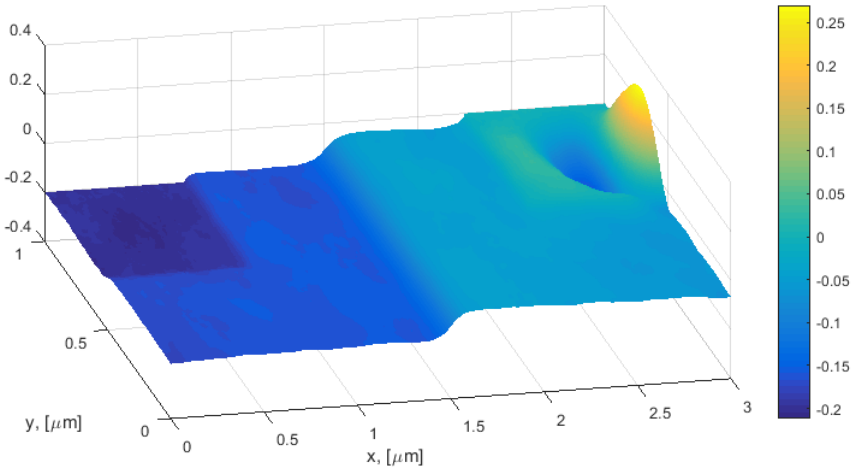


Figure 7.7: The potential after 60*ps* in a simulation with developed instabilities. An instability at the right contact is developing due to a Poisson time step chosen too large.

To summarize this section, it has been found that case 1 is forward biased, case 2 is near zero biased, and case 3 is backward biased. What is important here is that the simulation acts like a PN-junction. As the voltage is larger in forward bias, the current becomes larger, and as the voltage is larger in backward bias, the current becomes smaller. It can be concluded that the current behaves as expected, with the exception that there is no generation of particles for backward bias. Therefore from here on out, the three cases will be referred to as forward bias, near zero bias, and backward bias.

7.5 Depletion zone

From equations (7.1) and (7.2) the depletion zone of a one-dimensional PN junction can be calculated. In this section, the position plots of superparticles are shown to see how the depletion zone behaves. The theoretical one-dimensional depletion zone in zero bias is marked as two vertical lines in the figures. The cases are the same as in the previous section, shown in Table 7.2.

Figures 7.8 to 7.10 show the position plot of particles after $100ps$ for the three cases. In Figure 7.8, as expected from the forward bias PN-junction, some particles are inside the theoretical zero bias depletion zone. Although it has not been shown here, tests show that the simulators depletion zone becomes smaller as the voltage is increased further in the forward bias direction. In Figure 7.9 the depletion zone has increased and is close to the theoretical depletion zone, verifying that case 2 is close to the zero bias voltage. In figure 7.10 the depletion zone has increased further and is now clearly outside of the theoretical depletion zone from the one-dimensional model. The particles are doing as expected, since when the simulator is set in forward bias, the length of the junction zone decreases, and when the simulator is set in backward bias, the length of the junction zone increases. This verifies that the FEM solver is doing a correct job since it is the potential fields that cause this behavior. The Ohmic contact implementation seems to be working since there is no lack of particles in the contact region for any of the three cases. Since particles do not move away from, or toward the boundaries, the Neumann conditions are working. Self-forces does not impact the simulation since the particles are behaving as they are supposed to.

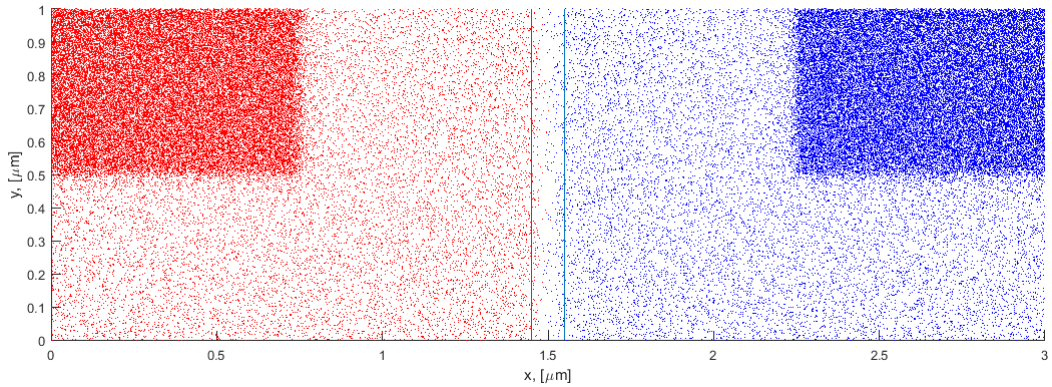


Figure 7.8: The particles' positions after 100ps for forward bias. Holes are represented as red particles and electrons are represented as blue particles.

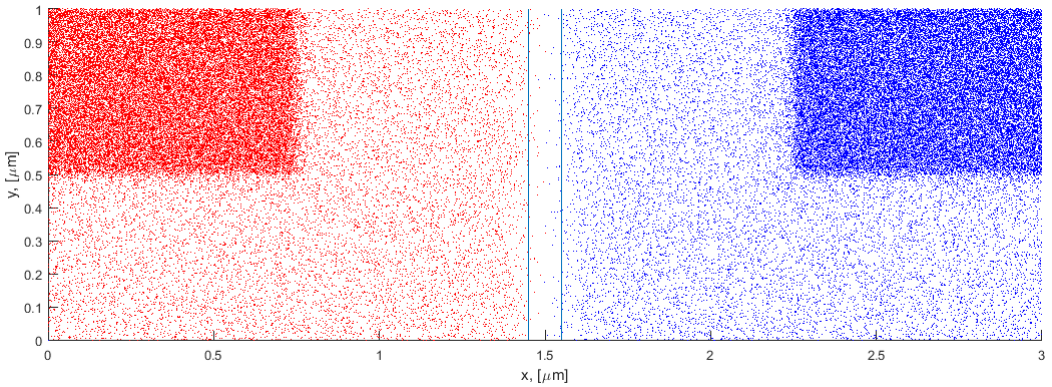


Figure 7.9: The particles' positions after 100ps for near zero bias. Holes are represented as red particles and electrons are represented as blue particles.

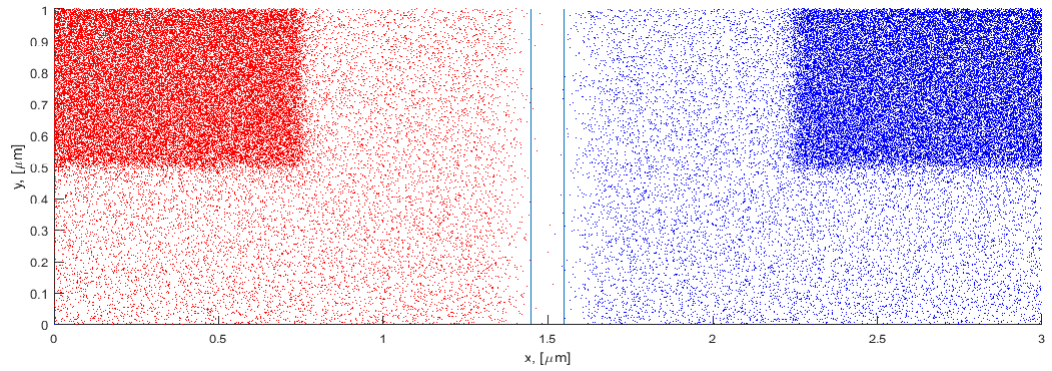


Figure 7.10: The particles' positions after 100ps for backward bias. Holes are represented as red particles and electrons are represented as blue particles.

7.6 Potential and electric fields

From Equations (7.3) and (7.4) the built-in potential and electric field at the junction is known in one-dimensional theory. In this section, the potentials and electric fields are shown for the simulation to see how these behave in comparison to the theory of PN-junctions. Since, as explained in Section 4.2 , the self-force cannot be extracted for the potential, the potentials are shown as is. Therefore, it is hard to say whether or not local fluctuations are from the self-force, or if they are from instabilities in the simulation. On the other hand, the self-force has been extracted for the electric fields and therefore these are the most accurate representations for this section. For the discussion, any fluctuations in the potential will be regarded as not being from the self-force. The one-dimensional model's built-in voltage is calculated from (7.4) to be 0.0287V and the electric field in the junction zone is calculated from (7.3) to be $-5.6\text{kV}\,\text{cm}^{-1}$.

The electric field plots are generated by making a Cartesian grid in MATLAB and showing the closest element center to each Cartesian grid node. This means that the electric field has been sampled and not averaged. So it is important to look at the broad picture in these figures and be aware that looking at a single electric field vector gives the electric field in the element at the vectors origin.

In Figure 7.11 the potential for the forward bias PN-junction is shown. Its electric field is shown in Figure 7.12. It would seem that when put in forward bias, the potential difference across the junction zone is small compared to the potential difference between heavier doped zones and lighter doped zones. Although the figure shows many fluctuations, these are very small and therefore are not as influential as they may seem. It has previously been experienced with MCCT that there may be some fluctuations and stability issues when the voltage is low.

Strictly speaking, the contacts are not Ohmic in this case since there is a small drop in the potentials in both contact regions. In the P-side, the potential decreases into the metal, and in the N-side the potential increases into the metal. From the position plot in Figure 7.8 it is seen that the effect is unnoticeable, and this is because the potential drop is very small. Figure 7.12 also demonstrates well that the Neumann zero boundary condition has been handled correctly. If the reference potentials were used in the elements with boundary nodes, then there would be an electric field on the normal of the boundaries. The electric field is not very large in the junction zone in forward bias. They are in the range $-2.5\text{kV}\,\text{cm}^{-1}$ to $-3\text{kV}\,\text{cm}^{-1}$, which is, as expected, smaller than the calculated electric field for a zero bias PN-junction. Most of the potential differences can be seen to be at boundaries of heavier doped regions, and so can be seen in the electric field as well. The potential difference across the junction for forward bias is close to the theoretical value for the unbiased PN-junction, it is 0.03V. This is an unfortunate result and is not following the one-dimensional theory.

In Figure 7.13 the potential for the near unbiased PN-junction is shown. Its electric field is shown in Figure 7.14. As the voltage difference at the contacts become larger,

the potential becomes smoother. Fluctuations are still present, but has become less distinct, and they stay in areas of approximate zero charge. On the other hand, the potential is smoother in areas of a large potential drop, for instance in the P^+, P boundary and in the junction zone. The potential drop in the contact region near the metal is smaller than before and is considered Ohmic in this case. The built-in voltage is the drop in potential in the junction zone; the one-dimensional theoretical value was $0.0287V$ and in the figure, it is seen that it is approximately $0.1V$ in MCCT. Here it can be seen that the theoretical value does not work well with the junction shown. The theoretical value is a good approximation in the range $0.2 - 1V$ and since the theoretical value is much less, in this case, it cannot be taken too seriously for the test case. The electric field in the junction is approximately $-6\text{kV}\,\text{cm}^{-1}$, which is somewhat larger than the theoretical value of $-5.5\text{kV}\,\text{cm}^{-1}$, but it is seen as a quite good result. As expected, the electric field is larger for the near unbiased case than for the forward bias case.

In Figure 7.13 the potential for a forward bias PN-junction is shown. Its electric field is shown in Figure 7.16. Now that there is a more substantial voltage, the potential is much smoother. The voltage difference across the junction is 0.2V, and therefore, the additional bias put into the contacts appear in the depletion zone, and not between heavier and lighter doped regions. The fluctuations are smaller, and there is no longer a potential drop at the contacts. The electric field in the junction is about $-10\text{kV}\,\text{cm}^{-1}$. This is much larger than the theoretical value and the other two cases and is as expected when the PN-junction is in backward bias. In this case, there is no potential drop in the metal in either contact.

It should be noted that the potential drop in the boundaries between heavier and lighter doped zones are nearly the same for all applied biases. These potential drops are unwanted, and make the comparison with the analytical model harder. Since these regions become large compared to the built-in voltage when the applied forward bias is large, it is expected that instabilities may be introduced since these potential drops become major components of the simulation. This also explains the pictures and how the total solution seems more stable for a large backward bias.

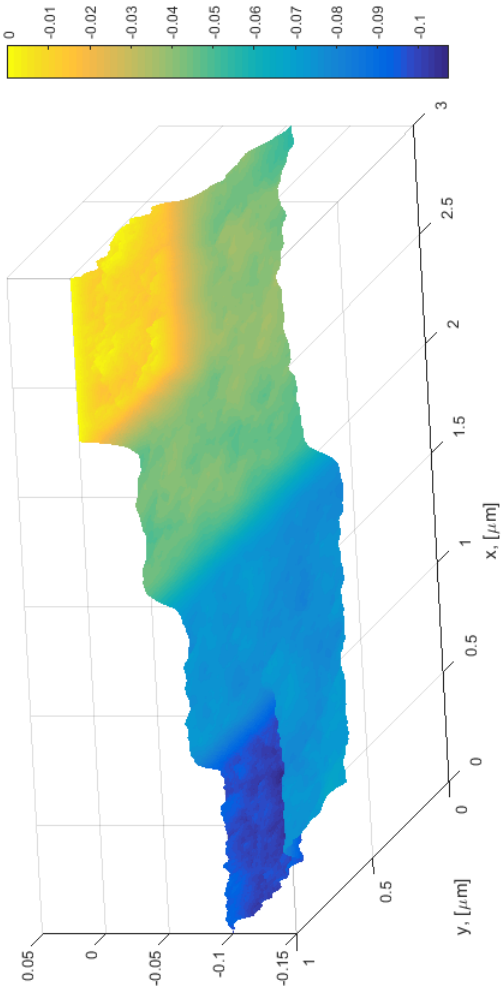


Figure 7.11: The potential after 100ps for forward bias.

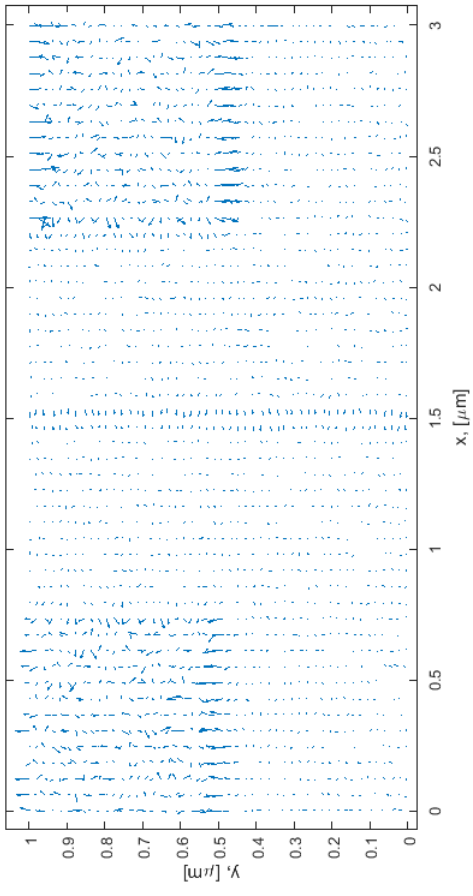


Figure 7.12: The electric field after 100ps for forward bias.

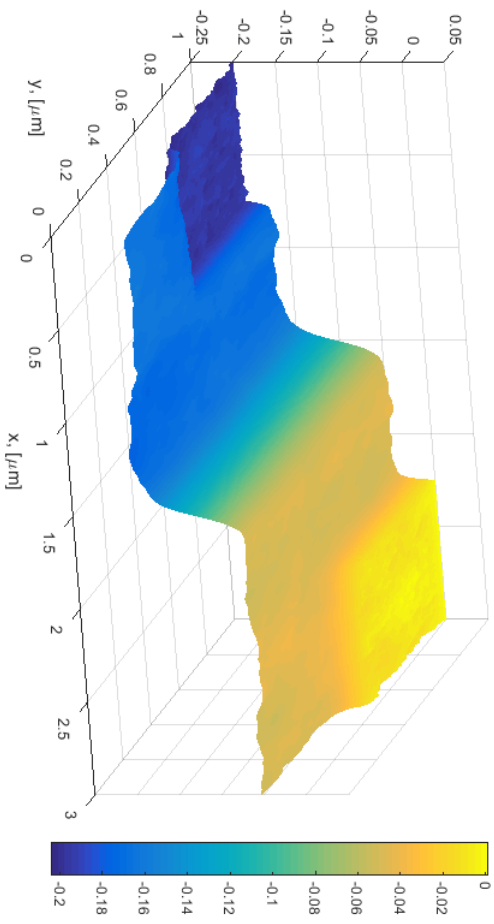


Figure 7.13: The potential after 100ps for near zero bias.

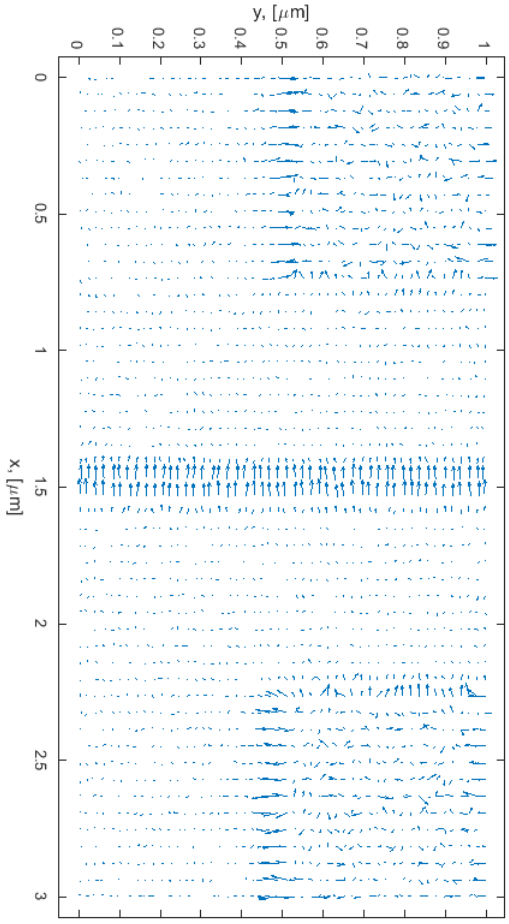


Figure 7.14: The electric field after 100ps for near zero bias.

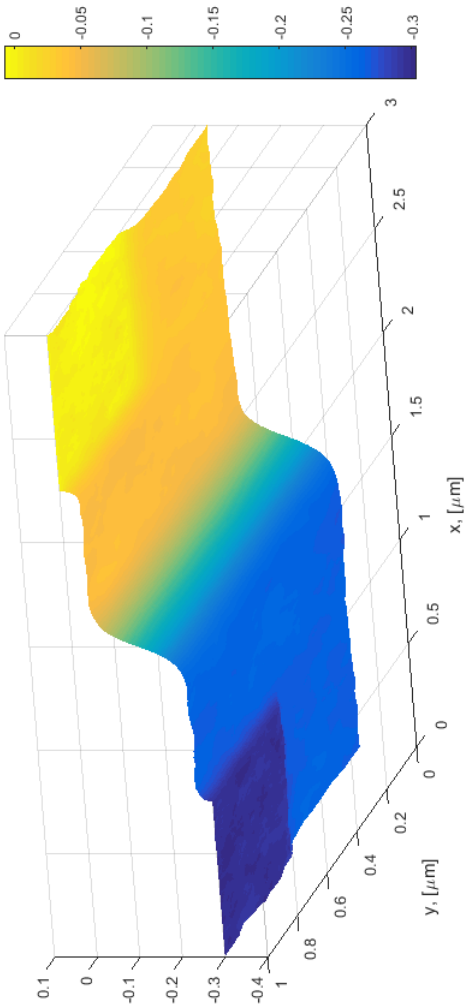


Figure 7.15: The potential after 100ps for backward bias.

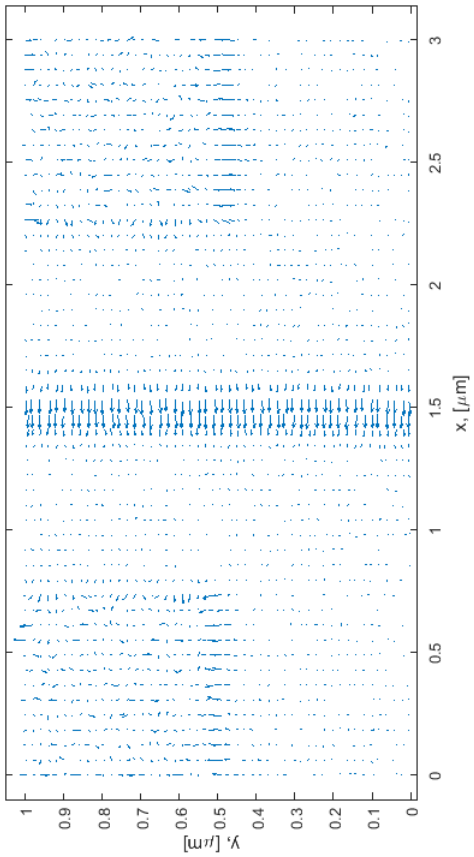


Figure 7.16: The electric field after 100ps for backward bias.

7.7 Discussion

The particles, electric fields, and the currents all seem to do as expected in the steady state of the PN-junction in accordance to the one-dimensional theory of PN-junctions. The built-in potential in the junction zone is, unfortunately, much larger than the theoretical value. This has been explained as the existence of potential drop between regions of heavier and lightly doped zones, as well as through the fact that the one-dimensional theory is best suited for modeling simpler cases where the built-in voltage is in the range of $0.2 - 1V$. It is seen that the simulations are most stable at a large backward bias and that the time step between each update of the potential must be low enough for the solver not to develop instabilities.

The contact implementation seems to give a nearly Ohmic contact, but may have some issues with a larger forward bias. The Neumann condition is working as there are no electric fields in the normal direction of the domain, and no electrons or holes are dragged to, or out of, boundary elements. The mesh recommendations are verified to yield a good result as it has performed well in this chapter, and although not shown in the report, performed as well as a uniform mesh of similar characteristic step length.

The simulations were ran using ILUT with the threshold $\tau_t = 10^{-6}$, and the non-improved version of the point location algorithm. This approach gave a run time of 25 hours for $100,000$ time step iterations. The average run time of PLA was 2.5s. For PCG, the run times vary in iterations, and they are approximately 0.1s before equilibrium and 0.02s during equilibrium. These run times suggest that the minor bug in the point location algorithm, mentioned in Section 7.4, slows the total run time by a considerable amount since it is the point location algorithm which is the bottleneck. From the low run time of PCG, it can be concluded that it is a very efficient solver in practice for MCCT and that the ILUT implementation has been verified to work well.

There has unfortunately not been enough time to test the total run time with the improved version of PLA and the ILU0 implementation. However, the tests performed, in Section 6.6, for ILU0 suggests that the average run time would increase to approximately 6.5s, which is more than double of what has been the case with ILUT. This would, theoretically, use more than 50 hours of computation time. Unfortunately, there has not been enough time to implement the usage of the improved PLA in the MCCT time loop. An error in the implementation cause the program to shut down when 25% of the time steps are done. However, the improved PLA spends 0.9 seconds on average for each time step up to this point using h_{gn} given by Equation (5.12). This time is improved by using $\frac{h_{gn}}{10}$, in which case the improved PLA spends 0.6 seconds per time step. Both of these, therefore, give a considerable speedup from the unimproved version.

Chapter 8

Discussion and further work

There have been many topics to cover in order to make the working FEM implementation for MCCT. These topics both individually and together need to be discussed regarding limitations, advantages, and further work. In this chapter, the work is summarized and discussed altogether. In Section 8.1, the work is summarized. In Section 8.2, the work and the possibilities are discussed. In Section 8.3, further work is suggested.

8.1 Summary

In this master's thesis, a working FEM implementation has been developed for MCCT using linear bases, constant background charge, and a particle-mesh coupling where the charges are modeled as infinitely small. The framework presented is general enough to employ other particle-mesh coupling schemes. The contacts have been treated in a working minimalistic approach using conditions of charge neutrality and thermal equilibrium. An estimate for the size of the mesh has been suggested and verified to be working in the PN-junction. The electric field has been calculated from the bases of the potential. To remove self-forces the approach of Kalna et al. has been implemented, and the reduction has been improved from their report. A point location algorithm has been implemented to find particle positions in the domain. It has been implemented efficiently using additional structures, and by handling insertion and removals of superparticles correctly, except for in the case of particles leaving the domain through contacts. This exception only matters for speed and not for robustness. A memory-speed tradeoff improvement has been implemented to be able to gain additional speed in the algorithm when needed. The preconditioned conjugate gradient method has been implemented with two preconditioners. The first preconditioner, ILU0, was made for cases where memory is a scarce resource, and the second preconditioner, ILUT, was implemented for speed. The implementations have been kept fast by utilizing the CRS format to keep the needed quantities continuous in memory. A PN-junction has been simulated to show the working implementation. The results are positive, the implementation is effi-

cient, and the PN-junction simulations verify that the implementation is working. The method is topical as FEM has had issues with self-forces until recent years, and the implemented self-force reduction is from 2015 and has been improved in this implementation.

8.2 Discussion

The results of the PN-junction verifies that the implementation is working. However, there are still many properties of the implementation left to discuss. First of all, the particle-mesh coupling and the self-force reduction demands a fine grid since the particles cannot interact if they are part of the same element. Also, the electric field calculations here have properties similar to the NEC scheme; they are discontinuous across elements. This is known to be a culprit of stability issues. At the same time, the contact implementation demands large elements to be able to keep the contact region Ohmic. These two requirements are opposites, and therefore give restrictions on the mesh. On the other hand, these restrictions are local, and one of the advantages of FEM is that it can vary the mesh size. To some extent, this makes up for the mesh restrictions, but it is also possible to fix these issues. Ohmic contact regions are currently fixed by large local elements, but can also be done using merged elements in the charge neutrality test.

The electric field can be made continuous by using a different electric field scheme, for instance, one can use one of the methods in [22] which makes an interpolation over neighboring elements' natural electric field. This interpolation scheme makes the electric field smoother. It also makes it so that the particles can interact, since the electric field can vary within the element. The only issue with this approach is that the self-force reduction needs to take additional steps to be a working implementation. However, the method has less self-force than other naive methods, and may, therefore, be a viable solution in some cases, even without the reduction. Making the self-force reduction may be a bit more intricate in this type of electric field calculation. A simpler way that eliminates all the mentioned issues is to use higher order elements in FEM. Higher order elements will make the natural electric field approximation vary within the element, and can guarantee that the electric field is continuous across elements. The issues with higher order elements are that the cost is higher, and they do not capture rapid variations as well as linear elements. On the other hand, higher order polynomials give a smoother potential, which in many cases is a property needed for a stable simulation. Also, it is likely that fewer higher order elements are needed than linear elements, and this is favorable for the speed of the point location algorithm.

Although the implementation gives restrictions on the mesh, it is already known that there are restrictions to capture the interaction between particles. At no time during this work has there been any indications that the implementation needs an even stricter mesh than what is known to be the case from before. On the other hand, it should be expected that this is the case because these previous statements are made for structured meshes, in particular, the nodes are distributed in the $x-$

and $y-$directions with either constant step size, or varying step size. Since the unstructured mesh has more variability in the details in any particular point in the mesh, it should be expected that the mesh needs to be finer than for the structured case.

It should also be noted that the particles are always calculated from the center of an element in the PMC scheme. Therefore, it is possible that a smoother potential is acquired if the particle-mesh coupling uses the particle's position in the element. This author's opinion is that it is unlikely to produce much of a difference since the particles do not interact in the element, and as long as the electric field is discontinuous across elements, it should not make a large difference in the electric field. Unfortunately, there has not been enough time to test if it does make a difference. Also, the focus has been on keeping the self-force out of the simulations. Therefore, the NEC scheme is the most appropriate solution because equilateral triangles yield zero self-force for this scheme. The mesh used does not employ equilateral triangles and therefore the self-force reduction is still needed, but it could be used as an advantage by keeping boundary elements equilateral to make sure these do not have a self-force.

The need to have a fine grid is remedied by the fact that the linear system solver and the point location algorithm are fast, even for fine grids. Keeping these algorithms quick has been a priority, but still, they have their pitfalls. The LU preconditioners lack a solid mathematical foundation for well behaving in the conjugate gradient method. This means that there is no guarantee this algorithm won't break down in some cases. So far, this has never happened. The point location algorithm is great as long as the particle boundary conditions are handled elsewhere in MCCT. It is capable of having particle boundary conditions in it, but it does so unnaturally. On the other hand, there are positive results regarding the implementation of the LU preconditioner and the point location algorithm. They are very well made for utilizing the available memory for speed gains and have been reliable throughout. They also interact well with previous time steps since the convergence of the PCG has been very quick for small timesteps, and the PLA has found its elements very fast when the particles travel a short distance.

One of the main reasons to implement a FEM solver is that it is capable of calculating the potential for arbitrary domains. This gives the possibility of simulating more complicated devices. Such an implementation requires that the attributes of the domain are associated with the elements and that the particle boundary conditions are naturally implemented with the particle location algorithm. Unfortunately, there has not been enough time to make the extension to more general domains. The other advantage of FEM is that it can use step sizes locally as an added advantage, which has been utilized to great success.

Speed is always a concern. In this implementation, there are two distinct cases of speed. Every time a new mesh is generated, the reference potentials of this mesh must be made. Since this is computationally expensive, the time taken is substantially more for the first run on a mesh. Fortunately, the reference potential calculations are near perfectly parallelizable, and this should be utilized in the future.

In the second case, when the same mesh is used, the PLA and the PCG are the two bottlenecks. PCG is very quick, and speedup using parallelization is limited, it cannot be done for the algorithm itself, but it can be done in the matrix-vector products involved. Such an approach is not recommended since it is a low-level parallelization, which is often suboptimal, and the solver is so quick that it is unlikely to make it faster. PLA, on the other hand, is in the original version quite slow and is highly parallelizable.

8.3 Further work

Since the solver has been made for a two-dimensional domain, a three-dimensional solver is a natural extension. The framework and the subjects are the same in that case, but many details must be implemented differently. Firstly, the FEM method with linear elements has a different analytical solution; secondly, the PLA needs a different algorithm since it is made for the two-dimensional domain, and finally the size of the matrices involved are much larger, and therefore, the linear system solver must be even more efficient.

From the discussion in the last section, it is apparent that there is also much work that can be done in the two-dimensional solution. To solve the issue of a non-continuous electric field across boundaries, it is recommended to test higher order elements. There are many favorable properties that suggest that this is worth a try. As said, this makes the electric field continuous across boundaries while making the potential smoother in the domain. The PLA becomes faster as there will be fewer elements needed. Since PLA is the bottleneck, and the solver is quite fast, it is a good tradeoff. What degree of polynomials should be used is hard to say. The HP-FEM methods, which vary both polynomial degrees and mesh size, has results of exponential convergence rate in some cases[48][49]. This suggests that testing higher order elements is worth an effort. In principle, quadratic triangular elements are sufficient to make the electric field continuous, but it is possible that higher order polynomials are better. An added advantage with higher order polynomials is that the needed density of nodes in the mesh increases which leads to less elements and more nodes. It should also be emphasized that lower order polynomials are better for rapid variation, and therefore it is probable that a mix of different types is the optimal solution.

A factor often overlooked in practice is that the mesh has a great influence on the convergence of FEM. It is suggested here that it might be fruitful to explore further the possibilities of generating optimal meshes. This is suggested because even the few steps taken for a more optimal mesh in Chapter 7 lead to a much more efficient simulation, without loss of accuracy. There are several options for this consideration, one can use HP-fem as mentioned, or it is possible to make use of either a previous simulation run to generate a suitable mesh for future runs by using error estimates from adaptive mesh refinement methods[50], or use simpler models of devices to get results which can be interpreted to generate the mesh. Since adaptive mesh refinement methods have highly expensive iterations, only their error

estimates should be used, and not the methods themselves. The error estimates can be used either to choose element sizes, or element polynomials in regions, or both. The issue with very advanced FEM implementations is that they would lead to very advanced self-force reductions. Heavy emphasis on this approach to improve FEM will need a mesh generation implementation if the options are to be fully explored, but existing mesh generators can be utilized to a large extent. A compromise between the different suggestions here would be to use different meshes depending on the stage of the simulation, and use a mesh generating software to generate a mesh that can be improved after generation.

The Conjugate Gradient method is likely to be one of the most efficient methods for FEM since it utilizes the properties of the matrix and is well tested. Therefore, it is unlikely to need any further work, or be exchanged with a different solver. On the other hand, most of the calculations lie in the preconditioners, and these are always hard to make sure are the best available. First off, it is known that, usually, the incomplete Cholesky preconditioner is a better choice, but in this case, all testing has shown that ILU and IChol perform the same. However, if the making of the preconditioner takes too much time, Cholesky is quicker to make, and can, therefore, be used. What is worth to note is that it may be much to gain from using other preconditioners, and more sophisticated versions of the ilu preconditioners may do better. One such new preconditioner is presented in [51], which is made for electromagnetic applications and another is the multigrid method [52] used for particle simulations on a rectangular mesh. As stated, knowing which preconditioners work well in advance is hard, so that testing these in MCCT is needed for any suggestions.

For the point location algorithm, there is, most likely, not much potential speedup if the improved version is used. But for the general domain, it would be favorable to upgrade this to [29] so that the particle boundary conditions can be naturally implemented. The current implementation is also reliant on the Delaunay attribute and needs to be replaced if another type of mesh is used.

It should be mentioned that some parts of the implementation can be parallelized. In particular, the PLA is highly parallelizable, and since it is the bottleneck, it should be considered for the future. Also, the calculation of reference potentials is near perfectly parallelizable since these iterations have the same workload. This is the bottleneck of the initialization of FEM and takes substantial computation time on the first simulation on a mesh. Other than these two, there is not much potential for using parallelization for the current implementation. On the other hand, these two are the most expensive implementations, and it is worthwhile to parallelize these. The current implementation of preconditioners cannot be parallelized, but in [53] a new method which makes this possible is shown. It is not something of an issue in today's implementation, but in the future, it could be considered.

Chapter 9

Conclusion

A working two-dimensional FEM implementation has been made and illustrated to work in a simulation of a PN-junction. The goal of the master's thesis was to make an efficient FEM implementation without the influence of self-forces. The self-force has been shown to be reduced to the order of the tolerance of the linear system solver. This solver and a point location algorithm have been implemented efficiently, and their convergence has been demonstrated. The simulation of a PN-junction has shown no influence of self-forces and is working as intended.

Efficiency and usability were focused on, and therefore estimates have been found for the implementations. In particular, estimates for the characteristic step length have been made, estimates and a discussion on step lengths of the Cartesian coordinate system in the improved PLA have been given, and a demonstration and discussion on the threshold parameter of ILUT is shown. These estimates have been used with great success in the simulation of a PN-junction.

A great deal of discussion on the subjects of advantages and disadvantages have been conducted throughout the work. In particular, it has been pointed out that there are many ways to do the particle-mesh coupling, contact treatment, and electric field calculations. Each choice has been reasoned upon, and the results show that the reasons have been good enough for a working implementation. The most important issues pointed out are that the electric field is discontinuous across elements and that the implementation needs adjustments for usage within a general domain. The discontinuity of the electric field is also the case for finite difference method with the NGP scheme, and therefore the FEM solver is not worse in this regard.

Overall, the conclusions throughout seem to agree with the practical application. It is believed that the current FEM implementation is well suited for Monte Carlo charge transport simulations for similar devices to those used with other solvers, but with the added advantage of being able to dictate the mesh properties to a much greater extent, leading to a better solver. It is also believed that with a few tweaks, the solver can simulate more general domains. Also, it has shown to be efficient and robust in the application for PN-junctions.

Appendix A

Calculations

A.1 Centered difference method

The electric field is given as

$$\vec{E} = -\nabla\phi. \tag{2.8}$$

Forward finite difference method, in half a step size, is defined as the approximation using first order Taylor series in $x-$ and $y-$ directions as

$$\begin{array}{rcl} \Delta^x_{i+\frac{1}{2},j} & = & \frac{\phi_{i+1,j}-\phi_{i,j}}{h}, \\ \Delta^y_{i+\frac{1}{2},j} & = & \frac{\phi_{i,j+1}-\phi_{i,j+1}}{h}. \end{array} \tag{A.1}$$

The backward finite difference method can be represented by inserting $i-\frac{1}{2}$ instead of $i+\frac{1}{2}$. Then the average of forward and backward finite difference is used to get the centered difference method, which is

$$E^x_{i,j} = \frac{\Delta^x_{i+\frac{1}{2},j} + \Delta^x_{i-\frac{1}{2},j}}{2}. \tag{A.2}$$

The same procedure is done in $y-$ direction. By insertion in Equation (2.8) the result is the centered difference approximation to the E field:

$$\vec{E}_{i,j} \approx \qquad \begin{bmatrix} \frac{\phi_{i+1.j}-\phi_{i-1,j}}{2h} \\ \frac{\phi_{i.j+1}-\phi_{i,j-1}}{2h} \end{bmatrix}. \tag{A.3}$$

A.2 Gradient of area basis

From section 3.3 the areas are

$$\begin{aligned} A_1 &= \frac{||\overrightarrow{(p_3 - p_2)} \times \overrightarrow{(p - p_2)}||}{2}, \\ A_2 &= \frac{||\overrightarrow{(p_1 - p_3)} \times \overrightarrow{(p - p_3)}||}{2}, \\ A_3 &= \frac{||\overrightarrow{(p_2 - p_1)} \times \overrightarrow{(p - p_1)}||}{2}. \end{aligned} \tag{3.21}$$

Taking A_1 as an example, the rest follows analogously.

$$\nabla A_1 = \nabla \frac{||\overrightarrow{(p_3 - p_2)} \times \overrightarrow{(p - p_2)}||}{2} = \nabla \frac{||\overrightarrow{(p_3 - p_2)} \times \overrightarrow{(p)}||}{2}. \tag{A.4}$$

With $u = [u_1, u_2]$ constant, $v = [x, y]$, and the z coordinate fixed as $z = 0$ the following holds true:

$$\nabla||\overrightarrow{(u)} \times \overrightarrow{(v)}|| = \nabla||0 \cdot \vec{x} + 0 \cdot \vec{y} + (u_1 v_2 - u_2 v_1)\vec{z}|| = \begin{bmatrix} -u_2 \\ u_1 \end{bmatrix}. \tag{A.5}$$

Inserting equation (A.5) in equation (A.4) yields that

$$\nabla \frac{||\overrightarrow{(p_3 - p_2)} \times \overrightarrow{(p)}||}{2} = \begin{bmatrix} -(p_3 - p_2)^y \\ (p_3 - p_2)^x \end{bmatrix}, \tag{A.6}$$

where $(\cdot)^y, (\cdot)^x$ denotes taking respectively the second and first coordinate in a Cartesian coordinate system vector.

Following the same approach for all the bases, the gradients are

$$\nabla A_1 = \begin{bmatrix} -(p_3 - p_2)^y \\ (p_3 - p_2)^x \end{bmatrix}, \tag{A.7}$$

$$\nabla A_2 = \begin{bmatrix} -(p_1 - p_3)^y \\ (p_1 - p_3)^x \end{bmatrix}, \tag{A.8}$$

$$\nabla A_3 = \begin{bmatrix} -(p_2 - p_1)^y \\ (p_2 - p_1)^x \end{bmatrix}. \tag{A.9}$$

A.3 Backward and forward substitutions

Given an upper matrix U

$$U = \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & u_{nn} \end{bmatrix} \tag{A.10}$$

the solution to the system

$$Ux = b \tag{A.11}$$

can be solved by backward substitution since

$$x_n = \frac{b_n}{u_{n,n}} \tag{A.12}$$

and then substituting this into $n-1$ gives

$$x_{n-1} = \frac{b_{n-1} - u_{n,n-1}x_{n,n}}{u_{n-1,n-1}} \tag{A.13}$$

that gives the general formula given the previous $i-1$ rows as

$$x_i = \frac{b_i - \sum_{k=i-1}^{n} u_{i,k}x_k}{u_{i,i}}, \tag{A.14}$$

which is employed as the backward substitution. For forward substitution, the same concept is applied, but now the first row is used instead of the last row.

A.4 GMSH

As stated in Section 3.2 the Delaunay algorithm is applied through the usage of the free mesh generation software GMSH [18]. The version used in this master thesis is 2.10.1. GMSH lets the user draw a domain using their interactive menu and built-in computer-aided design engine. This generates a .geo file which can be altered to set proper values.

The output of GMSH is a .msh file, which is a list of nodes and their corresponding coordinates, and a list of elements and their corresponding nodes. The list of elements includes the edges on the boundary, which can, therefore, be used to identify boundary nodes. The software also includes post-processing which could be used to visualize results. However in this master thesis, MATLAB has been used instead.

In the future, the mesh should be generated by MCCT itself because there are several considerations to take into account. The most important being that the mesh size needs to be smaller than the Debye length at subdomains where the density of charge rapidly varies as mentioned in section 2.3.1. For further efficiency and accuracy, this consideration can be coupled with an adaptive mesh refinement finite element method. See [54] for the development of mesh generation software.

The method described in [55] is interesting in this context. This algorithm makes the elements close to equilateral, which is an alternative to minimize self-forces. Also, it is made suitable for moving grid points or elements, which makes it able to adapt the grid to where particles are during the simulation. In [56], this property is used to make such an adaptive triangular element generation method.

Appendix B

Implementation details

B.1 CRS format

The CRS format stands for compressed row storage format. It is a general format for storing large sparse matrices without any particular structures. The idea stems from the idea of storing each element of a matrix as the three vectors

$$\begin{aligned} A &= a_k \\ i &= i_k \\ j &= j_k \end{aligned} \tag{B.1}$$

where $A = A_{i,j}$ such that a_k stores the value of the matrix element at row $i = i_k$ and column $j = j_k$. In which case, only nonzero elements and their position in the matrix needs to be stored. The CRS format takes this approach one step further and instead stores the following three vectors.

$$\begin{aligned} A &= a_k \\ i &= i_k^{pointer} \\ j &= j_k \end{aligned} \tag{B.2}$$

such that $i_k^{pointer}$ denotes the position k in the matrix A for each row i, and j is a column index. $i_k^{pointer}$ is number of rows +1 long. The total number of nonzero elements is stored in the last $i_k^{pointer}$. In this way each row $A_{i,\cdot}$ is found as

$$A_{i,\cdot} = a_{(i_k),(i_k+1),\ldots,(i_{k+1}-1)} \tag{B.3}$$

so that the length of each row i is $i_{k+1}^{pointer} - i_k^{pointer}$. This storage type complements the fact that many of the operations needed in the solvers are row-wise operations which are naturally stored in succession using this format. Every matrix made in MCCT has been stored using this format.

For the reader to easily grasp the concept, an example should suffice. Take the following arbitrary matrix,

$$A_{full} = \begin{bmatrix} 1 & 0 & 0 & 9 & 2 \\ 3 & 0 & 0 & 0 & 7 \\ 0 & 0 & 8 & 4 & 0 \\ 1 & 2 & 3 & 4 & 5 \\ 0 & 0 & 0 & 0 & 6 \end{bmatrix} \tag{B.4}$$

now, this matrix is stored continuously in its nonzero entries, as

$$A = [1, 9, 2, 3, 7, 8, 4, 1, 2, 3, 4, 5, 6] \tag{B.5}$$

the starting entry in A for each row is given in i as

$$i = [1, 4, 6, 8, 13, 14] \tag{B.6}$$

and the column index of each entry in A is given as

$$j = [1, 4, 5, 1, 5, 3, 4, 1, 2, 3, 4, 5, 5] \tag{B.7}$$

Now the number of nonzero entries in A is given by the last entry in i as $14 - 1 = 13$ nonzero entries. The most used operation is to get a row of A. Say that the third row is needed, then the number of nonzero entries in this row is $i_4 - i_3 = 8 - 6 = 2$, and the starting entry is given as i_3, so the third row contains the values $A_{i_3} = A_6 = 8$ and $A_{i_3+1} = A_7 = 4$. The column index is given in the same entries as $j_6 = 3$ and $j_7 = 4$.

In MCCT it is often needed to do matrix-vector products. For instance, say there is a vector $\vec{b}$,

$$\vec{b} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \tag{B.8}$$

then taking the product $A\vec{b}$ can be done row-wise. For instance for the first row, the i vector gives that A must be iterated over entry number $1, 2$,and 3, and these entries are in the columns $j(1), j(2), j(3)$, which are the corresponding rows of $\vec{b}$. This means that

$$A_{1,.}\vec{b} = \sum_{i=1,3} A_i b_{j_i} = 1 \cdot 0 + 9 \cdot 1 + 2 \cdot 1 = 11 \tag{B.9}$$

and as can be seen, only nonzero entries of A needs to be done in the calculations. Furthermore, the calculations are done continuously in memory row-wise, and if the rows are done from the first to the last, it is continuous for the full calculation. These two attributes make the calculations efficient.

Bibliography

[1] J.L. Meek. A brief history of the beginning of the finite element method. *International journal for numerical methods in engineering*, 39:3761–3774, 1996.

[2] Carlos A Felippa. Introduction to finite element methods. *Course Notes, Department of Aerospace Engineeing Sciences, University of Colorado at Boulder*, 2004.

[3] C.N. Kirkemo. Monte carlo simulation of pn-junctions. Master's thesis, University of Oslo, 2011.

[4] Juri Selvåg. High precision, full potential electronic transport simulator. Master's thesis, Norwegian University of Science and Technology, 2014.

[5] S.E. Laux. On particle-mesh coupling in monte carlo semiconductor device simulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(10):1266 – 1277, 1996.

[6] H. Kosina and M. Nedjalkov. The monte carlo method for semi-classical charge transport in semiconductor devices. *Mathematics and computers in simulation*, 55(1):93–102, 2001.

[7] Geir Uri Jensen. *Monte Carlo simulation of III-V semiconductor devices*. PhD thesis, The Norwegian Institute of Technology, 1989.

[8] Dragica Vasileska, Stephen M. Goodnick, and Gerhard Klimeck. *Computational Electronics: Semiclassical and Quantum Device Modeling and Simulation*. CRC Press, 2010.

[9] C. Jacoboni and P. Lugli. *The Monte Carlo Method for Semiconductor Device Simulation*. Springer-VerJag/Wien, 1989.

[10] J.M. Thijssen. *Computational physics*. Cambridge University Press, 2007.

[11] Ole Christian Norum. Monte carlo simulation of semiconductors – program structure and physical phenomena. Master's thesis, Norwegian University of Science and Technology, 2009.

[12] J.J. Harang. Implementation of maxwell equation solver in full-band monte carlo transport simulators. Project thesis, Norwegian University of Science and Technology, 2015.

[13] Tomás González and Daniel Pardo. Physical models of ohmic contact for monte carlo device simulation. *Solid-State Electronics*, 39(4):555–562, 1996.

[14] Tobias Kramer, Viktor Krueckl, Eric J. Heller, and Robert E. Parrott. Self-consistent calculation of electric potentials in hall devices. *Physical Review B*, 81(20):205306, 2010.

[15] Karl Hess. Monte carlo device simulation: Full band and beyond. *The Springer International Series in Engineering and Computer Science*, 144, 1991.

[16] Alfio Quarteroni. *Numerical models for differential problems*, volume 2. Springer Science & Business Media, 2009.

[17] Jochen Alberty, Carsten Carstensen, and Stefan A. Funken. Remarks around 50 lines of matlab: short finite element implementation. *Numerical Algorithms*, 20(2-3):117–137, 1999.

[18] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[19] Joe Warren, Scott Schaefer, Anil N. Hirani, and Mathieu Desbrun. Barycentric coordinates for convex sets. *Advances in computational mathematics*, 27(3):319–338, 2007.

[20] Manuel Aldegunde, Karol Kalna, S.P. Hepplestone, and PV Sushko. Multi-scale simulations of metal-semiconductor contacts for nano-mosfets. In *Computational Electronics (IWCE), 2014 International Workshop on*, pages 1–4. IEEE, 2014.

[21] Ian Moffat Smith and Denwood Vaughan Griffiths. *Programming the finite element method*. John Wiley & Sons, 5 edition, 2014.

[22] M. Aldegunde, N. Seoane, K. Kalna, and A.J. García-Loureiro. Reduction of the self-forces in monte carlo simulations of semiconductor devices on unstructured meshes. *Computer Physics Communications*, 181(1):24–34, 2010.

[23] Manuel Aldegunde and Karol Kalna. Energy conserving, self-force free monte carlo simulations of semiconductor devices on unstructured meshes. *Computer Physics Communications*, 189:31–36, 2015.

[24] DT Lee and Franco P Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on computing*, 6(3):594–606, 1977.

[25] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM transactions on graphics (TOG)*, 4(2):74–123, 1985.

[26] Peter J. C. Brown and Christopher T. Faigle. A robust efficient algorithm for point location in triangulations. Technical report, Cambridge University, 1997.

[27] Genong Li and Michael F Modest. An effective particle tracing scheme on structured/unstructured grids in hybrid finite volume/pdf monte carlo methods. *Journal of Computational Physics*, 173(1):187–207, 2001.

[28] R. Chorda, J.A. Blasco, and N. Fueyo. An efficient particle-locating algorithm for application in arbitrary 2d and 3d grids. *International journal of multiphase flow*, 28(9):1565–1580, 2002.

[29] A. Haselbachera, F.M. Najjarb, and J.P. Ferryk. An efficient and robust particle-localization algorithm for unstructured grids. *Journal of Computational Physics*, 225(2):2198–2213, 2007.

[30] Michał Wichulski and Jacek Rokicki. Fast point location algorithm on triangular meshes. *Journal of Theoretical and Applied Mechanics*, 46(2):315–324, 2008.

[31] Jean-Lou De Carufel, Craig Dillabaugh, and Anil Maheshwari. Point location in well-shaped meshes using jump-and-walk. In *CCCG*, 2011.

[32] S.B. Kuang, A.B. Yu, and Z.S. Zou. A new point-locating algorithm under three-dimensional hybrid meshes. *International Journal of Multiphase Flow*, 34(11):1023–1030, 2008.

[33] Hanhui Jin, Chao He, Sutao Chen, Canxing Wang, and Jianren Fan. A method of tracing particles in irregular unstructured grid system. *The Journal of Computational Multiphase Flows*, 5(3):231–237, 2013.

[34] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6), 1952.

[35] Alfio Quarteroni. *Numerical models for differential problems*, volume 2. Springer Science & Business Media, 2010.

[36] Jean Charles Gilbert and Jorge Nocedal. Global convergence properties of conjugate gradient methods for optimization. *SIAM Journal on optimization*, 2(1):21–42, 1992.

[37] Tomáš Gergelits and Zdeněk Strakoš. Composite convergence bounds based on chebyshev polynomials and finite precision conjugate gradient computations. *Numerical Algorithms*, 65(4):759–782, 2014.

[38] Zdenek Strakoš and Petr Tichỳ. On error estimation in the conjugate gradient method and why it works in finite precision computations. *Electron. Trans. Numer. Anal*, 13(56-80):8, 2002.

[39] Hussein Hoteit and Abbas Firoozabadi. Numerical modeling of two-phase flow in heterogeneous permeable media with different capillarity pressures. *Advances in Water Resources*, 31(1):56–73, 2008.

[40] Rosilene A. Kraft and Alvaro L.G.A. Coutinho. Deflated preconditioned conjugate gradients applied to a petrov-galerkin generalized least squares finite element formulation for incompressible flows with heat transfer. *International Journal of Numerical Methods for Heat & Fluid Flow*, 25(2):272–298, 2015.

[41] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, 2003.

[42] Youcef Saad. Ilut: A dual threshold incomplete lu factorization, 1994.

[43] Michael Riordan and Lillian Hoddeson. The origins of the pn junction. *IEEE spectrum*, 34(6):46–51, 1997.

[44] K.A. Ingebrigtsen and A. Tonning. *Forelesninger i fysikalsk elektronikk*. Seksjon for fysikalsk elektronikk, NTH, 1979.

[45] C. Moglestue. Monte carlo particle simulation of the hole-electron plasma formed in a pn junction. *Electronics Letters*, 7(22):397–398, 1986.

[46] G.L. Hansen and J.L. Schmit. Calculation of intrinsic carrier concentration in hg1- xcdxte. *Journal of Applied Physics*, 54(3):1639–1640, 1983.

[47] G. L. Hansen, J.L. Schmit, and T.N. Casselman. Energy gap versus alloy composition and temperature in hg1- xcdxte. *Journal of Applied Physics*, 53(10):7099–7101, 1982.

[48] Martin Costabel, Monique Dauge, and Christoph Schwab. Exponential convergence of hp-fem for maxwell equations with weighted regularization in polygonal domains. *Mathematical Models and Methods in Applied Sciences*, 15(04):575–622, 2005.

[49] Jens Markus Melenk and Christoph Schwab. hp fem for reaction-diffusion equations i: robust exponential convergence. *SIAM journal on numerical analysis*, 35(4):1520–1557, 1998.

[50] Miguel G Filippi, Marcelo G Vanti, and Patrick Kuo-Peng. A performance comparison of adaptive operator-customized wavelet basis and adaptive-refinement methods for 2-d finite-element analysis. *IEEE Transactions on Magnetics*, 52(3):1–4, 2016.

[51] J Cerdán, J Marín, and J Mas. A two-level ilu preconditioner for electromagnetic applications. *Journal of Computational and Applied Mathematics*, 2016.

[52] Marco Saraniti, Achim Rein, Günther Zandler, Peter Vogl, and Paolo Lugli. An efficient multigrid poisson solver for device simulations. *IEEE transactions on computer-aided design of integrated circuits and systems*, 15(2):141–150, 1996.

[53] Edmond Chow and Aftab Patel. Fine-grained parallel incomplete lu factorization. *SIAM Journal on Scientific Computing*, 37(2):C169–C193, 2015.

[54] Pascal Frey and Paul-Louis George. *Mesh generation*. John Wiley & Sons, 2013.

[55] Per-Olof Persson. *Mesh generation for implicit geometries*. PhD thesis, Citeseer, 2004.

[56] M.C. Lee and M.S. Joun. Adaptive triangular element generation and optimization-based smoothing, part 1: On the plane. *Advances in Engineering Software*, 39(1):25–34, 2008.