Geometrical Optimization of Planar Nano Vacuum Channel Transistors

by

Adina R. Bechhofer
B.A., Queens College, City University of New York (2020)
B.S., Columbia University School of Engineering and Applied Science (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2023

Author .....
Department of Electrical Engineering and Computer Science
January 25, 2023

Certified by .....
Luca Daniel
Professor of Electrical Engineering and Computer Engineering
Thesis Supervisor

Certified by .....
Karl K. Berggren
Joseph F. and Nancy P. Keithley Professor in Electrical Engineering
Thesis Supervisor

Certified by .....
P. Donald Keathley
Principal Research Scientist , Research Laboratory of Electronics
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Geometrical Optimization of Planar Nano Vacuum Channel Transistors

by

Adina R. Bechhofer

Submitted to the Department of Electrical Engineering and Computer Science
on January 25, 2023, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Nano vacuum devices have demonstrated tunneling emission in low voltages due to their 10 nm scale gaps that create order 10 GV/m electric fields with just 10 V. The small gaps give rise to ballistic transport through the channel, which combined with the low capacitances of the electrodes, gives rise to ultrafast response times. Nano vacuum channel devices have also exemplified robustness in the face of extreme radiation and temperature conditions [28, 18]. The design of nano vacuum devices is unintuitive due to the complicated and partially unknown physics governing their operation. In this thesis, we present an approach to performing shape optimization on nano vacuum channel devices based on an adaptation of a simulated-annealing [55] algorithm. We defined figures of merit to maximize the current in a diode, minimize the off-to-on current ratio in a transistor, and minimize the gate leakage current in a transistor. We implemented a finite element electrostatic simulation to calculate the emission-current-density profiles on emitting tips in diodes and transistors. We also implemented a heuristic to particle tracking to speed up the simulations and optimization of transistors.

Using the optimization framework developed in this work, we are able to reach device designs that achieve a 6-orders-of-magnitude performance improvement compared to the initial geometry in approximately 10,000 optimization steps. For each emission model assumed, we uncover unique geometrical features that enhance the performance of devices on figure of merit of interest.

This work establishes a free and open-source framework for electronic device optimization. Using this framework, device designers and engineers can spend less time, money, and research efforts on developing efficient and high performing devices.

Thesis Supervisor: Luca Daniel
Title: Professor of Electrical Engineering and Computer Engineering

Thesis Supervisor: Karl K. Berggren
Title: Joseph F. and Nancy P. Keithley Professor in Electrical Engineering

Thesis Supervisor: P. Donald Keathley
Title: Principal Research Scientist , Research Laboratory of Electronics

Acknowledgments

I would like to express my deepest gratitude to my advisors, Dr. P. Donald (Donnie) Keathley, Prof. Karl Berggren, and Prof. Luca Daniel for their invaluable advice, mentorship, guidance, and support in all stages and aspects of this work. Their contributions range from introducing me to physical models, optimization techniques, and debugging methods to giving me access to tools and infrastructure that enabled this work. Without their help, this work would simply not exist. Special thanks to Prof. Berggren and Dr. Keathley for their editorial suggestions and comments.

Many thanks to Jose E. C. Serralles and my collaborators at the University of Colorado Boulder, including Dr. Greg Werner, Luke Adams, Jesse Snelling, and Prof. John Cary for their advice and suggestions on this work.

I would like to thank Marilyn Meyers for her assistance with the technical aspects of this work.

I am grateful to the National Science Foundation as well as the Air Force Research Laboratory for generously supporting my work.

I am incredibly grateful to the members of the Quantum Nanostructures and Nanofabrication group, past and present, for their professional support, friendship, and their help in navigating the bureaucracy of MIT. Special thanks go to Dr. Marco Turchetti, whose work laid the foundations for my work, and to Torque, Emma, and John for their kind friendship.

I must acknowledge my dear friends who have stood with me, offering endless words of support and encouragement. Thank you Julia, Theo, and Reuven for always being there for me. Thank you Akiva, Alison, Anna, Yael, Aleeza, Toby, Tziona, Liat, Magd, Jojo, and many more people whose names this page is too short to contain, for always lending an ear to listen and always offering a kind word. Many thanks to my friends on the MIT Grad Hillel board, including Sabrina, Tal, Will, and Lucy for their friendship and community.

Lastly, I would like to thank my parents and siblings for supporting and being proud of me, even though they do not understand what it is that I do.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The physics governing the operation of vacuum devices is nonlinear, making design of optimal devices an unintuitive task. Exploring device design by trial and an error is costly because of the resources invested in every round of fabrication and testing. Therefore, we implement shape optimization to computationally arrive at an optimal design without costly exploration with fabrication and testing.

There are few works applying shape optimization to electron emitters, probably due to the computational complexity involved in simulating and optimizing electron devices. In this thesis, we will introduce a framework for shape optimization applied to vacuum electron emitters. We will use the framework to optimize the emitter tip in vacuum channel diodes and transistors. We parameterize the emitter by a collection of points with the polygon resulting from their interpolation as the emitter. We use a global non-gradient algorithm because the cost functions considered are ill suited for gradient calculations.

1.1 A background of vacuum electronics

The first vacuum tube diode was demonstrated in 1898 Thomas Edison who noticed current flowing between a heated filament and a plate that were both encased in a glass bulb that has been evacuated from air. The current flow in the diode was one directional, a property that Fleming used in 1904 to rectify electrical signals and to detect radio waves. Three years later, in 1907, DeFrost showed that the current between the filament and the plate can be controlled

by a grid with independent voltage placed between the filament and the plate. Thus, the first triode device was born. The newly invented triode was used to amplify radio signals and significantly contributed to the development of long distance communications. In the 1930's it was observed that the triode operation starts to break down when operated at frequencies with wavelengths comparable to the size of the device's elements. It became understood that the inductance in the leading wires along with the capacitance of the terminals created a "short circuit" effect in the vacuum tube device. Many attempts were made to change the area of the electrodes and the length of the connecting wires in effort to extend the frequency range of vacuum amplifiers. The frequency limitations were overcome to some extent by development of techniques that coupled microwave energies to resonant cavities and waveguides. By the 1940's, "lighthouse" cavity devices reached maturity and were ubiquitous in communication and computing [3]. Bell communications used vacuum tubes for their transcontinental telephone and radio network until 1982 when those were finally replaced by GaA transistors [13].

The first planar semiconductor transistor appeared in 1959, taking over the world of computing by storm. The FET technology allowed for lower power consumption and faster computation than the original vacuum tubes. The semiconductors also got rid of the expensive and fragile vacuum encasing. For the 50 years following the introduction of the planar transistor, computation experienced an exponential growth driven by transistors being produced at lower costs and smaller sizes. This allowed manufacturers to fit more transistors on a chip for an affordable cost. At the same time, operating clocks frequencies were increased, allowing more computation per device per second [19, 5]. In recent years, it appears that the physical limit for semiconductor transistor scaling was reached. This has been driving researchers to look at alternative devices and exotic materials.

It is commonly thought that the success of semiconductor transistors brought about the vacuum tube's fall from glory. However, vacuum tubes did not fall to complete disuse when semiconductors took over. They just got repurposed for niche applications. In the early 2000's vacuum devices were used mostly as sources in transmitters operating between UHF and X-ray with peak powers of up to tens of gigawatts. They had a range of applications in the military, commercial sector, and scientific research. Examples include particle ac-

celerators, satellite communication, hypothermia applicators, and military communication devices. The old glass encasing of the devices was replaced by a new metal-ceramic encasing. Additional new features include the use of rare-earth magnets and moderate temperatures instead of high temperatures [14].

With the recent advances in nano fabrication technology, vacuum electronics have experienced a resurgence in the research community. Reborn as nano-fabricated metal and ceramic electrodes with 10 nm scale gaps, they are interesting for several reasons. First, the gaps between the emitter and collector are smaller than the mean free path of air, allowing for vacuum operation in ambient conditions and ballistic transport of electrons between terminals. Additionally, the small area of the electrodes gives rise to device capacitances on the order of 10 aF. Together with the ballistic transport, this give rise to ultra fast response times and operating frequencies of 100THz and beyond. Furthermore, nano vacuum devices have been observed to be resistant to extreme radiation and temperature conditions [28, 18], unlike traditional semiconductors, which are quite sensitive to harsh environments. Some device geometries consist of vertical emitters sticking up from the substrate [27]. However, here we consider planar gold and TiN devices because of the convenience of integrating them into integrated circuits. Those devices have been tested and found to have stable currents for approximately 1,000 hours of operation, making them reliable for application [42]. Nanofabricated vacuum electronic devices can be used in high performing integrated circuits, room temperature sensors, field emission displays, and space electronics [28].

The physics governing the vacuum device operation is rather complicated and highly nonlinear. Thus, human intuition might not be the best guide to designing optimal devices. The general wisdom in the field is to make emitters as sharp as possible, but sharp tips are difficult to fabricate and tend to degrade over time. Many resources are invested in every round of fabrication and testing. In this work, we apply shape optimization as a way to computationally arrive at an optimal design without costly trial and error exploration in the lab.

1.2 Shape optimization

Shape optimization made its first appearance in the field of mechanical structure optimization. The usage of optimization methods for improved design is an old idea and has been in use for a long time in the form of optimization over a small set of rigid design parameters. More flexible and free-form design algorithms start appearing in the 1960s with the grid-and-truss optimization. In 1981, Cheng and Olhoff [4] published a work about optimizing thickness distribution in elastic plates. Seven years later, their work was followed by a publication by Bendsøe and Kikuchi [6] focusing on the material distribution method for topology design using computational methods. Those methods were revolutionary because they expanded the design space from a small one described by few parameters to an essentially infinite design space with hundreds of parameters. Material density methods remain the leading methods for parameterizing problems in structural design. It also inspired works in photonics inverse design and optimization. The material density method divides the design domain into pixels or voxels where the material density is decided as a continuous value between 0 and 1, where 0 indicates air, and 1 indicates full material [15].

Inverse design started appearing in the field of nanophotonics in the late 1990's, with a contributions by Spühler et al. and by Cox and Dobson. Spühler used a genetic algorithm to evolve a telecom fiber into a ridge waveguide coupler [11]. Cox and Dobson used a gradient type algorithm to optimize the bandgap in a periodic photonic crystal [12]. Those two methods became archetypal in the field shape classical optimization for photonics. As of this day, almost all works follow either a gradient descent method or an evolutionary (genetic) algorithms.

Each optimization method requires a parameterization, a cost function, and an optimization algorithm. In photonics there are two very common ways to parameterize a design problem. The material density, which draws inspiration from the mechanical material density, relies on dividing the design domain into voxels and describing the permittivity voxel i as

$$\epsilon_i = \epsilon_1 + \lambda_i(\epsilon_2 - \epsilon_1) \tag{1.1}$$

with $\lambda_i \in [0, 1]$. ϵ_1 and ϵ_2 are the permittivity of the two design materials (typically air

and another material). In each iteration of the optimization, λ_i is updated. After the final iteration, a thresholding segments the permittivities to either ϵ_1 or ϵ_2. The second common parameterization is known as the level set method. It assumes a smoothly varying function $\Phi(x)$ over the design space. The two design materials interface with each other at the level set of $\Phi = 0$. To move towards a design, Φ is evolved with Hamiltonian motion or gradient descent.

Shape optimization has become ubiquitous in photonics with applications in design for nonlinear optics and for nano scale photonics. It has been used to reduce the self heating of a coil, design a material cloak, optimize near field effects, achieve a target diffraction, optimize coupling and polarization, and design solar cells [32].

The figure of merit or cost function vary by application. In a work by Piggot et al., the authors use a two step cost function known as "objective first". First, Maxwell equations are calculated. Then, the electric fields that would produce the desired response were imposed on the structure. The cost function is then defined as the error due to the mismatch between the imposed electrical fields and the ones required by Maxwell's equations. Adjoint sensitivity is then used to adjust the geometry to minimize the error [25].

The optimization problem can be constrained or unconstrained. An additional work by Piggott et al. seeks to include fabrication constraints in an optimization for a photonic spatial mode demultiplexer. Restricting the voxel size to a grid of the minimum allowed feature size was deemed to be too restrictive while the convolution with a smoothing filter was deemed too permissive. The authors opted to constrain the curvature of the structure's outline. They used the level set parameterization with gradient descent to reach local minimum [30].

The boom in neural networks and deep learning made its way to the inverse design world in recent years. There are works using supervised, unsupervised, and reinforcement learning to optimize photonic structures for different applications. The general approach is to design a surrogate neural network that learns the relevant physics. Then, the network is used as part of a design network or to train a generative adversarial network. The deep learning is incredibly powerful, but results are limited to the design space included in the training examples. A lot of data is required to effectively train a network with many weights, and to create a representative design space [40, 47].

Currently, the are far fewer works applying shape optimization to electron emitters or devices that interact with electron trajectories than for photonics. That is probably the case because electron trajectory optimization is far more computationally expensive, and therefore, more difficult to do efficiently.

In this thesis, we will introduce a framework for shape optimization for electron emitters. We will use the framework to optimize the emitter tip in diodes and transistors. Many of the works in photonic shape optimization use the density parameterization like [22], but that is an ill suited method for our applications. It can generate a geometry with many disconnected components and many holes. Our formulation requires a continuous emitting component that is large enough to make contact with in a circuit. We also require the emitter to have one well defined surface from which we can emit electrons. We parameterize the emitter by a collection of point with the the that polygon results from their interpolation as the emitter. Neustock et al. parameterized electron lenses singularly with several parameter points that are interpolated to make polygons defining the shapes. The authors use the adjoint sensitivity method to compute gradients for optimization [37]. Gradient methods are very powerful and converge to local minima quickly, yet we will abandon them in favor of global non-gradient methods. The cost functions we consider here do not lend themselves nicely to gradient calculations because they depend on highly nonlinear emission functions which depend on PDE solutions. We made use of the simulated annealing method because it is well suited for non-convex and irregular optimization landscapes.

Using the optimization framework developed in this work, we are able to reach device designs that achieve a 6-orders-of-magnitude performance improvement compared to the initial geometry in 10,000 optimization steps. For each emission model assumed, we uncover unique geometrical features that enhance the performance of the device on figure of merit of interest.

The remaining chapters in this thesis are organized as follows:

Chapter 2

This chapter describes in detail the optimization algorithm implemented, along with the parameterization of the problem and the constraints. The principles governing the evolution

of the geometry in each iteration are described. The second half of the chapter focuses on the software implementation of the the optimizer and of the PDE solver and the code calculating the emission. Methods there is a discussion of methods that are used to speed up the simulation and optimization process.

Chapter 3

This chapter describes the process of optimizing a diode under different assumptions of emitting mechanisms. It includes a mathematical description of the cost function, an analysis of the runtime, constraint failure, and the results. The results are followed by a dive into the limitations of the modeling, the simulation, and the optimization.

Chapter 4

This chapter describes optimizing a transistor with two different figures of merit. A mathematical formulation is described for each cost function. There is a discussion of heuristics for particle tracking, followed by the results of optimizing the devices for each cost function assuming different emission mechanisms. The chapter concludes with an analysis of the trade-off between the two figures of merit associated and additional limitations.

Chapter 5

This chapter concludes the thesis by describing the work done, highlighting impacts, and provide an outlook for future implementations building on this work.

Chapter 2

Numerical Approaches

This chapter describes the methods and algorithms used in the thesis to simulate the planar nano vacuum channel (PNVC) devices, evaluate their performance, and optimize them. We used a finite element method PDE solver to simulate the electric potential in the space between the device's electrodes. We then calculate the electric field right outside the emitter and use it to evaluate the emission current at the surface. That information was used to evaluate the figure of merit of the candidate geometry. We initially used an out of the box simulated annealing method as the optimizer. However, it did not offer the full range of functionality and efficiency we sought. Therefore, we wrote our method with an aim towards efficiency, flexibility, and debugging abilities.

Similarly to Neustock et al. [37], we parameterize the emitter by a collection of points on the surface, but unlike the authors, we use Cartesian coordinated instead of radial ones. This choice requires us to restrict the movement of the parameter points as we evolve the structure.

Particle trajectory tracking is required to evaluate the performance of gated devices. In this work, we implement a heuristic to particle tracking and a bisection search to speed up the computation.

2.1 Optimization

Optimization problems generally consist of the following components: decision variables, an objective (or cost) function, and an algorithm. The algorithm updates the decision variables to a set of new values that improve the figure of merit. The way the algorithm works heavily depends on the structure of the problem, the objective landscape, and the desired level of performance for the final solution. Rarely can optimization problems be solved analytically. That is the case when the objective function has an analytical closed form dependence on the decision parameters, has well defined gradients, and a small number of dependant decision parameters. Most optimizations, however, require an iterative process of numerical calculations. That is certainly the case with shape optimization problems, which have a large design space and and an objective function that depends on a PDE solution.

2.1.1 Algorithm

As discussed in section 1.2, there are two leading methods of shape optimization: gradient methods and evolutionary methods. Gradient methods typically converge rather quickly when the objective landscape is smooth. In each iteration, the direction of steepest change in the objective function is calculated, and the decision variables are updated with a step along that direction. When the problem is a minimization problem, the objective function is refereed to as a cost function and the method is called gradient descent. The optimization terminates when the gradient is zero or when a maximum number of iterations is reached. The gradient method finds globally optimum solutions when the optimization landscape is convex. A convex function $f(x)$ is defined as one where a line drawn between any two points on the function will always be above the function [16]. Mathematically, this definitions can be expressed as:

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2) \qquad \text{for } 0 \leq t \leq 1. \tag{2.1}$$

Most problems, however, are non-convex and have multiple local optimum solutions. In those cases, the gradient method converges to the local optimum closest to the starting point.

When the number of decision parameters is very large, as is the case in shape optimization, calculating the gradient using the finite difference method becomes computationally infeasible. That is because the method requires $n + 1$ PDE solutions for a problem with n decision parameters. The adjoint method was introduced as an approach to calculating performance gradients efficiently [35]. Stochastic gradient descent was proposed as a way to calculate approximate gradients using fewer computational resources and is widely used in neural network training [8].

Evolutionary algorithms typically take longer to converge compared with gradient methods. There are many flavors of evolutionary methods, but they all include an exploration component and an improvement component. The exploratory component jumps around the objective landscape, attempting to discover more of it, while the improvement component seeks to improve the objective function and pull the decision variables towards a nearby local optimum. There is a trade off between the exploratory component and the downhill one. The more an algorithm explores, the more likely it is to find the true global optimum, but it comes at the price of a slower convergence time. The more an algorithm tried to improve the objective, the faster it will converge to the closest local optimum, but it might miss the true global optimum.

The genetic algorithm was inspired by biology. It works by creating random mutations of the decision variables' values and evaluating the "fitness" of the mutations. The most fit mutations survive, reproduce, and mutate further, while the less fit mutations "go extinct" [36]. The mutation is the exploratory component, and the pruning of unfit mutations is the improvement component of the geometry.

Simulated anneal was inspired by the annealing process of metals. Initially, the metal is very hot and the atoms in it move freely in random directions. As the metal cools, the atoms start moving towards a low-energy configuration in the metal. The probability of atoms moving towards a high-energy arrangement drops exponentially as a function of temperature [55].

The optimization problem introduced in this thesis is non-convex. We can demonstrate that by assuming that there is one unique emitter shape that is optimal. That optimal shape can be represented in many ways by the parameter points. If we take one optimal

arrangement of parameter points, we can generate another arrangement by having each parameter point transfer its value to the neighboring point in a counterclockwise way. Even though the values of all of the parameter points have changed, the shape remains identical and so does the performance. To further complicate things, our landscape is also full of holes. A hole is a region where the objective value is undefined. In our case, holes consist of regions where the values of the decision parameters generate an invalid polygon, such as an open curve or a self intersecting one.

Given the non-convex and irregular nature of our problem, we found the simulated annealing method to be the most fitting. Algorithm 1 outlines the algorithm that we used.

Algorithm 1 Simulated annealing algorithm adapted from [55]

`iter_num` = number of iterations
T_0 = initial temperature
x_0 = initial design
$x^* \leftarrow x_0$
for t = 1 to `iter_num` **do**
  $x_{\text{test}} \leftarrow \text{PERTURB}(x_{t-1})$ ▷ Generate a random move in x_{test}
  $\Delta E \leftarrow \text{cost}(x_{\text{test}}) - \text{cost}(x_{t-1})$ ▷ change in objective or "energy"
  if $\Delta E < 0$ **then** ▷ The move is in the desirable direction
    $x_t \leftarrow x_{\text{test}}$
    if $\text{cost}(x_t) < \text{cost}(x^*)$ **then**
      $x^* \leftarrow x_t$
    end if
  else ▷ If the perturbation does not reduce the cost
    Draw a random variable $r \sim \text{Uniform}(0, 1)$
    if $r > p = e^{-\Delta E/T_t}$ **then** ▷ Accept with probability $p = e^{-\Delta E/T_t}$
      $x_t \leftarrow x_{\text{test}}$
    end if
  end if
  return x^*
end for
Require: $T_t < T_{t-1}$

We start with an initial solution x_0, and evaluate how well the solution performs with respect to a defined objective function. Then, for each iteration, the optimizer creates a candidate solution x_{test} by perturbing the current solution, and evaluates the objective function at x_{test}. There are many ways to perturb the solution; one method is suggested in algorithm 2. If the change in cost is negative, the x_{test} solution is accepted. However, if the

change in cost is positive, there still is a chance of accepting the solution. The optimizer randomly decided whether or not to accept the x_{test} solution by drawing from an acceptance probability distribution that depends on temperature, such as $p = e^{-\Delta E/T_i}$ or $\frac{1}{1+e^{\Delta E/T_i}}$. We require the temperature to decrease between iterations, resulting in a relatively higher probability of a counterproductive solution being accepted at the beginning of the simulation, when the system is hot and in the exploration phase. As the simulation cools down, the probability of accepting a counterproductive solution decreases to zero. Throughout the optimization we keep track of the best candidate seen thus far in a variable x^*. Eventually, when the maximum number of iterations is reached, the algorithm returns x^* as the final solution.

2.1.2 Parameterization

We parameterized the emitter with a set of n points in 2D space with $p_i = (x_i, y_i)$ for $i = 1, \ldots, n$. We used a cubic spline interpolation through the parameter points to defines the outline of the emitter as a closed curve in 2D. The spline is an easy way to enforce smoothness on the structure, which is required for fabrication. Our original parameterization consisted of 88 points evenly spaced around the emitter. After it became apparent that over-parameterization might lead to issues, we introduced a second parameterization consisting of 36 points. The reduced parameterization has a high density of points near the tip, and a lower density further out. Both parameterizations are visualized in fig. 2-1.

As mentioned in the previous chapter, parameterizing a shape by points on the surface is a lesser common approach in shape optimization. For many applications in photonics and mechanical structures, the parameterization of choice consists of pixels or voxels with a decision variable equal to the density of the material occupying that space [22, 25, 32]. That implementation is well suited for photonics, but is ill suited for our application which requires us to have a continuous shape with a defined outline.

Neustock et al. use a parameterization consisting of points that define a closed curve [37]. Unlike our implementation, they use polar coordinates where the angular variable remains fixed while the radial one changes. Their parameterization avoids the issue of self intersecting polygons, but also forbids hook-like features which might be beneficial in some applications.

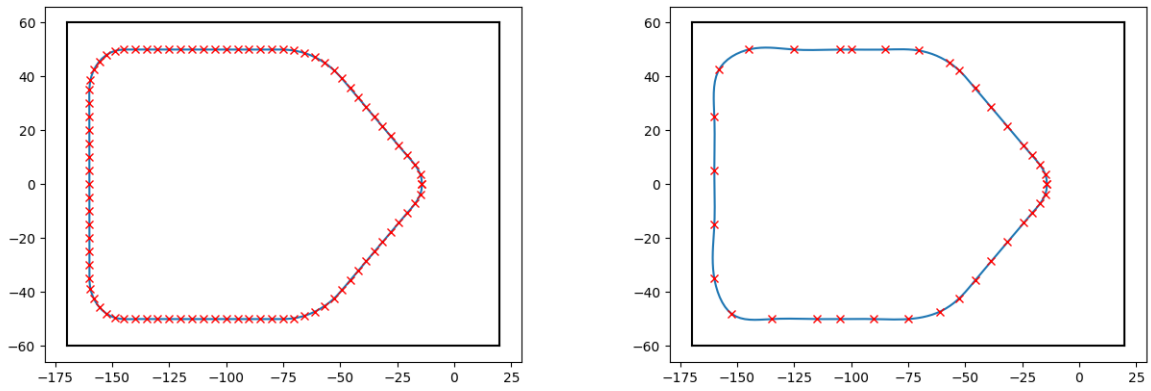

Figure 2-1: Parameterization of the optimization problem for initial geometry. Parameter points are shown in red and the spline interpolation is shown in blue. (left) The original parameterization consisting of 88 points allows for a lot of flexibility, which can both be beneficial and cause issues. (right) The reduced parameterization consisting of 36 points creates a more rigid structure, which can both be restricting and prevent shape irregularities that crash the solver.

Our implementation allows both x and y to vary for each point, but as a result we must restrict the movement of points to maintain shape feasibility. More on that in section 2.1.3.

2.1.3 Annealing

In out-of-the-box simulated annealing optimizers, the default annealing step randomly perturbs the parameter points without constraints or requirements for certain parameters to be tied to each other. In those implementations, the (x_i, y_i) values of each point are treated as independent variables, which is ill suited for our implementation. When x_i and y_i move independently of each other, the parameter points quickly arrive in a configurations that forms an invalid polygon, and therefore, is non physical.

To increase the probability that the `PERTURB` method attempts valid parameter point arrangements, we restrict each parameter points, (x_i, y_i), to move along the line normal to the surface going through that point. We find the line normal to the surface by taking cross product of the tangent vector with the out of plane unit vector. We average the two

approximate cross product vectors at each point for better accuracy as follows:

$$v_1 = ((x_i, y_i, 0) - (x_{i-1}, y_{i-1}, 0)) \times (0, 0, 1) \tag{2.2}$$

$$v_2 = ((x_{i+1}, y_{i+1}, 0) - (x_i, y_i, 0)) \times (0, 0, 1) \tag{2.3}$$

$$v_n = \frac{v_1}{2} + \frac{v_2}{2}. \tag{2.4}$$

We perturb each point by a random amount q_i drawn from a uniform distribution centered at zero. If q_i is positive, the point will be perturbed outwards along the normal. Otherwise, the point will be perturbed inwards along the normal. The of variance the uniform distribution from which we draw, can either depend on the temperature parameter or be a constant. Algorithm 2 outlines the perturbation method applied to the full set of parameter points.

Algorithm 2 Annealing perpendicularly

function PERTURB(x, y)
  for i = 2 to n-1 **do**
    $\delta_1 \leftarrow (x_i - x_{i-1}, y_i - y_{i-1}, 0)$
    $\delta_2 \leftarrow (x_{i+1} - x_i, y_{i+1} - y_i, 0)$
    $v_1 \leftarrow \delta_1 \times (0, 0, 1)$ ▷ Cross product
    $v_2 \leftarrow \delta_2 \times (0, 0, 1)$ ▷ Cross product
    $q_i \sim \text{Uniform}(-1, 1)$
    if Constant_pert **then** ▷ Constant variation in perturbation enabled
      $\epsilon \leftarrow \frac{1}{2\|\delta_1\|}$
    else ▷ If not
      $\epsilon \leftarrow \frac{\sqrt{T}}{5\|\delta_1\|}$
    end if
    $x_{\text{test},i} \leftarrow (x_i, y_i) + \epsilon \frac{v_1 + v_2}{2}$
  end for
  return x_{test}
end function

While algorithm 2 presents the method in loop-form for readability, our implementation was done with array operations for efficiency. The code implementation with arrays is found in appendix A.2.

Bounds and Constraints

There are a lot of manufacturability constraints for PNVCTs limiting the range of allowed shapes for emitters. Some constraints come from the state of technology, like resolution of contemporary electron beam lithography (EBL) tools, while others are more fundamental physical artifacts like the typical grain size of metal when it evaporates and the size of atoms. To account for our finite resolution, we do not include explicit constraints. Instead, we encode them in the distance between parameter points in the initial geometry and in the spline interpolation.

In addition to that, the are also simulation constraints requiring the emitter to be contained in the simulation domain and forbidding self-intersecting polygon shaped emitters. We therefore establish upper and lower bounds on the x and y values of each simulation point. If the `PERTURB` function attempts to push any point beyond the bounding box, the corresponding x or y value will be set to the one on the surface of the box. The implementation of this constraint can be found in appendix A.2. To reduce the probability of self-intersecting polygons occurring, we require the `PERTURB` method to move points along lines perpendicular to the surface. When self-intersecting polygons do occur, the electrostatic simulation crashes. We implemented error handling to prevent the optimization process from exiting every time the simulation crashes. The self-intersecting constraint is handled implicitly by the except block of the try-except.

2.1.4 Stepping

In every iteration, the optimizer sends x_{test} to the cost function to evaluate the performance of the perturbed solution. If the perturbation reduces the cost, the perturbed solution becomes accepted, and is then stored by the current solution variable. If x_{test} is the best solution seen thus far, it gets stored in x^*. If the perturbation increases the cost, the algorithm draws from the acceptance probability, and decides stochastically whether or not to accept it. This step is described by algorithm 1 and the code implementation for it is in appendix A.2.

2.1.5 Cost

The figures of merit that are relevant to us depend on currents collected by either the collector or the emitter. To calculate the currents, we solve a full electrostatic simulation, calculate the emitted current density at the surface of the emitter, and then track current trajectories through space. We used FEniCS [29] as our PDE solver to do to do a full electrostatic simulation. We calculate the emitted currents using a few different emission models that depend on the electric field. To calculate the current at the collector or gate, we approximate the current trajectories rather than implement a full particle in cell simulation. The simulated annealing algorithm is well suited for minimization problems, but not for maximization problems. We, therefore, formulate all of our objective functions as cost functions. When a figure of merit is desirable, we multiply it by -1 to turn it into a cost. Sections 3.1 and 4.1 describe the cost functions that were used in this work and why they are of interest to minimize. Section 3.1.1 describes the different emission mechanisms considered for current generation. See section 2.3 for implementation details.

2.1.6 Temperature

Temperature is decreasing during the optimization process. It is natural to stick to the physical inspiration of the optimization and decrease the temperature exponentially, as described by Newton's law of cooling. However, that might decrease the temperature too quickly and prevent the algorithm from exploring. For that reason, we implemented several options for the temperature evolution to allow the user to choose their desired cooling process. The implemented options include a decrease as $1/t$, a linear decrease, and an exponential decrease. The results presented in chapters 3 and 4 were generated by an optimization with a linear cooling. See appendix A.2 and table B.1 for reference.

2.2 Optimizer Implementation

There are some existing out-of-the-box solutions for simulated annealing optimizations. We naturally, looked at existing solutions and evaluated them before deciding to write our own.

The simulated annealing method included in Scipy [41] is convenient for usage because it is natively embedded in python and does not require cross-language translation. However, the method comes with little user enabled flexibility to change the perturbation method or to include bounds on the solution. We therefore, turned to MATLAB which allows more flexibility with user defined annealing, user defined cost functions, and easy integration of bounds.

2.2.1 MATLAB Implementation

This section can be skipped by the reader because all the code described in it has been deprecated and replaced with significantly more efficient code. There is, however, entertainment value and perhaps, a lesson to be learned from the author's MATLAB journey. The motivation for using the MATLAB Simulated Annealing function was driven by a desire to make as much use of packages already written by others. It was a noble desire, after all, as it should have given us more time to focus on the applications of the code and the results of the optimization. However, the opposite was accomplished, teaching us that noble intentions alone are rarely enough.

The FEniCS solver has a python interface, which required us to find a way to communicate between the MATLAB layer and and python layer. MATLAB is equipped with a function that runs python scripts, `outvars = pyrunfile(file, inputs)`. However, the variable passing between the layers was not built for arrays. Another issue that presented itself was that the C++ version that came with MATLAB conflicted with the one used by FEniCS. To get the code to run, downloaded and built a different subversion of C++14. Even with that fix, there were issues with the direct communication between MATLAB and FEniCS. We finally managed to implement the optimization by using the MATLAB function that runs bash scripts. The bash script ran a python script containing the FEniCS simulation code and the cost function evaluation. Variable passing between the different layers of the code was enabled by input from and output to external files. See fig. 2-2 for a detailed visualization of the software stack.

Because the objective function ran a new python instance for each iteration of the optimizer, simulator crashes were undetectable. This feature made debugging an impossible

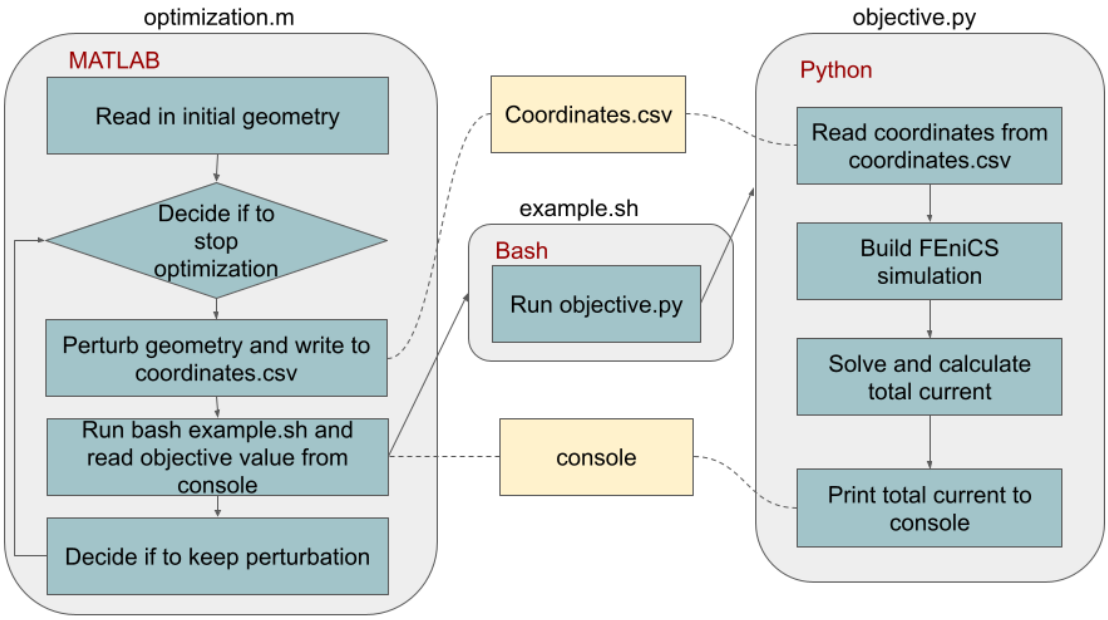

Figure 2-2: Optimization software stack using the simulated annealing method in MATLAB. The MATLAB container acts as the main control. It starts by reading in the initial geometry from a file and evaluating its performance. Then, the optimization loop begins. In each iteration, the geometry is perturbed, then the cost function is evaluated by writing the test geometry to a file and running a bash script that calls a python script that reads the geometry from a file and runs the FEniCS simulation on it. After the FEniCS simulation is complete, the cost function is evaluated and the value is printed to the console. MATLAB reads the cost function value from console and passes it to the "decide if to keep" block. The optimization loop terminates when the iteration number reaches the maximum iteration value. Then, the final geometry and the cost function value in each iteration are written to external files.

task. The back and forth input and output from and to external files increased the run time significantly. For those reasons, we eventually decided to build our own in-house simulated annealing optimization method.

2.2.2 Python Implementation

This section describes the optimization software stack written entirely in python which was used to generate most of the results presented in this thesis. The code is available on github at https://github.com/AdinaBechhofer/Simulated_annealing.

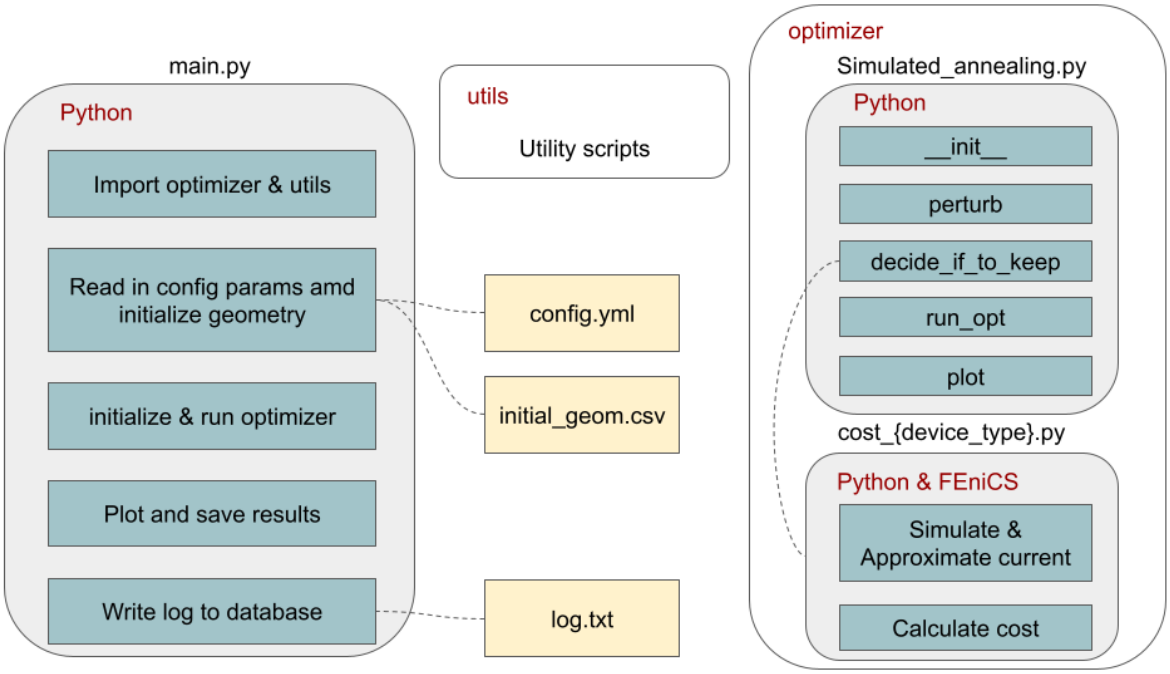

Figure 2-3: Simulated annealing optimization software stack in python. The code in main.py is the controller; it reads the parameters from config.yml and the initial geometry, initializes the optimizer, saves the results, and adds the optimization metadata to a database in log.txt. The optimizer class in Simulated_annealing.py contains an `__init__` method that initializes the class and a `run_opt` method that runs the optimization. The `perturb` method perturbs the geometry. The `decide_if_to_keep` method calls the relevant cost function on the test geometry and decides if to accept it.

The simulated annealing class was written to replace the out-of-the-box optimizer, while incorporating greater flexibility and debugging abilities. Additional code was written as supporting architecture to make the optimization user friendly and reusable. A configuration file contains all of the user controlled simulation and optimization parameters for easy

customization. The main file acts as the main controller; it reads the parameters from the configuration, initializes the optimization, and writes the simulation results and metadata to external files after the optimization concludes. Each device type has a cost function python file associated with it where the FEniCS electrostatic simulation is run and the cost value is calculated. The utils folder contains utility script that are useful for analyzing the optimization results and metadata. Below is a more detailed description of the code in each file and its functionality.

config.yaml

Many parameters control the optimization procedure and the finite element simulation embedded in the cost function. The parameters range from the size of the gap in the simulation to the number of optimizer iterations. It can be incredibly tedious for a user to locate and change all of the parameters of interest where they naturally appear in the code. Therefore, we concentrated all of the parameters that a user might want to change in the `config.yaml` file which gets read by the main script and passes the relevant parameters to the optimizer. The parameters in configuration are organized into 2 categories; optimization parameters and geometry parameters as described by tables B.1 and B.2.

main.py

The main.py file contains the main entry point to the optimization process. This file imports the simulated_annealing module and additional relevant utility modules. It reads the configuration parameters from config.yml, and starts a loop of N iterations, where N is a parameter specified in config.yaml by the `optimization runs` parameter in the optimization section. In each iteration of the loop, the initial geometry is read from a file specified in the config, a simulated annealing object is created, and the optimization is run. After the optimization terminates, relevant figures are generated and all the optimization metadata is written to a log file where it is encoded as a json string. The optimization metadata includes the parameters specified in the config file along with information about the runtime and constraint violation rate of the optimization (constraint violation rate refers to the proportion of total geometries attempted that crash the simulator). If the `log errors` flag is set to

true in the config, a sample of the bad geometries will be saved along with the error message that caused them. That information is useful for analysis and debugging.

Simulated_annealing.py

The simulated annealing class contains the optimization framework. The simulated annealing algorithm works by perturbing the current solution, evaluating the cost of the perturbed solution, and deciding whether or not to adopt the perturbed solution. The optimization code along with relevant parameter tables can be found in appendix A.2.

Our implementation provides the user with flexibility to customize their optimization and functionality that extends beyond what is offered by out-of-the-box methods. Our method has a defined custom annealing, which the user can control by choosing the variance of the the random distribution of the perturbation steps. The user can set the distribution variance to be either constant or temperature dependant. The user can also choose to smooth out the perturbation with an averaging window. We provide the flexibility for the user to change the initial temperature and the temperature cooling with respect to optimization iteration. We allow the user to easily access all the past objective values seen by the optimizer. Access to past values is useful for a principle component analysis (PCA) method that is under development at the time of writing these lines (see section 3.7.4 for more details). Additionally, we include several plotting methods for the user to easily visualize the results of the optimization. Arguably most importantly, we provide an easy way to log simulator crashes and debug errors.

error_analysis.py

The code in error_analysis.py is used to analyze simulation crashes. When the `log errors` parameter is set to True, the optimizer saves copies of geometries that cause the simulator to crash. The code in error_analysis.py reads all of the "bad" geometries from a specific directory and attempts to simulate them. We implemented three try-except blocks along the simulation code to detect which step is the culprit responsible for the simulation to crash. After running through all the geometries, the results are aggregated into crash statistics, which are reported to the user. The constraint violation statistics in section 3.2 were compiled

using this code.

2.3 Simulation

The figure of merit in our optimization depends on the electrical currents collected in various parts of the device due to field-driven emission. Calculating those currents requires simulating the electrostatics in the device, calculating the emission current, and approximating current trajectories. Many different simulation software packages for electrostatics exist, one of such is FEniCS. FEniCS is a open source scientific computing engine and differential equation solver written as a collaboration between people at The University of Chicago and Chalmers University of Technology[23]. FEniCS uses the finite element variational method to solve partial and ordinary differential equations. FEniCS can also be used for computational linear algebra, but in this work it was used only for differential equations.

Being open source, FEniCS is completely free to use. It also does not require a licence. Additionally, it allows access to all layers of code and objects, allowing us to manipulate the solution matrices and perturb the mesh directly. This feature gives us flexibility when interacting with the simulator and sets the stage for dramatically speeding up the optimization with adjoint like methods like we discuss in section 5.4.2. In comparison, commercial simulators such as Comsol [10] and Lorentz [53] require user licences which can get pricey, reaching upwards of 4000 USD without maintenance fees. They do allow scripting around them, but have a limited array of predesignated functions, and they do not give direct access to the underlying matrices. We use FEniCS to solve the Laplace equation in the gap between the electrodes of the device and to calculate the electric field near the edge of the emitter. We then use our own code to calculate the emission current and approximate where it ends up.

2.3.1 Laplace Equation

Solving differential equations in FEniCS generally follows a procedure similar to the one described in this section, but different forms are also possible. In this work, we solve Laplace's equation $\nabla^2 U = 0$ with non trivial boundary conditions to obtain the electric field outside the

emitter and the electric field lines. Sample setup code can be found in appendix appendix A.1.

Setup

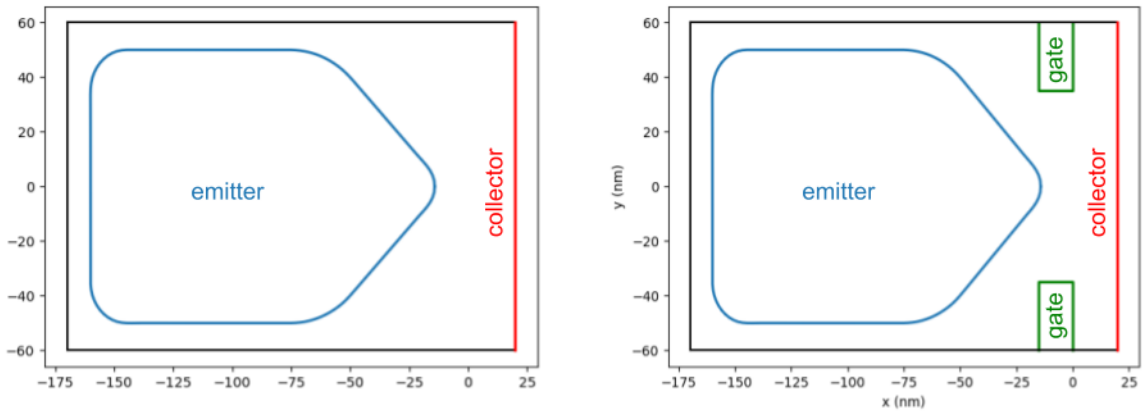

Figure 2-4: Simulation domain for ungated (left) and gated (right) devices. Domain is enclosed in the box with black edges. The emitter is traced in blue, the collector which is also the right edge of the simulation is marked red, and the gates (in the gated device) are green.

We start by defining the solution domain as a rectangle of length 190 nm and width 120 nm. The right edge of this domain rectangle is the collector ($x = 20$ nm). The emitter is a polygon with vertices defined by the spline interpolation of the parameter points. A polygon in FEniCS is defined by a sequence of point objects ordered in a counter clockwise manner with the last point exactly equal to the zeroth point. In a simulation of a gated device, the gates are rectangles of length 25 nm and width 15 nm centered at $y = 47.5$ nm and $y = -47.5$ nm and placed between the collector and the emitter in the horizontal direction. In FEniCS, the domain can be described as the addition or subtraction of geometrical objects. Here, we subtract the emitter (and gates if relevant) from the large domain rectangle, so that the resulting domain simulates the potential in the gap between the electrodes of the device. We generate a mesh for the domain by passing the domain of interest and the desired mesh resolution to meshing function. See appendix A.1 for startup code.

Boundary conditions are set in FEniCS by defining a function for each boundary. The function takes an (x, y) pair as input and returns a Boolean indicating whether or not (x, y) is on that specific boundary. We set Neumann condition on the upper, left, and bottom

edges of the simulation (black edges in fig. 2-4). Neumann conditions have the same effect as having a mirrored simulation cell reflected across the boundary. Using Neumann conditions makes sense for PNVCTs because they are fabricated and operated in large arrays. We set Dirichlet boundaries to the collector (red edge in fig. 2-4) and to the emitter (blue). The boundary condition for the emitter is slightly more complicated than a straight edge and it requires an interpolation between all the polygon vertices. The code in appendix A.1 verifies whether or not an input point is on the boundary of the emitter with tolerance `tol`. The code as is written now loops over each one of the emitter polygon line segments and checks if the point is on that line segment. Looping is inefficient in python and should probably be replaced with array operations.

Variational solution

The finite element method discretized PDEs and projects them onto a basis where they are solved using the variational method. There are many bases popular for finite element, they are typically different polynomials of varying degrees. Here we use the continuous Galerkin (CG) polynomial of order 1 as the element basis. Each basis element, ϕ_i is a triangle with height 1 on the node i, and zero on other nodes. It can be written mathematically as

$$\phi_i(n_j) = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{for } i \neq j. \end{cases} \tag{2.5}$$

The value of $\phi_i(x)$ at any point between nodes is a linear interpolation using the closest node points. When the solution is represented in this basis, the height of triangle is the value of the solution at the node. We can think of this basis as a linear averaging scheme for all mesh points that is not a node point.

To solve

$$-\nabla^2 u = f, \tag{2.6}$$

we use a trial function, u and an arbitrary test functions v. After applying the variational

method, we end up with

$$\int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx. \tag{2.7}$$

Now we apply the finite element discretization

$$u = \sum_j a_j \phi_j \text{ and } v = \sum_j b_j \phi_j, \tag{2.8}$$

and substitute that into eq. (2.7)

$$\int_\Omega \sum_i \sum_j a_i b_j \nabla \phi_i \cdot \nabla \phi_j dx = \int_\Omega \sum_j b_j f \phi_j dx. \tag{2.9}$$

Note that the test and trial functions do not have to come from the same function space or have the same finite element basis, but in this case it is convenient to do so and we have no incentive to choose otherwise.

We set $a = \int_\Omega \sum_i \sum_j a_i b_j \nabla \phi_i \cdot \nabla \phi_j dx$ and $L = \int_\Omega \sum_j b_j f \phi_j dx$ and solve $a = L$. The solution is attained by assembling matrices for a and L, and computing a matrix solution [29]. We select a Krylov-Solver for the matrix solution because using a Krylov subspace method is generally faster than applying a brute force matrix inversion. See appendix appendix A.1 for the code implementation in FEniCS.

2.3.2 Current emission

In chapters 3 and 4, we consider different models for the current emission in NVC devices based on previous works in literature. For many years, it was unanimously understood that emission from tips in vacuum was perfectly described by Fowler-Nordheim tunneling [20]. However, recent works have shown that there are voltage ranges in which other emission mechanisms are dominant [50]. To implement the different emission models considered in chapters 3 and 4, we wrote a Fowler-Nordheim emission function and a Schottky emission function with parameters that can be adjusted for the different models.

For the Fowler-Nordheim function, we used the one dimensional version due to its relative simplicity compared with the higher dimensional version. We assume that modeling electrons

as tunneling through a one dimensional potential barrier works sufficiently well for small area chunks that are relatively flat. The current density due to Fowler-Nordheim emission is

$$J = \alpha_{FN}\gamma^2|E|^2 \exp\left(-\frac{\beta_{FN}\phi^{3/2}\left(1 - \gamma\eta_{FN}|E| + \frac{1}{6}\gamma\eta_{FN}|E|\ln(\gamma\eta_{FN}|E|)\right)}{\gamma|E|}\right), \tag{2.10}$$

where $|E|$ is the magnitude of the electric field normal to the surface, γ is an arbitrary enhancement factor, $\alpha_{FN} = e^3/(8\pi h)$, $\beta = 8\pi\sqrt{2m_e}/(3eh)$, $\eta_{FN} = e^3/(4\pi\epsilon\phi^2)$, ϕ is the metal's work function, e is the electron charge, m_e is the electron mass, ϵ is the permittivity of free space, and h is Plank's constant.

For Schottky emission, we use a current density of

$$J = A^* T^2 \exp\left(\frac{-\phi + \frac{e}{2}\sqrt{\frac{e\gamma|E|}{\pi\epsilon}}}{k_B T}\right), \tag{2.11}$$

where T is the temperature, k_B is the Boltzmann constant, A^* is a fitting parameter, and all other quantities are as defined above.

To get the electric field for the emission functions, the gradient of of the potential is needed $E = -\nabla u$. After obtaining the potential from the finite element method, we can calculate the gradient numerically with finite difference. However, given that the basis for u is a first order polynomial, finite difference will result in a gradient composed of piece-wise constant functions. It is more robust to project the gradient into a vector function space of an appropriate basis, and then extract the value of the gradient at a point. We use the framework provided by FEniCS to project the gradient and to get its value at a point. The code implementation can be found at appendix A.1 and the code to calculate the emission is likewise in appendix A.1.

We need to evaluate the gradient of the potential at the surface of the emitter, but the gradient is undefined exactly on the boundary of the simulation. For that reason, we calculate the gradient at an ϵ distance away from the boundary of the emitter. To find the points at which to calculate the gradient, we calculate the outward pointing normal from each polygon segment and take a step of size ϵ along that direction. Calculating the outward pointing normal is done by taking the cross product between the tangent vector,

$(x_i, y_i) - (x_{i-1}, y_{i-1})$, and the out of the plane normal. Initially, this was done in a loops which is inefficient in python. In a later iteration of the code, we replaced the loop with Numpy array operations. See code in appendix appendix A.1.

After obtaining the gradient of the potential, we need to check its direction relative to the surface of the emitter. When the electric field is pointing outwards from the surface, electrons will be emitted. When the electric field is pointing into the surface, emission is suppressed. The emission functions that we wrote do not take the field direction into account; they only consider magnitude. Therefore, we must ensure that the electric field is pointing outwards before calculating the emission current. We already have the outward pointing normal from the cross product calculation. We compute the dot product of the gradient with the outwards normal; if the result is positive, we consider that field for current generation. Otherwise, we disregard the field in that spot. To get the total current, we calculate the area for each segment and integrate all the current densities over the total surface area of the emitter.

At the time of writing these lines, the code for integrating the current density shown appendix A.1 is still in loop form. It could be made more efficient by replacing the loop with Numpy arrays.

Gated devices

In ungated devices, we assume that the current emitted from anywhere on the surface of the emitter is swept into the collector by the electric field. In gated devices, there are multiple targets that can collect current. To evaluate the transistor's operation, it is critical to distinguish between the current hitting the collector and the current arriving at the gate. Properly simulating electrons being emitted and traversing the gap is computationally expensive. Particle-in-cell (PIC) simulators iteratively update the position and velocity of each electron in the simulation along with the electric potential due to the charge of the electrons in space. In this work, we create a fast heuristic for particle tracking by approximating current trajectories with electric field lines. The approximation assumes that electrons have zero mass, and therefore, zero momentum. As a result, the electrons follow the electric field lines starting from the emission point until they reach an electrode surface or a boundary of the simulation. Algorithm 3 calculates the electric field line starting at

(x^*, y^*).

Algorithm 3 Tracing electric field lines

```
function TRACE-FIELD-LINE(∇u x, y, x*, y*, num_iter)
    dot_product← (x* − x, y* − y) · ∇u   ▷ Dot normal to surface with potential gradient
    if dot_product > 0 then
        x_new ← x* + ∇_x u(x, y)
        y_new ← y* + ∇_y u(x, y)
        for q = 1:num_iter do
            if reached_collector(x_new, y_new) then
                return "collector"
            else if reached_gate(x_new, y_new) then
                return "gate"
            else if reached_edge(x_new, y_new) and x_new > x_gate then
                return "r-edge"
            else if reached_edge(x_new, y_new) and x_new < x_gate then
                return "l-edge"
            else if intersect_emitter(x_new, y_new) then
                return "zero"
            else if ‖∇u(x, y)‖ ≤ tol then                    ▷ If gradient = 0
                if (x* − x_new, y* − y_new) < 10 then        ▷ In the vicinity of the emitter
                    return "zero"
                else
                    p ← (x_new − x_prev, y_new − y_prev) ▷ Away from emitter use momentum to update trace
                    (x_new, y_new) ← (x_new, y_new) + p
                end if
            else
                (x_prev, y_prev) ← (x_new, y_new)
                (x_new, y_new) ← (x_new + ∇_x u(x_new, y_new), y_new + ∇_y u(x_new, y_new))
            end if
        end for
    else
        return "zero"
    end if
end function
```

For each emission point, we calculate the electric field line by iteratively stepping in the direction of the electric field. The initial point is the emission point on the emitter's surface. From there, we iteratively calculate the gradient, take a step in the direction of the gradient, and calculate the gradient again in the new location until a stopping condition is met. The obvious stopping conditions are reaching the collector or reaching the gate. When either one

of those is met, we break out of the loop and attribute the current emitted at the origin point to the collector or gate respectively. A less obvious condition is a field line escaping through the non-collector side of the simulation or looping back to the emitter. When one of those conditions are met, we stop the loop iteration, but discard the emitted current because it will not be detected at the collector or at the gate. When the gradient vanishes, we do not immediately terminate the loop. If the gradient vanishes very close to the emitter, we can assume that no current would be emitted. However, if the gradient vanishes after allowing the emitted electrons to gain momentum in the electric field, we can assume that the electrons will preserve their momentum and will keep traveling along a straight line until they collide with the gate, collector, or different edge of the simulation. The code for this calculation can be found in appendix A.1.

To calculate the collector current and the gate current, we can iterate through all the points on the surface of the emitter and calculate the trace for each surface point, but that formulation would be rather slow. We realized that we do not actually need to calculate all of the field lines; we just need to find the last line to end up on the collector and the first line to reach the gate. Because of the way that the field lines are ordered, we can implement a binary search and find the desirable information much faster. Algorithm 4 describes the binary search for the trajectories originating at points with $y > 0$. The result of algorithm 4 is an index *middle*. The trajectories indexed by a number lower than *middle* are counted as collector current, while the trajectories with index higher than middle get counted as gate current. The code in appendix A.1 contains two searches, one for trajectories originating at points with $y > 0$ and one for trajectories with origin points with $y < 0$.

A slight complication arises from having array indices that correspond to trajectories that do not leave the emitter or that fly off the edge of the simulation. When we encounter those indices, we skip them without bisecting the array. We use a variable to keep track of the skipped indices and use it in the bisection search. That way, we avoid getting stuck in those indices, which could result in an infinite loop.

Algorithm 4 Binary search for trajectory tracing

let $high$, low ▷ Trajectory indices
$x^*, y^* \leftarrow$ get_ppoints(x, y) ▷ x, y are on the surface, x^*, y^* are ϵ away from surface
$middle \leftarrow \left\lfloor \frac{\text{high+low}}{2} \right\rfloor$
$skipped \leftarrow 0$
$found \leftarrow$ False
while $(high - low) > 1$ **and not** found **do**
  $i \leftarrow middle$
  $dest \leftarrow$ TRACE-FIELD-LINES$(\nabla u, x_i, y_i, x_i^*, y_i^*, \texttt{num_iter})$
  if $dest =$ "gate" **or** destination = "l-edge" **or** $middle = high$ **then**
    $high \leftarrow middle - skipped$
    $middle \leftarrow \left\lfloor \frac{high+low}{2} \right\rfloor$
    $skipped \leftarrow 0$
  else if $dest =$ "collector" **then**
    $low \leftarrow middle$
    $middle \leftarrow round \left\lfloor \frac{high+low}{2} \right\rfloor$
    $skipped \leftarrow 0$
  else if $dest =$ "edge-r" **or** $dest =$ "zero" **then**
    $middle \leftarrow middle + 1$
    $skipped \leftarrow skipped + 1$
    if $middle = low$ **then**
      $found \leftarrow$True
    end if
  else if $middle \geq high$ **or** $middle \leq low$ **then**
    $found \leftarrow$True
  end if
end while

2.3.3 Visualization

FEniCS supports visualization of the PDE solution in a built-in way. It can be done by calling `plot(u)`, where `u` is the solution of the PDE. The resulting figure is a 2D heatmap where the color corresponds to the value of the potential at each pixel of the 2D grid. Figure 2-5 shows a visualization of the potential solution in the gap of a two terminal device with a potential difference of 15 V imposed between the emitter and collector.

We can superimpose additional plots on-top of this plot to convey more information. For example, we can calculate the approximate current trajectories and plot them over the potential plot. In fig. 2-5, the thickness of the trajectory is proportional to the amount of current carried by it. We can also print relevant quantities such as the collector current or total emitted current on empty parts of the figure.

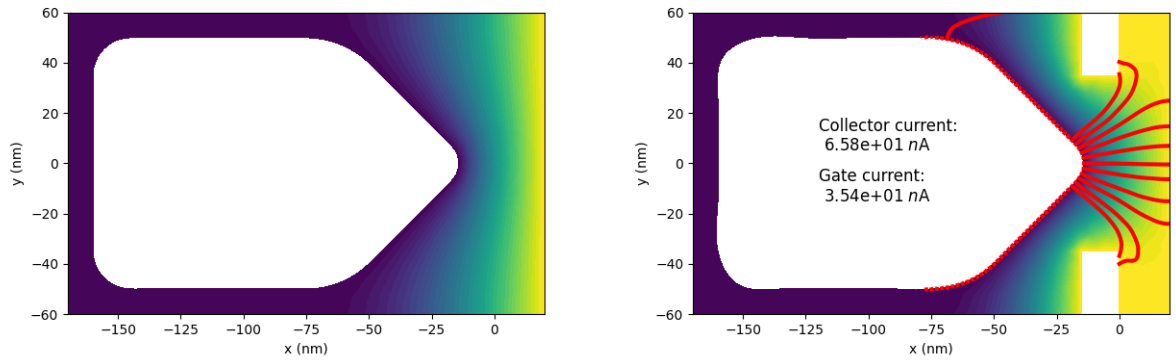

Figure 2-5: Simulation results visualized. (left) Electric potential in the gap between an emitter and a collector plotted as a heatmap with yellow as the highest voltage and purple as the lowest voltage. (right) Electric potential and approximate current trajectories plotted for a gated device with the gate on. The trajectories' thickness is proportional to the current carried in them and the total calculated currents are printed on the figure.

Chapter 3

Global optimization for two terminal devices

In this chapter, we seek to optimize the emitter part in a two-terminal electron emitting device using the stochastic global optimization method discussed in chapter 2. Here, we optimized the emitter's geometry to maximize the current drawn from it. We assume that all of the emitted current will make its way across the gap to the collector, where it can be measured. We considered a few physically motivated emission mechanisms and use each one of them in defining the cost function.

Well-stated constraints are crucial for limiting the scope of optimization to relevant and physically and numerically plausible results. The constraints describe in this chapter are required for the simulation to be able to run. Some of the constraints are explicitly coded into the optimization, while others are implicitly coded and are expressed by the the FEniCS simulation crashing when they are violated. The physical constraints due to fabrication are different than the simulation constraints and are yet to be implemented. We explored different methods to prevent constraints from being violated.

The simulation used to calculate the emission current has limitations that rise from the mesh discretization and from the the 2D electrostatic modeling which make the result differ from experimental data. However, we have reason to believe that the optimization improvements in the simulated model translate into improvements in the real devices.

We see that under some emission conditions the optimizer seeks to maximize the surface

area to emit from, and in others, it seeks to create as many sharp tips as possible to maximize the field enhancement from. These results indicate that the problem of optimizing a diode under the current specifications and constraints is perhaps ill defined. At the limit of infinite iterations, the diode tip will either converge to a flat edge just pushed against the boundary of the collector or to a comb with infinitely sharp teeth, both of which, are problematic for different reasons. More physical constraints are required to make the problem better defined. However, even in the current state, this is a valuable endeavour because optimizing the emitter tip in a diode is an important first step in the process of shape optimization. Having a smaller target of optimization with a relatively simple cost function allows us to thoroughly test our optimization approach and conduct sanity checks before advancing to more complicated and less intuitive designs.

3.1 Cost function (Mathematical Representation)

Finding a figure of merit to optimize is one of the challenges of optimization problems. Most physical systems cannot be reduced to a single number that describes perfectly how well the system is performing. Here, we chose to maximize the total current emitted from the emitter tip of the device (see fig. 2-4 for reference). Using this figure of merit is imperfect because of reasons that will be outlined in the remainder of this chapter, however the advantage of it is that it corresponds to a desirable quality of our devices and is one of the simplest imaginable figures of merit to optimize. Additionally, using it as a figure of merit is an important sanity check. It assures us that we are not being lead astray by a numerical error or a bug in the code. Because there exists some intuition in the field for features that maximize the emission. We can mathematically represent the total emission current maximization objective as:

$$\max_{p_1,\ldots,p_n} \int_{\partial\Omega(p_1,\ldots,p_n)} J(E) \cdot dS \tag{3.1}$$

Such that

$$E = -\nabla\Phi \tag{3.2}$$

$$\nabla^2\Phi = 0 \qquad \text{on} \quad \Omega(p_1,\ldots,p_n) \tag{3.3}$$

$$\Phi = V_0 \qquad \text{on} \quad \partial\Omega_1(p_1, \ldots, p_n) \tag{3.4}$$

$$\nabla\Phi = 0 \qquad \text{on} \quad \partial\Omega_2(p_1, \ldots, p_n) \tag{3.5}$$

Where $p_1, \ldots, p_n$ are the parameter points defining the emitter with $p_i = (x_i, y_i)$ living in 2D space. $J(E)$ is an electric field dependant emission current density. We consider different mechanisms of emission for $J(E)$, which we elaborate on in section 3.1.1. Equation (3.3) is the Laplace partial differential equation which is solved on $\Omega(p_1, \ldots, p_n)$, the simulation domain that is defined by the decision variables. The simulation has mixed boundary conditions with Dirichlet boundary conditions on the emitter and the collector, denoted as the set $\partial\Omega_1 = \{\text{emitter}, \text{collector}\}$ in eq. (3.4). There are Neumann conditions on all simulation boundaries that are not in $\partial\Omega_1$, denoted as $\partial\Omega_2$ in eq. (3.5).

To calculate the total emission current, we had to define a domain using the emitter parameter points. Then, solve the Laplace equation in eq. (3.3), calculate the electric field just outside the emitter, and use the magnitude of the calculated field to calculate the field dependant emission. We considered different physically motivated emission mechanisms, namely Fowler-Nordheim, enhanced Fowler-Nordheim, and a combined Fowler-Nordheim Schottky emission. The different emission mechanisms give rise to different features which we see rising in the results optimization.

3.1.1 Emission mechanisms

paragraph here

Fowler-Nordheim

For a long time the dominant theory in electron vacuum devices relied on Fowler-Nordheim cold-field-driven emission [20]. It is derived from a triangular approximation of the potential barrier. It assumes that the electrons have no thermal spread and are tunneling through the barrier which gets narrower with increasing voltage [24]. The triangular approximation gives rise to the following formulation

$$J_{FN}(E) = \alpha |E|^2 \exp\left(-\frac{\beta\phi^{3/2}}{|E|} v(f)\right) \tag{3.6}$$

$$v(f) = 1 - f + \frac{1}{6} f \ln(f)$$

$$f = \frac{e^3}{4\pi\epsilon\phi^2}|E|,$$

where $\alpha = e^3/(8\pi h)$, $\beta = 8\pi\sqrt{2m_e}/(3eh)$, ϕ is the metal's work function, e is the electron charge, m_e is the electron mass, ϵ is the permittivity of free space, and h is Plank's constant. There are additional formulations that are specific to the emitter geometry and potential barrier shape, making the result precise for each case. We, however, take the coarser formulation because it is more general. Our emitter shape is changing with each iteration and we cannot make any assumptions about the potential barrier shape or the tip geometry. Additionally, we care about finding good relative accuracy for the optimization, rather than absolute accuracy.

Enhanced Fowler Nordheim

Many works assume a field enhancement that is at least 7× the one found by performing the electrostatic simulation of the device [33, 38]. In most cases, an artificial enhancement is assumed in an effort to match experimental data. For the enhanced Fowler-Nordheim simulations we assume a factor of 7 field enhancement. We initially assumed the enhancement factor to implement the method done in literature. Later, we realized that an artificial enhancement might be justified as a way to compensate for the disparity between 2D and 3D simulations. More on this in section 3.6.3, where limitations of 2D modeling are discussed. The enhanced Fowler-Nordheim formulation is

$$J_{EFN}(E) = \alpha |\gamma E|^2 \exp\left(-\frac{\beta\phi^{3/2}}{|\gamma E|} v(f)\right) \tag{3.7}$$

$$f = \frac{e^3}{4\pi\epsilon\phi^2}\gamma|E|,$$

where $\gamma > 0$ is the enhancement factor, and all other quantities are as describe above.

Fowler-Nordheim + Schottky

In work done by Turchetti et al., it was shown that the electron devices experience a multi regime type of emission. In low voltages, they emit like Schottky. In mid-range voltages, they emit like Fowler-Nordheim, and eventually they saturate at Child-Langmuir for high voltages. We could assume an empirical cutoff and model Schottky emission for $v < v_{\text{cutoff}}$ and Fowler-Nordheim for $v > v_{\text{cutoff}}$. However, the cutoff might be different for each type of device, so we instead combine the emissions by addition. We assume that both mechanisms are active simultaneously, but in the different regimes a different one of the emission mechanisms dominates.

Schottky emission is described in [39] as a thermal emission from a device with a work function ϕ. The application of a voltage across it acts as a reduction of the work function and allows more of the thermally spread electrons to jump over the barrier.

$$J_S(E) = \alpha T^2 \exp\left(\min\left(\frac{-\phi + \frac{e}{2}\sqrt{\frac{eE}{\pi\epsilon}}}{k_B T}, 0\right)\right) \tag{3.8}$$

Where T is the temperature in degrees Kelvins, ϕ is the work function of the metal in eV, ϵ is the permittivity of the material in $e^2 eV^{-1} nm^{-1}$. E as before, is the electric field magnitude in $V\text{nm}^{-1}$. The square root dependence on the electric field can be viewed as reducing the work function barrier. We select the minimum of $\frac{-\phi+\frac{e}{2}\sqrt{\frac{eE}{\pi\epsilon}}}{k_B T}$ and 0 because we assume that once the work function barrier has been reduced below zero, the electron emission remains the same. This also allows us to make the transition regimes cleaner.

We let the emission current density be

$$J(E)_{FN+S} = J_{FN}(E) + J_S(E) \tag{3.9}$$

with $J_{FN}(E)$ from eq. (3.6) to match the modeling of the experimental results attained by Turchetti et al.

Enhanced Fowler-Nordheim + Schottky

We also considered a combined Schottky and Fowler-Nordheim emission where the Fowler-Nordheim is also enhanced. This form came about after realizing that the disparity between the 2D and 3D electrostatic models is around the order of 7, which we consider for the gamma. The hope is for this model to match experiment closer than previous models, due to the compensation. The emission current density considered is

$$J_{EFN+S}(E) = J_{EFN}(E) + J_S(E) \tag{3.10}$$

with $J_{EFN}(E)$ from eq. (3.7) and $\gamma = 7$.

3.1.2 Other Variations in Cost Function

In addition to a large emission current, we care about fabricability of the device and where the current in it is going. Those factors led us to consider adding two terms to the cost function: the total surface area and the component of the emission current directed in the x direction. Penalizing the total surface area might help reduce spikiness, which the optimizer might favor because it is an easy way to increase the surface area and the field enhancement. Yet, we wish to avoid spikiness in the surface because it is difficult to fabricate and unreliable to operate. Adding a term that encourages the initial direction of the emitted currents to be biased towards the x direction could influence more current to arrive a the collector. However, the electric field sweeps the electrons and changes their direction of travel as soon as they are emitted. Perhaps the initial current direction does not affect the final result much.

We considered two ways to transform the current maximization problem into a cost minimization. The first obvious choice was to take the negative multiple of the current, which ended up being our method of choice here. We also considered taking the inverse of the total current. That method, however, is not a linear transformation. It stretches the change in current at small currents and shrinks the change in current at large currents. In general, we did not see major differences between the resulting geometries under the different cost functions. That can probably be attributed to the fact that $-i$ is related to $1/i$ with a

logarithm, which is a monotonically increasing function.

3.2 Constraints

As mentioned before, our optimization has a mix of explicitly stated and implicit constraints. The explicit constraints restrict each of the parameter points $p_i = (x_i, y_i)\ \forall i = 1, \ldots, n$ to be inside a box as follows

$$\text{xmin} \leq x_i \leq \text{xmax} \quad \text{and} \quad \text{ymin} \leq y_i \leq \text{ymax} \quad \forall i = 1, \ldots, n,$$

where xmin, ymin, xmax, and ymax are at a distance of 5 nm from the edge of the simulation. This constraint ensures that the emitter outline does not contain any points that are outside of the simulation domain.

The implicit constraints in our optimization are the requirements for the simulation to run without crashing. Those constraints consist of a requirement for the emitter to be a valid polygon with no edges that cross over each other, and a requirement for the surface of the emitter to have a well defined normal at every point.

When the explicit boxing constraint is violated, the optimizer takes the rebellious point that has crossed the boundary, and puts it in its place; at the closest point in the allowed region. When either of the unstated constraints is violated, the FEniCS solver crashes.

We implemented a try-except block around the simulator to allow the optimization loop to continue running even after the solver crashes (appendix A.2). However, as life teaches us, not all is well and good when problems go ignored and undealt with. It appears that when the solver crashes and the program does not terminate, the garbage collection of the FEniCS simulation fails to clear allocated memory. Eventually, with enough occurrences of solver crashes, the whole program fails with a memory error. We saw that the reduced parameterizations has a better robustness against these violations due to it's less flexible nature.

We implemented a program that runs through geometries that have crashed the simulator and makes use of try-except blocks to determine which constraint was violated. An invalid

polygon crashes the simulation in the domain creation or in the meshing step. A polygon with poorly defined normal lines crashes the solver when calculating the electric field. From an analysis of the simulator crashes, we can see that approximately 10% of the simulation crashes are caused by the optimizer attempting to evaluate a self-intersecting emitter.

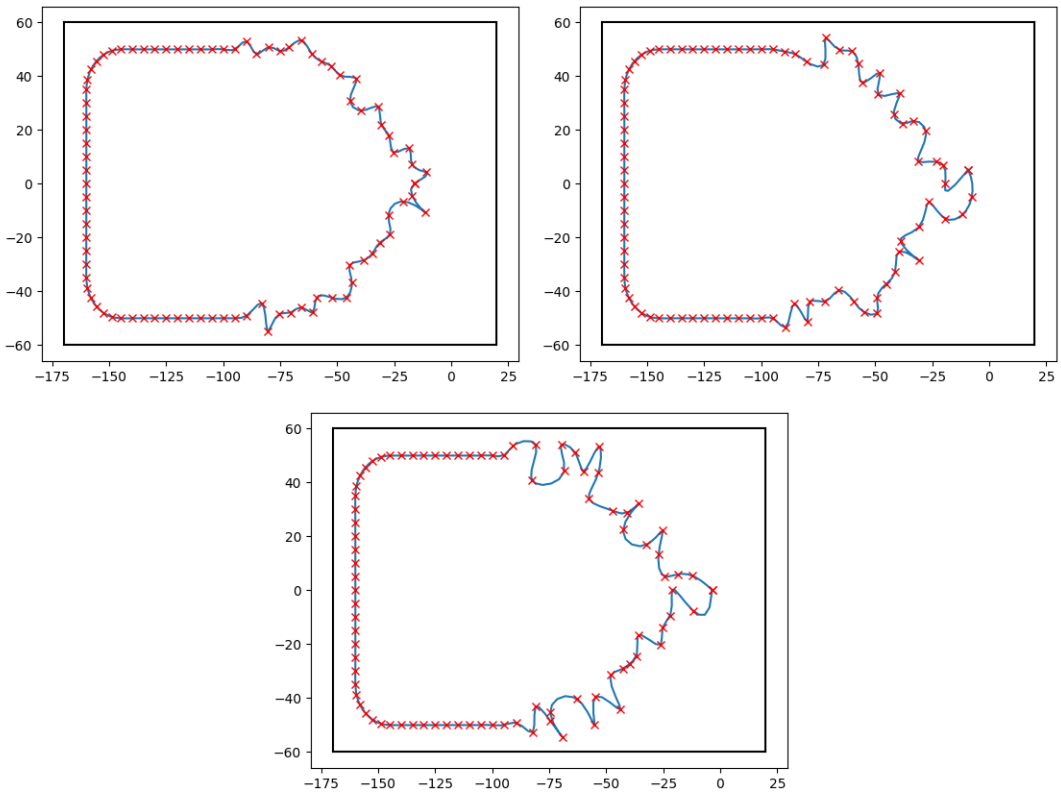

Figure 3-1: Geometries responsible for crashing the solver in the domain formation and meshing stage. Parameter points are shown as red exes and the spline interpolation is shown in the blue line. When enlarging the figures, it can be seen that some of the sharp tips contain narrow loops, making the polygons invalid.

90% of solver crashes are accounted for by gradient calculation errors. When the surface of the device becomes rough, it becomes difficult to calculate the cross product and find the normal to the surface. As mentioned before, the gradient is not defined at the boundary of the simulation, therefore we need to take a step into the simulation along the direction perpendicular to the boundary. When the boundary is rough and the normal line is ill defined, the algorithm might try to evaluate the gradient at a point inside the emitter, which is not part of the simulation domain. The attempt to evaluate the gradient at a point outside the domain results in a simulator crash.

3.3 Initial design

The initial design is a naïve rectangle with a triangular tip that is rounded at the end and has a radius of curvature of about 10 nm. This geometry corresponds to a realistic initial design. The current emitted from it depends on the emission model used and is indicated in fig. 3-2.

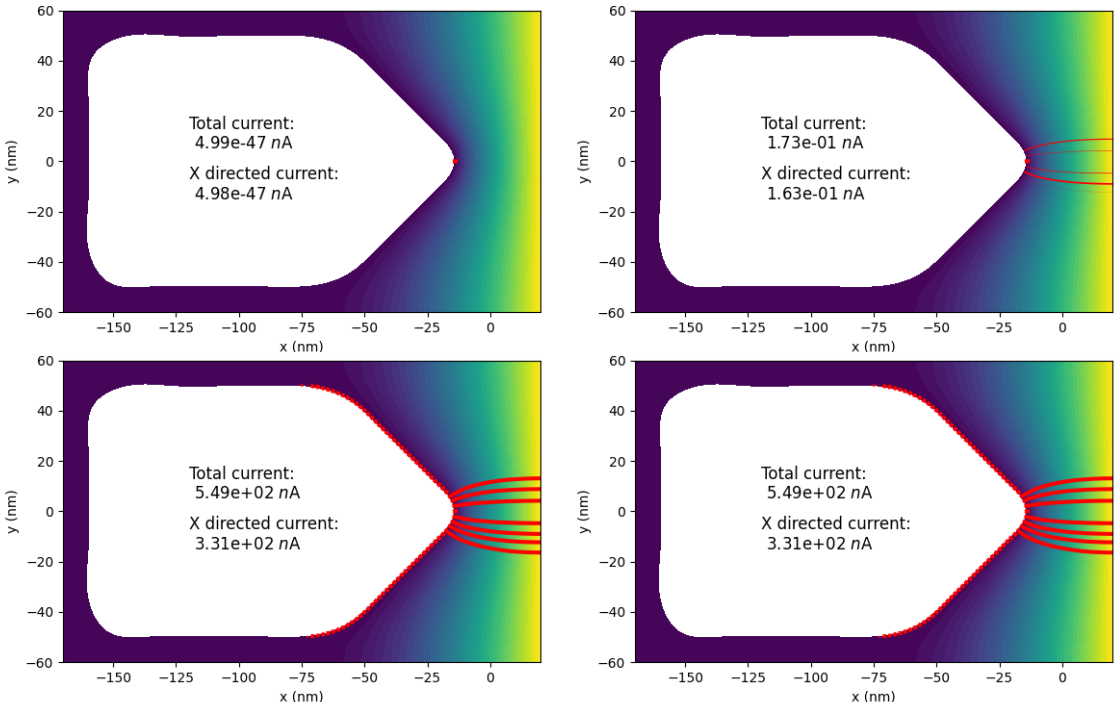

Figure 3-2: The initial guess for geometry yields wildly different current predictions depending on the emission mechanism assumed when 15 V are put across it across it. (Top left) Simulation predicts 5×10^{-47} nA, basically zero for pure Fowler-Nordheim, (top right) 0.173 nA of current for enhanced Fowler-Nordheim. (bottom left) Simulation predicts 550 nA of current for with Schottky + Fowler-Nordheim.(bottom right) Simulation assumes predicts basically the same amount of current for the Schottky + enhanced Fowler-Nordheim emission mechanism

For the normal Fowler Nordheim, we get an initial current of 4.99 $\times 10^{-47}$ nA, which means that an electron is emitted every 10^{29} years, essentially a zero current. For the enhanced Fowler-Nordheim, we get an initial current of 0.173 nA, a respectable current. In the Schottky un-enhanced case, we see 549 nA of current and the same for the enhanced Fowler-Nordheim + Shcottky.

3.4 Runtimes

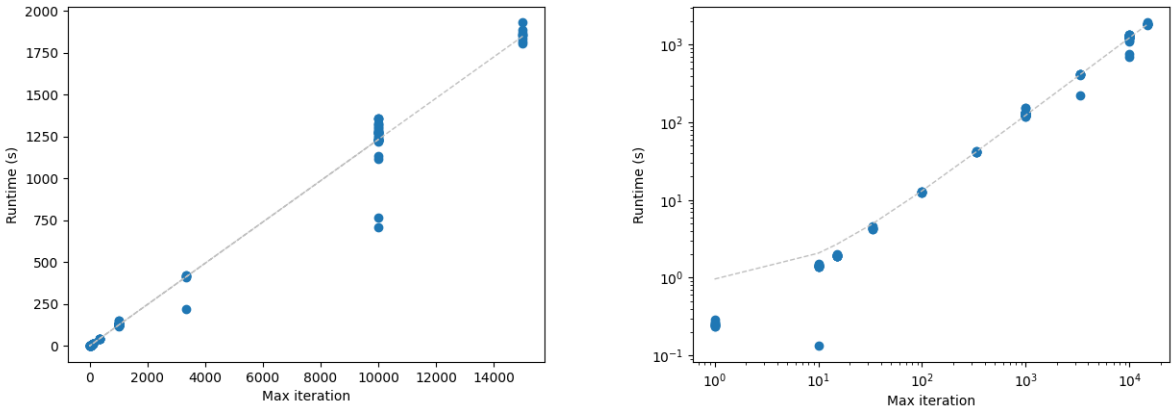

Figure 3-3: Runtimes for a simulated annealing optimization in python plotted against the number of iterations. (left) A linear plot is shown with the line of best fit. (right) A log-log plot is shown with the line of best fit to resolve the lower iteration numbers better.

When using the framework written in MATLAB, the runtime of an optimization with 1,000 iterations was approximately 28 - 30 minutes. When using the python home-brewed simulated annealing framework, an optimization of 1,000 iterations took approximately a minute to run, a 30 times speed up. The lion's share of the speedup is accounted for by the removal of the reading from and writing to external files which was used in the communication between the layers of code in the MATLAB implementation. Additional speed ups were provided by the usage of Numpy array operations instead of loops in some sections of the simulation code. The switch to array operations instead of loops was concurrent with the move of the optimizer code to python. So, we cannot distinguish between the speedups provided by the array operation and by the removal of communication with external files. Figure 3-3 shows the runtime of the optimization in python as a function of the maximum iteration. It should be noted that the runtime doesn't quite scale linearly as a function of maximum iterations. One reason for the nonlinear trend is that the more iterations pass, the weirder the emitter shape will be and the more likely it will be to evolve into a shape that pushes against the unstated constraints and crashes the FEniCS solver. Section 3.5.5 investigates the tendency of the rate of constraint violation to grow with response to an increase in the number of optimization iterations.

3.5 Results

We optimized the initial geometry to maximize emission current under the assumption of four different emission mechanisms. The cost function under each emission assumption is different, giving rise to different optimal results. Therefore, we present the results of the optimization and evaluate its performance under the different emission assumptions in separate sections.

3.5.1 Fowler-Nordheim cost

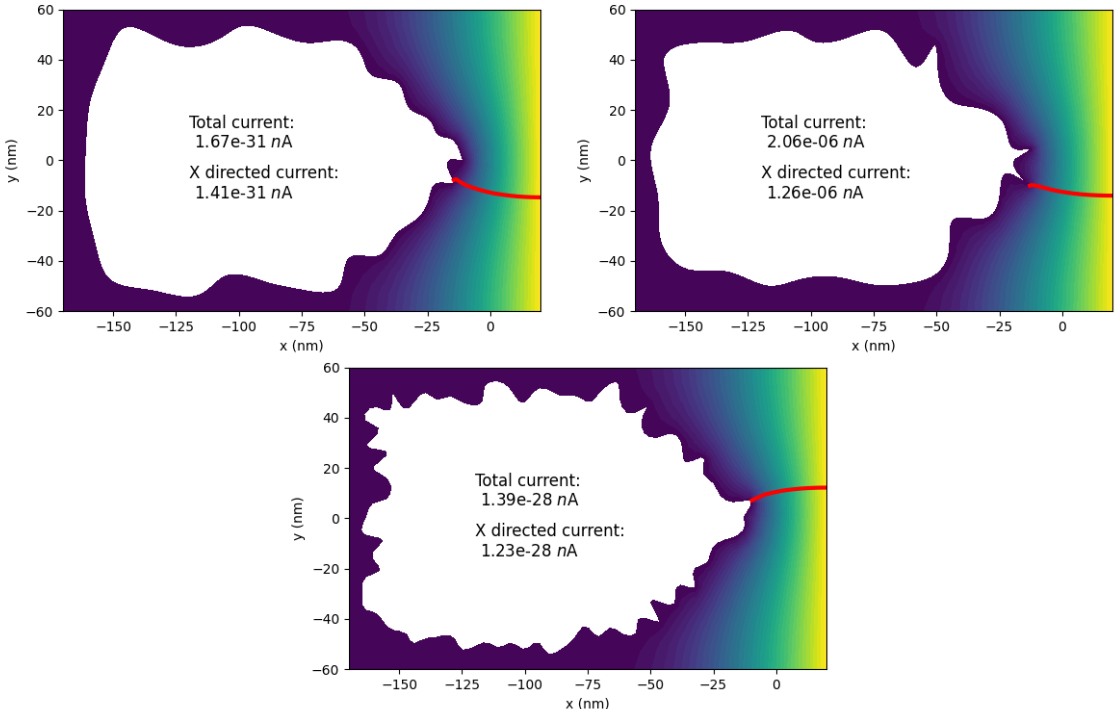

Figure 3-4: Results of optimization with a cost function depending on un-enhanced Fowler-Nordheim emission. (top left) The intermediate results of the optimization with 1000 iterations on a reduced parameterization give 1.67×10^{-31} nA current, a 3.35×10^{16} improvement compared to the initial current. (top right) the intermediate result after 10,000 iterations for the reduced geometry gets a current 2.06×10^{-6} nA, a 4.13×10^{40} times improvement. (bottom) The intermediate result for optimization with 1,000 iterations on the original parameterization gives rise to a current of 1.39×10^{-28} nA, an improvement of a factor of 2.8×10^{18}

We ran several optimizations for the initial geometry diode with a cost function that depends on a pure Fowler-Nordheim emission. In some runs of the optimization, we parame-

terized the geometry with a dense point distribution (termed "original" parameterization in section 2.1.2). Other runs were parameterized by a reduced density of points, which we refer to as the "reduced" parameterization. The optimization for the original parameterization with maximum iteration of 10,000 crashed. Figure 3-4 shows a selection of the results that survived which highlight the formation of sharp tips.

The top left sub-figure in fig. 3-4 shows the results of optimizing the diode with a reduced parameterization assuming Fowler-Nordheim emission with a maximum iteration of 1,000. The resulting current is 3.35×10^{16} the initial current. However, this figure is misleading. The initial current was 4.99×10^{-47} which means that one electron is emitted every 10^{29} years. The optimized current corresponds to an electron every 10^{12} years, which is still essentially a zero current. The same applies to the results represented in the bottom of fig. 3-4. There, the optimization also has a maximum iteration of 1,000, but it was run with the dense original parameterization. The resulting current is 2.8×10^{18} times the initial current, corresponding to an electron emitted once every 10^{10} years, an essentially a zero current.

We got a substantial current from the optimization that was run for 10,000 iterations (top right of fig. 3-4). It looks like the pair of very sharp tips are responsible for the non-zero current emission.

3.5.2 Enhanced Fowler-Nordheim cost

We ran the optimization for a diode with a cost function that depends on an enhanced Fowler-Nordheim emission with an enhancement factor of $\gamma = 7$, as is done many times in literature to match experimental results. However, in our simulations, the electric field is smaller than expected for the structures due to the 2D nature of the simulations. The enhancement factor could be correcting to the 3D electric field value. The optimizations were run on the original parameterization on the reduced one with different maximum iteration conditions. Like had happened for the pure Fowler-Nordheim cost, the optimization on the original parameterization with maximum iteration of 10,000 crashed. The intermediate results seem to imply that some combination of a sharpen tip and a bulb optimize the current well. The top left sub-figure of fig. 3-5 converged to a geometry with a sharp tip and a bulb and it emits a current of 101 μA, 5.8×10^5 times the initial current. It seems like most of the

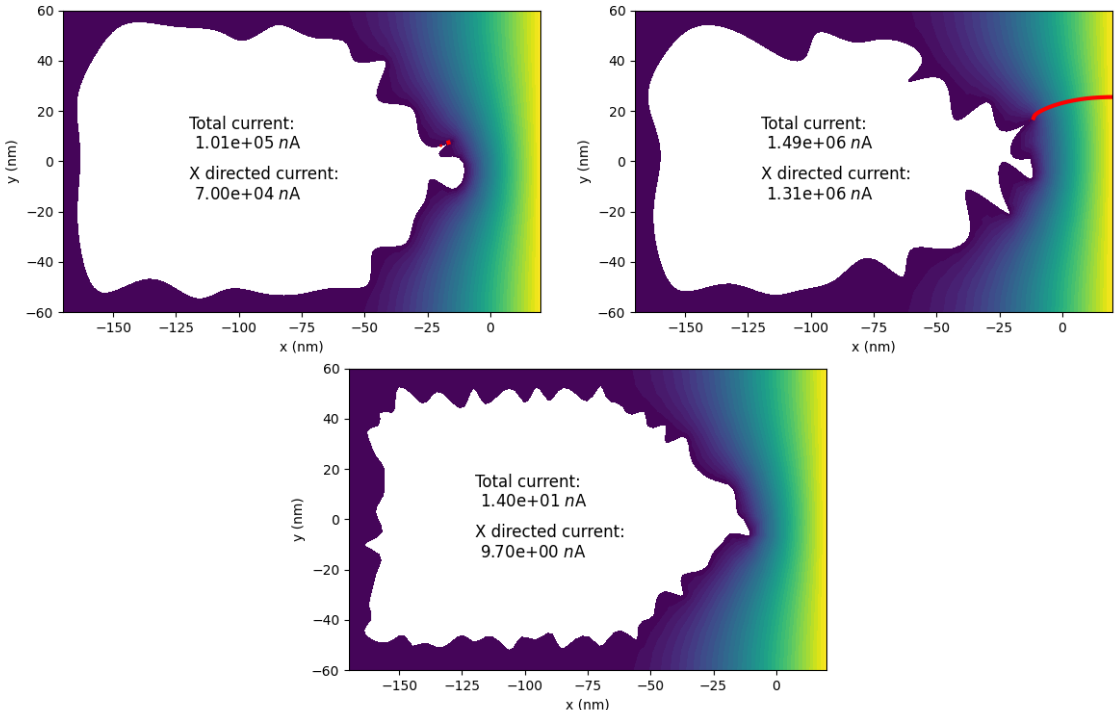

Figure 3-5: Results of optimization with a cost function that depends on an enhanced Fowler-Nordheim emission with $\gamma = 7$. (top left) The optimization with 1000 iterations on the reduced parameterization gives a current of 101 μA, 5.8×10^5 times the initial current. (top right) The optimization with 10,000 iterations on the reduced parameterization gives a current of 1.5 mA, a 8.6×10^6 times improvement. (bottom) The optimization with 1,000 iterations on the original parameterization, gives a current of 14 nA, an 81 times improvement compared to the initial current.

current is coming out of the sharp tip near the bulb. The structure in the sub-figure on the top right gives a current of $1.5mA$, a 8.6×10^6 times improvement compared to the initial current. It has a sharp tip which seems to supply the lion's share of the current in addition to a flat edge/bulb part. The geometry on the top right was obtained after 10,000 iterations and produces an order of magnitude more current than the geometry on the top left that was obtained after 1,000 iterations. The optimization shown in the bottom of fig. 3-5 resulted in a structure with a not-particularly-sharp tip that gives a current of 14 nA, an 81 times improvement compared to the initial current.

3.5.3 Fowler-Nordheim + Schottky

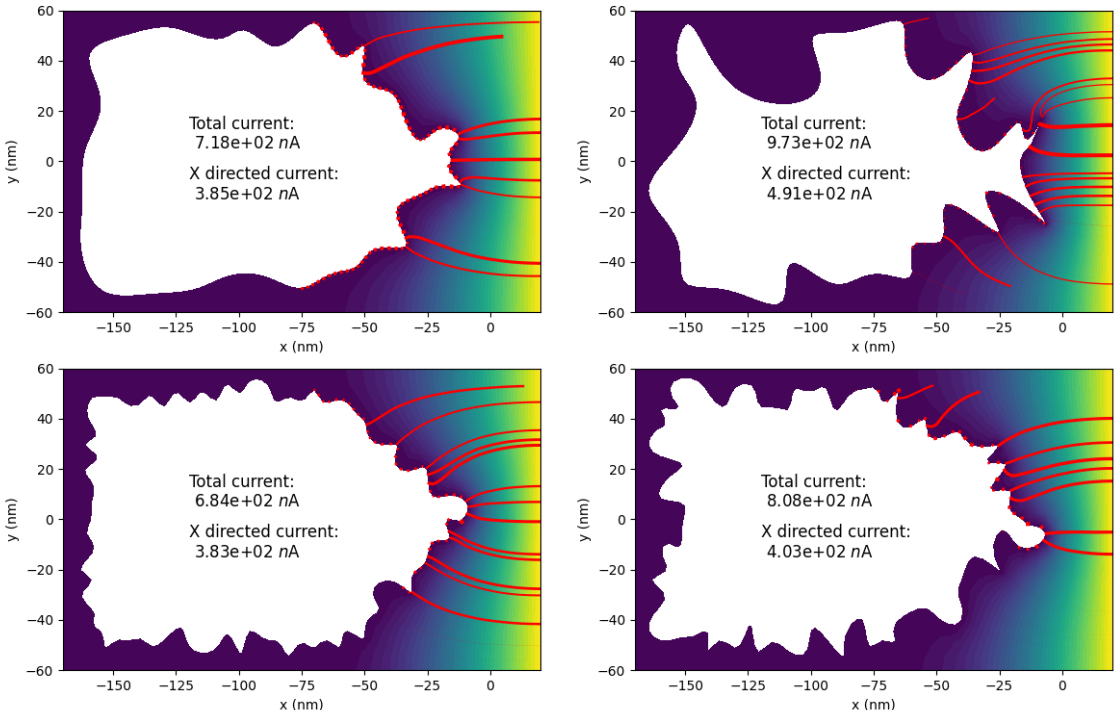

Figure 3-6: Results of optimization with a cost function that depends on an emission model of un-enhanced Fowler-Nordheim + Schottky. (top left) The optimization on the reduced parameterization with 1,000 iterations gave 718 nA, 1.31× the initial current. (top right) The optimization on the reduced parameterization with 10,000 iterations gave 973 nA, 1.77× the initial current. (bottom left) The optimization on the original parameterization with 1,000 iterations resulted in a current of 684 nA, 1.25× better compared to the initial design. (bottom right) The optimization on the original parameterization with 10,000 iterations gave a current of 808 nA, 1.47× better than the initial current.

We ran several runs of optimizations with a cost function that assumes a combined

emission of un-enhanced Fowler-Nordheim and Schottky. A sample of the results obtained are presented in fig. 3-6. Overall, we see a tendency of the optimizer to favor large surface area. Some sharp tips arise, but it seems like the dominant features are flat edges like in the top right sub-figure of fig. 3-6 and bulbs like in the bottom 2 geometries in the same figure. Favoring large surface areas seems to be a property of the Schottky term in the cost function. The best improvement we see in the in cost function is an improvement of 1.77×.

3.5.4 Enhanced Fowler-Nordheim + Schottky

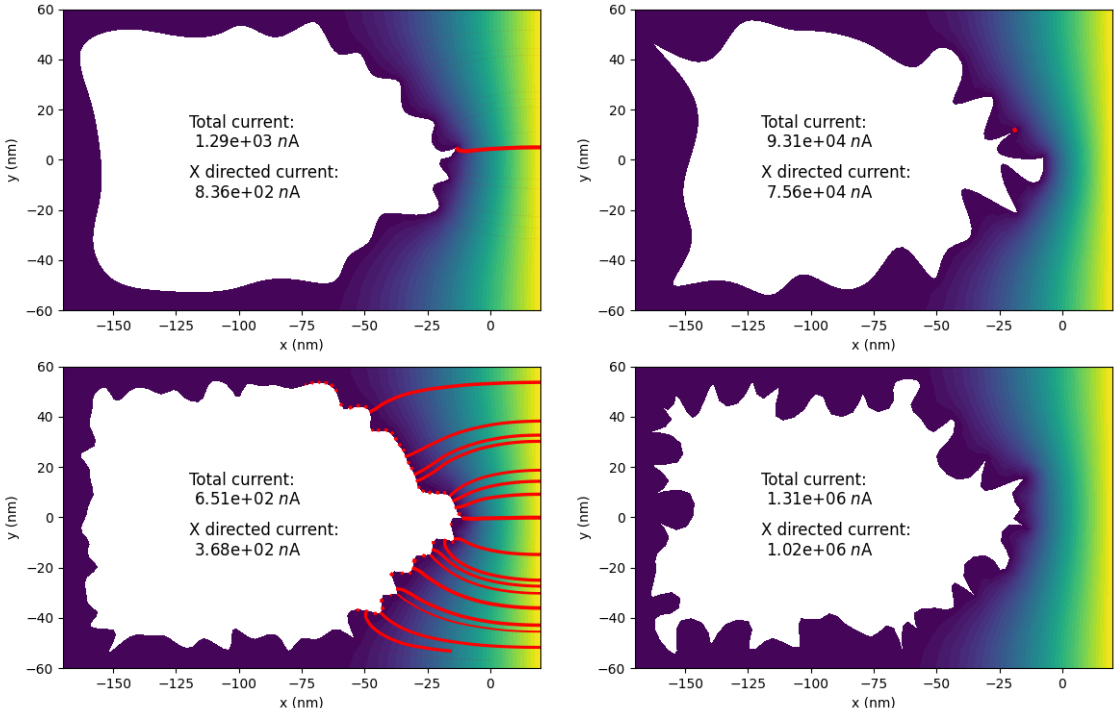

Figure 3-7: Results of optimization assuming an emission model of enhanced Fowler-Nordheim + Schottky. (top left) The optimization on the reduced geometry with 1,000 iterations gives a current of 1.29 μA, 2.35× the initial current. (top right) The optimization on the reduced parameterization with 10,000 iterations resulted in a current of 93.1 μA, 169.6× better compared to the initial current. (bottom left) The optimization on the original parameterization with 1,000 iterations gave 651 nA of current, 1.19× the initial current. (bottom right) The optimization on the original parameterization with 10,000 iterations gave 1.31 mA of current, $2,386\times$ better than the initial current.

We ran optimization with a cost function that depends on an emission model that combines enhanced Fowler-Nordheim with Schottky. The optimizer favors sharp tips, though some of the geometries develop bulb like features as well, for example the structure in the

top right sub-figure of fig. 3-7. The best improvement in current is seen for the optimization on the original parameterization with 10,000 iteration (bottom right of fig. 3-7). The final current of 1.21 mA, is 2,386× the initial current. The high current is achieved by the multiple sharp ridges at the tip.

3.5.5 Constraint violation

As mentioned in section 3.2, there are unstated constraints in the optimization, the violation of which, causes the FEniCS simulation to crash. We collected data about the rate of constraint violations for each independent optimization. We see from fig. 3-8, that for both parameterizations, the rate of constraint violation grows as the maximum number of iterations of the optimization goes up. As the number of maximum iterations increases, the optimizer finds its way to more obscure and more random geometries. Therefore, each individual geometry in the later optimizations is more likely to be a polygon that self-intersects or have poorly defined normal lines. Self intersecting polygons crash the simulator in the domain definition and meshing stage, and poorly defined normals crash the simulator in the electric field calculating stage.

We can see that generally the rate of constraint violation is lower for the reduced parameterization than for the original parameterization. More interestingly, the rate of growth of the rate of constraint violation is slower for the reduced parameterization than for the original one. This is indicated by the less steep slope of the orange dotted fitting line in comparison to the blue dotted fitting line in fig. 3-8. The significance of this finding lies in the fact that the garbage collection of FEniCS fails when the simulator crashes without terminating the program. With enough constraint violations, a significant memory leak occurs and the optimizer crashes with a segmentation fault. With the reduced parameterization, we can run the optimizer for many more iterations before it encounters enough errors to reach a segmentation fault. That's why sections 3.5.1 and 3.5.2 include results for the reduced parameterization with 10,000 iterations, but not for the original parameterization.

If we wish to run the optimizer without fear of crashing, we can probably code a garbage disposal process to be triggered after a pre-determined number of simulation crashes. We can, additionally, use the Shapely python library [56] to check whether the polygon is valid before

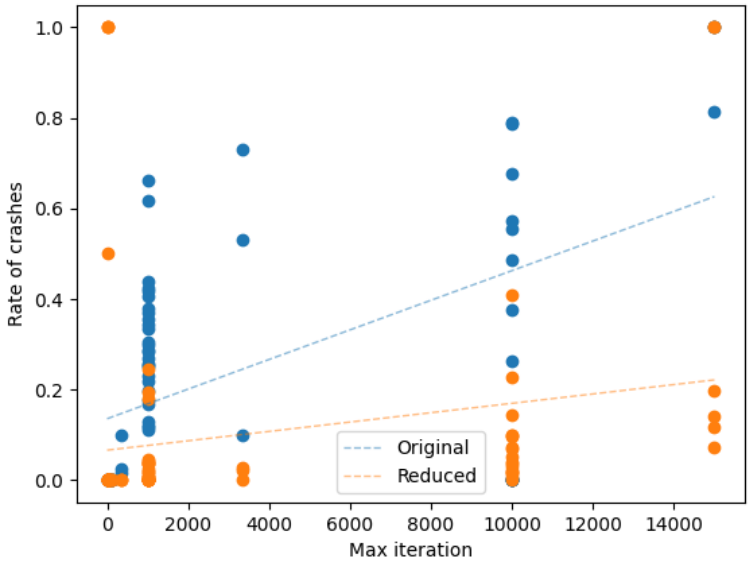

Figure 3-8: Constraint violation rate vs maximum iteration of the optimizer for original and reduced parameterization. The spread is due to the usage of different emission-mechanism cost functions.

attempting to simulate it. However, it is more ideal to minimize the error numbers to begin with. We can consider forgoing the spline interpolation and accept a linear interpolation with smoothing on the harsh corners. Alternatively, we can look at quadratic splines instead of cubic ones. The lower curvature will prevent the curves from having the flexibility to loop over themselves and form self intersecting polygons. Additionally, we propose new methods of annealing and new parameterization which might be less prone to constraint violation in section 3.7.

3.6 Model limitations

"All models are wrong, but some are useful" is quote often attributed to George E. P. Box. Our simulations do not perfectly agree with experimental works, or even with more advanced simulations in the field. However, their usefulness is found in their inaccuracy. The very same features that make the model agree less with reality, makes it faster, allowing us to implement optimizations with 10,000 iterations that run in less than 25 minutes. This section

addresses the sources of inaccuracy in the modeling which enable the speedups.

3.6.1 Emission regimes

Recent work in the field has established that metallic and ceramic electron emitters display a multi-regime emission behavior. Starting with Schottky emission at low voltages, with Fowler-Nordheim emission in mid range voltages, and saturation with Child-Langmuir in high voltages [9, 50]. There is no theory to dictate or explain the relative weighting of the different emission mechanisms or the cutoff between them. This work considered a Fowler-Nordheim and a combined Schottky and Fowler-Nordheim emission. Since the theory is not well understood, the cutoff in the combined emission model was fitted to one experimental data set. We did not consider Child-Langmuir emission in this work.

As we showed in section 3.5, the assumption of emission mechanism fundamentally changes the total current measured and the geometrical features that are selected by the optimizer. The emission model needs to be better understood to better represent reality and produce truely optimal devices.

3.6.2 Mesh resolution

The effect that the mesh resolution has on the solution of numerical PDEs is often overlooked, but it is incredibly important. As a general rule, the higher the mesh resolution, the higher the accuracy of the PDE solution. In finite element methods, the mesh can be thought of as a basis that the solution gets projected onto. When the basis is larger, there are more degrees of freedom to express the solution, therefore it is more flexible and accurate. The major downside of a dense mesh, is the computation time required to solve the PDE on that mesh [49]. The underlying matrices are $n \times n$ where n is the number of mesh nodes. Matrix solutions are typically $\mathcal{O}(n^3)$, so the solution time grown non linearly with the mesh resolution.

To get a hold of the magnitude of the error from 20 mesh partitions, we simulated a tip structure with different mesh resolutions and and plotted the field enhancement measured at the tip and the solution time vs the mesh resolution. The left sub-figure in fig. 3-9 shows

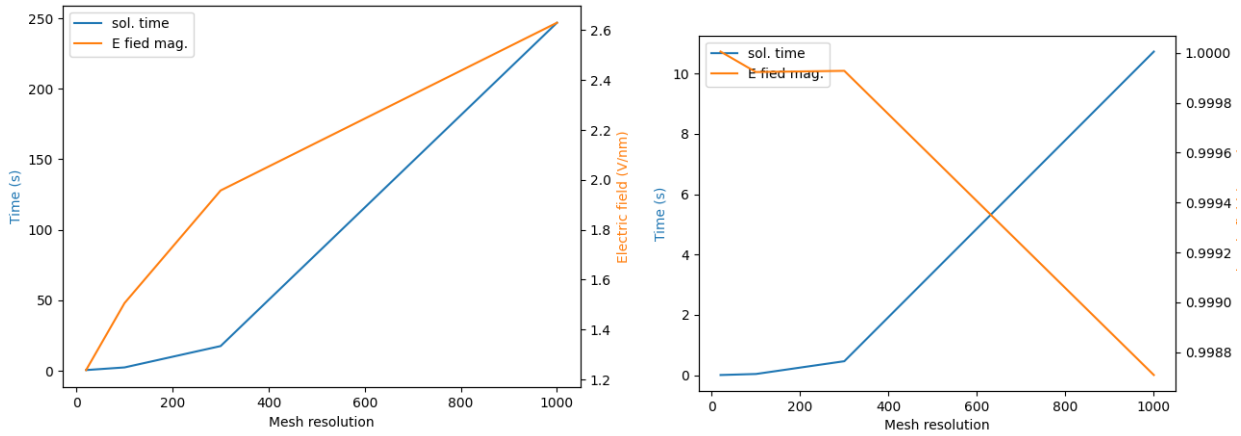

Figure 3-9: The effects of the mesh resolution has on the solution can be huge. (left) For a structure with a tip, the field enhancement at the tip is double when going from 20 to 1000 mesh partitions. (right) The mesh resolution has no noticeable effect on the solution of a parallel plate capacitor

a more than 2× change in the field enhancement calculated for the same structure with different mesh resolution. The better accuracy comes at a computational cost, going from 0.2 seconds for a 20 partition mesh, to over 4 minutes for 1000 partitions. For a simulation of a parallel plate capacitor, on the other hand, we see no major change in either the field enhancement as the mesh resolution increases. The computation time increases, but it not as much as we see for the tip, as seen in the right sub-figure of fig. 3-9.

3.6.3 2D electrostatic models

An additional source of inaccuracy comes from our usage of a 2D electrostatic simulation instead of a 3D one [51]. 2D simulations map well to 3D when the objects simulated are uniform and extend to approximately infinity in the missing dimension (the dimension excluded from the simulation). To quantify the quality of the infinite approximation in our modeling, we recreated 2D versions of 3D simulations seen in literature. We simulate a 2D cross section of the 3D device simulated by Turchetti et al., in which the authors see a field enhancement factor of $\gamma = 7$ where $E = \gamma\frac{v}{d}$ where d is the gap distance [50]. The field enhancement factor that we see in our 2D simulations is off by a factor of 3-4, depending on the mesh resolution. To verify that the deviation is, in fact, due to the 2D simulation

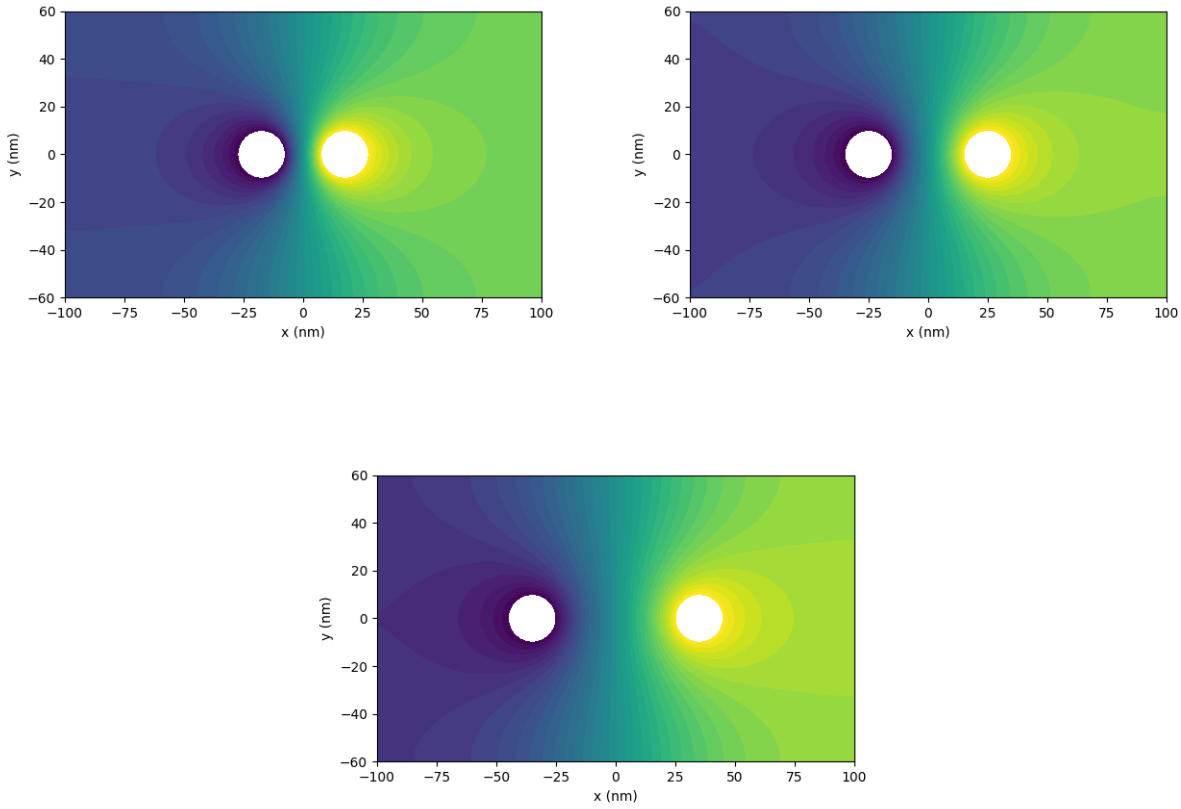

Figure 3-10: Simulations of circles of

and not other factors, we simulated structures that have a well known analytical solution for the field enhancement. We simulated two circles of radius 10 nm with a fixed voltage across them. We then compared the simulated electric field at the surface of the circles to the theoretical value of the electric field for spheres and infinite cylinders with the same radius embedded in 3D space. The formulas we used are

$$E_{max} = 0.9 \frac{V}{d} \frac{r + \frac{d}{2}}{r} \tag{3.11}$$

for spheres, and

$$E_{max} = 0.9 \frac{V}{2r \left(\ln \left(1 + \frac{d}{2a} \right) \right)} \tag{3.12}$$

for cylinders [52].

We see a field enhancement that matches the theoretical value for cylinders with more

than 90% accuracy for gap distances of 10 mn - 50 nm between the circles. In comparison, the calculated electric field matches the theoretical value for the sphere model with close to 90% accuracy for distance of 10 nm between the circles. However, the accuracy drops as the distance increases. For a distance of 50 nm, there's a 60% agreement between the measured electric field and the the theoretical value.

It seems that a 2D simulation is a very good approximation when the simulated structure is uniform and very tall in the dimension that's omitted. The required length-scale is probably an order of magnitude or more than the features along the other dimensions. The devices under study are about 20 - 40 nm tall, around the same order of the gap distance.

When simulating two sheets with a voltage difference, the 2D simulation agrees perfectly (within numerical noise) to the theoretical structure. The distance between the sheets doesn't seem to change the field seen in simulation.

3.6.4 Boundary condition

There seems to be a little bit of an issue with the application of Dirichlet boundary conditions in certain locations. When zooming in on fig. 3-11, we can see that there are a few points along the geometry for which the Dirichlet condition is not applied. That probably occurs when the condition applying the boundary conditions assumes a smooth or rounded surface, but the mesh discretization causes a deviation from that. Therefore, the edges are in a location different than the boundary condition expects. To remedy this, have a larger tolerance on the boundary condition location.

3.7 Conclusion and outlook

As we've seen from the results in this chapter, there are two features which are selected for by the optimization, a sharp tip with a very small radius of curvature, and a bulb. The sharp tip generates a large field enhancement which drives a high current density. The bulb provides a large surface area to emit current from. The features selected by the optimizer depend on the assumed emission mechanism. When Schottky dominates the emission, the algorithm will prefer a bulb while, and when Fowler-Nordheim dominates, a sharp tip will be

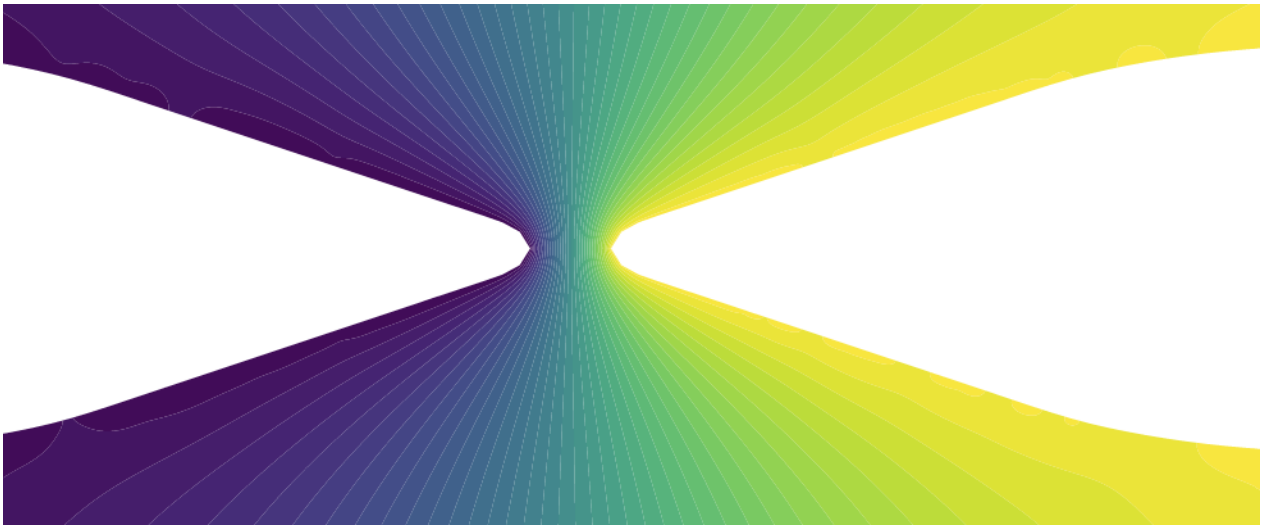

Figure 3-11: On non straight edges, the Dirichlet boundary conditions sometimes fail to get applies due to insufficient tolerance of the boundary condition function

selected- within a range. Fowler-Nordheim starts to saturate at higher field enhancements [9, 48], so when we assume an artificially large γ, or have a high voltage difference, the algorithm will start tending towards increasing surface area rather than forming sharp tips for Fowler-Nordheim too. At the limit where the number of iterations approach infinity, we expect the emitter shape to evolve into either a flat edge infinitely close to the collector or into a comb with many infinitely sharp tips. Both or which are problematic for different reasons. The flat edge emitter is a poor design because the emitted current will be uniformly spread across it and we will have poor control over the current trajectories. Having a beam with a high current density is much more convenient because it can be bent and re-routed if needed, while a wide beam with a uniform spread is much more difficult to re-route. A comb of infinitely sharp tips is problematic because those are essentially impossible to fabricate. The resolution of modern fabrication is limited by the technology available, but it is fundamentally limited by the atomic size of the material used.

In our runs of the optimizer, we haven't seen the algorithm settle to either the flat edge or comb, presumably because we didn't let the optimization run for for long enough. The memory leakage due to repeated constraint violation crash the optimizer before those structures have a chance to emerge. Below are suggestions for implementations that can help reduce the rate of constraint violation and speed up convergence.

3.7.1 Dynamic step

In the original simulated annealing formulation, the random perturbation that generates the new test geometry, depends on the temperature at that iteration. The temperature dependence is usually encoded as the variance of the distribution from which the perturbation size is drawn. Towards the beginning of the simulated annealing, T is high, meaning that the variance of the distribution is large, allowing a wide spread of perturbation sizes with a high upper limit. As the annealing proceeds and the system cools down, T gets smaller, and the perturbation sizes approach the mean. Our formulation implemented a temperature dependant perturbation and a constant variance one. The temperature dependant step is usually associated with faster convergence to an optimal result. However, we were unable to test the convergence rate of the temperature dependant version of the algorithm because the number of constraint violation encountered by it had crashed the optimizer every time. With a different parameterization or better checks before simulating, we can get the temperature dependant step to work.

3.7.2 Symmetry

By imposing symmetry constraints across the x axis, the dimensionality of the search space of the optimal geometry gets significantly reduced. The reductions of the search space is generally associated with convergence in fewer iterations. We can implement this feature pretty easily by removing half of the parameter points, constraining the remaining points to the top half plane, and reflecting the interpolation result onto the bottom half plane.

3.7.3 Smoothing

The results presented in this chapter were generated by a series of random perturbations and a stochastic selection of beneficial perturbations. The geometrical perturbations consisted of an independent random perturbation of all parameter point along directions perpendicular to the surface. Perturbing all the points independently of each other contributes to the surface roughness of the overall geometry, thus leading to a higher probability of constraint violation which crashes the simulator. We implemented an averaging window smoothing to

the perturbations after every step in an attempt to make the perturbation smoother. That had the effect of decreasing the expected step size and lowering the variance. After 1,000 iterations with the smoothed perturbations, the resulting shape has not changed by much. That is because the mean for the perturbation size is zero due to the consideration of inwards and outwards perturbation, so the expected perturbation when averaging is zero. Another smoothing method should be considered.

3.7.4 Principle Component Analysis

We notice that the parameterization considered in this work is overly flexible and expressive, and creates a huge search space of solutions. Meanwhile, the space of valid devices is a small subspace of the full search space, and the space of optimal devices is even smaller. We can attempt to find the manifold of valid devices and of optimal devices by doing a principle component analysis (PCA) which takes the components corresponding to the largest singular values from a singular value decomposition (SVD). That will cut the optimization into tow stages. Stage one: run the optimization as before and save all the intermediate geometries. Then, perform a PCA and transform the parameters. Stage two: run the remainder of the optimization on the reduced model. Uncovering the manifold of valid devices and running the optimization on that manifold is guaranteed to stop simulator crashes due to constraint violation and will likely help us reach the optimal device quicker.

As we discussed earlier, the truly optimal diode under the total emission current maximization objective, is somewhat ill defined. Which is why we moved to dealing with transistors which are discussed in next chapter.

Chapter 4

Global optimization for gated devices

In the previous chapter, we focused on optimizing the current emitted from diodes. We considered different emission mechanisms and incorporated them into the cost functions that were optimized. While diodes are incredibly useful in many applications and can be used to form complicated logic circuits [1], they do not form a complete set of logic elements. Typical diodes require the use of a transistor to form a NOT gate. Though, tunneling transistors which have a negative resistance region can form a NOT gate with a transformer [2]. Transistors are the building block of modern electronic computation. In this chapter, we optimize a planar nano vacuum channel transistor (PNVCT) with respect to two figures of merit: the off/on current ratio and gate leakage. The full device depicted in right sub-figure of fig. 2-4 has an emitter, two gates, and a collector. To control the scope, we optimize only the emitter geometry, though next steps should consider optimizing the geometry of the other components in the device as well. The behavior of transistors is complicated to express mathematically as an objective function, so we focus on simpler individual attributes of desirable transistors behaviors such as the on/off current ratio and the gate leakage current. Optimizing the on/off current ratio is synonymous to minimizing the source-drain leakage in MOSFETs and is a common figure of merit [31]. Gate leakage, however, is not a concern in MOSFETs because of the insulating oxide, but in PNVCTs, there is no strong potential barrier that prevents leakage to the gate, so we need to mitigate that too. We implement heuristics to speed up the figure of merit calculations and explore regularized optimization.

4.1 Cost functions

There are many figures of merit that can be optimized in a transistor such as the cut off voltage, saturation, gate modulation, power consumption, frequency response, etc [46, 7]. An ambitious cost function would be to optimize the the full I-V response to fit to a target desirable response. That would require simulating each structure along a two dimensional sweep as the collector-emitter voltage was changes and as the gate-emitter voltage was changed. To make things worse, for each simulation in the sweep, we would have to solve the electrostatics, simulate particles emitted, and track them as they made their way from the emitter to the gate or the collector. That would be incredibly computationally expensive and impractical for any application. Therefore, we focus on figures of merit that are computationally less demanding and therefore better suited for an optimization with thousands of iterations.

4.1.1 Switching

In a transistor, the gate controls the conductivity of the channel. When fixing the voltage between the emitter and the collector, the gate voltage dictates the current emitted from the emitter and what portion of it makes it to the collector. An ideal transistor has infinite gate control; when the gate is at high voltage, it turns the device on and causes a high current to flow between the collector and emitter. When the gate is at a low voltage, it ensures that no current flows between the emitter and collector. We aim to maximize the on-off current ratio, hoping to push it as close to infinity as possible. The gate voltage acting as a switch is the reason we refer to this figure of merit as the switching behavior in addition to calling it the on/off current ratio or gate control merit. We can mathematically represent this cost function as

$$\min_{p_1,\ldots,p_n} \frac{\int_{\partial\Omega(p_1,\ldots,p_n)} J_{\text{collector}(E_{\text{gate off}})} \cdot dS}{\int_{\partial\Omega(p_1,\ldots,p_n)} J_{\text{collector}(E_{\text{gate on}})} \cdot dS} \tag{4.1}$$

Where $J_{\text{collector}}$ is the current density of a beam that leaves from the emitter and ends up on the collector and $p_i = (x_i, y_i)$ like defined in chapter 3. Integrating the beam over the surface of the emitter, $\int_{\partial\Omega} J_{\text{collector}} \cdot dS$, gives us the total collector current. We cannot directly measure the current in the collector. So instead, we are tasked with determining which

trajectories leaving the emitter arrive at the collector, and integrating those. Section 4.2 deals with how we calculate which beams arrive at the collector and which at the gate efficiently. The term $E_{\text{gate on}}$ refers to the electric field extracted from the electrostatic simulation with the gate at the on voltage. Similarly, $E_{\text{gate off}}$ refers to the electric field extracted from the electrostatic simulation with the gate at the off voltage. The electric field, E, is calculated as described in eqs. (3.2) and (3.3) with boundary conditions as described in eqs. (3.4) and (3.5). However, the set of Dirichlet boundary conditions now includes the gates which can have a high or low voltage across them. For each geometry, we run two simulations; one with the gate at a high voltage and one with the gate at low voltage. The current density $J(E)$ depends on the emission mechanism assumed. We considered three of the four mechanisms described in section 3.1.1: Fowler-Nordheim, Enhanced Fowler-Nordheim, and Fowler-Nordheim with Schottky.

4.1.2 Gate leakage

In gated devices that do not have a gate separated by a layer of oxide, there is a concern that a significant fraction of the emission current will escape to the gate, leading to low signal at the collector and poor power efficiency. We introduced a cost function that encourages a strong and robust collector signal and suppresses the gate signal to be weak in comparison. The cost can be written as

$$\min_{p_1,\ldots,p_n} \frac{\int_{\partial\Omega(p_1,\ldots,p_n)} J_{\text{gate}}(E_{\text{gate on}}) \cdot dS}{\int_{\partial\Omega(p_1,\ldots,p_n)} J_{\text{collector}}(E_{\text{gate on}}) \cdot dS} \tag{4.2}$$

where $J_{\text{gate}}(E)$ refers to the current density of a beam that generates from a spot on the emitter with electric field E and reaches the gate. Similarly, $J_{\text{collector}}(E)$ is the current density of a beam that generates on the emitter at a spot with electric field E and reaches the collector. For this cost function, the electrostatic simulation needs to be run only once, with the gate at a high voltage. The low gate voltage case needs not to be considered, because no current reaches the gate when it is set to low.

4.1.3 Regularized

When optimizing an objective, the algorithm could inadvertently make the performance on another objective worse even if there is no strong inherent trade-off between the objectives. There are multiple ways to implement multi-objective optimization, where the performance on one objective is not sacrificed for the benefit of the other. One popular option is to optimize one objective while forcing the performance on another objective to remain above a baseline using a constraint. Another approach is to attempt to optimize multiple objectives simultaneously with a regularizer.

In our devices of interest, a trade-off has been shown between the gate control and the collector signal strength. When the gate voltage has a lot of influence over the emitter-collector current, the collector current tends to be low. In the case of a strong trade-off between objectives, a satisfactory compromise can be achieved by the Pareto front, the collection of points with a tight trade-off obtained by regularized optimization.

We implemented a multi-objective regularized optimization to asses the trade-off between the switching cost and the gate leakage cost. A regularized optimization can be written mathematically as

$$\min \lambda \text{cost}_1 + (1-\lambda)\text{cost}_2 \qquad \text{s.t.} \quad \lambda \in [0,1], \tag{4.3}$$

or more explicitly

$$\min_{p_1,\ldots,p_n} \frac{\int_{\partial\Omega(p_1,\ldots,p_n)} \lambda J_{\text{gate}}(E_{\text{gate on}}) + (1-\lambda) J_{\text{collector}(E_{\text{gate off}})} \cdot dS}{\int_{\partial\Omega(p_1,\ldots,p_n)} J_{\text{collector}}(E_{\text{gate on}}) \cdot dS}. \tag{4.4}$$

A regularized cost function requires two cost electrostatic simulations to be run, one with the gate at high voltage and another with the gate at low voltage.

4.2 Heuristic for particle tracking

With gated devices, current emitted can end up in the collector, gate, or even return back to the emitter. For that reason, we can no longer use the framework we used for diodes, which

assumes that all the current emitted makes its way to where it is needed. Here, particle tracking is necessary. Typically, particle-in-cell simulations would be used for that purpose. The different varieties of particle-in-cell simulations include a self consistent simulation where in each iteration the position and momentum of emitted particles are advanced in an electric field and the electric field is updated to include the effects of the charged particles. A lighter-weight version updates the position and momentum of the simulated particles but not the electric fields. There is a particle-in-cell simulation package called LEopPart that can be added to FEniCS. However, full particle tracking is a computationally expensive process, especially when done self consistently to account for space-charge effects. Instead, we create a rough heuristic for particle tracking as a zeroth order estimate for the purposes of optimization. We assume that emitted particles have no mass and therefore no momentum, and that they follow the electric field lines until they reach the collector, the gate, or the edge of the simulation. While the absolute accuracy of this heuristic might not be high, all that is needed for optimization is a good relative accuracy that maps to some improvement in the absolute cost. So long as the heuristic guides the optimizer to a geometry that is close to the optimal geometry, further local optimization can be done with more accurate models to converge to the true local optimum. The heuristic for particle tracking was initially developed as an iterative algorithm for each point on the emitter's surface as described in chapter 2. That implementation was sub-optimal, and was sped up by doing an implementation of binary search on the trajectories.

4.2.1 Bisection search

The current trajectories are indexed by their origin point on the emitter, with the 0 index having an origin at the tip. Trajectory indices increase in the counter clockwise direction of their origin point going around the emitter. We realized that the current trajectories are sorted to some extent; the trajectories near the tip generally end up at the collector, and the trajectories further from the tip generally end up at the gate. This means that in order to calculate the current on the collector, we need to find the first and last trajectories to end up on the collector. All the trajectories indexed by a number smaller than the first gate trajectory encountered are collector trajectories. All the trajectories indexed by

a number larger than the last collector trajectory, will end up on the gate. The bisection search in practice is slightly more complicated than a normal bisection search because some trajectories end up neither on the the gate nor the collector; they either fly off to the edge of the simulation or they loop back to the emitter. This means that some indices of trajectories need to be skipped before a bisection could be applied to narrow down the search space. Algorithm 4 in chapter 2 describes the bisection search process.

4.3 Initial performance

To establish a baseline for benchmarking the optimization, we simulated the initial geometry device under two gate conditions: the gate at low voltage (0 V) and gate at high voltage (15 V). In both simulations, the emitter is at 0V and the collector is at 15V. The performance of the initial device is strongly variable depending on the assumed emission mechanism. Therefore, we applied the performance analysis separately to each assumed emission type and ran separate optimizations for each emission mechanisms.

	Gate on	Gate off
$i_{\text{collector}}$ (nA)	$2.63\times^{-36}$	3.66×10^{-60}
i_{gate} (nA)	1.96^{-48}	0.00

Cost	Value
Switch	1.39×10^{-34}
Leakage	7.45×10^{-13}.

Table 4.1: Simulated currents (left) and the corresponding cost functions (right) for transistor with the initial design assuming Fowler-Nordheim emission. Switch cost refers to the off/on current ration defined in section 4.1.1. Leakage cost refers refers to the gate/collector current ratio defined in section 4.1.2.

Table 4.1 shows the gate and collector currents and associated cost function values for an initial transistor with Fowler-Nordheim emission. The figures of merit seem satisfactory on first glance, but are essentially meaningless because all currents being compared are essentially zero, making the figure of merit undefined. The largest-magnitude current, 2.63×10^{-36} nA, translates to an electron once every 2×10^{18} years.

The currents for a simulation that assumes Fowler-Nordheim + Schottky emission are listed in table 4.3. These currents are orders-of-magnitude larger than those calculated under the assumption of un-enhanced Fowler-Nordheim emission and about an order-of-magnitude larger than those calculated assuming enhanced Fowler-Nordheim, implying that in this

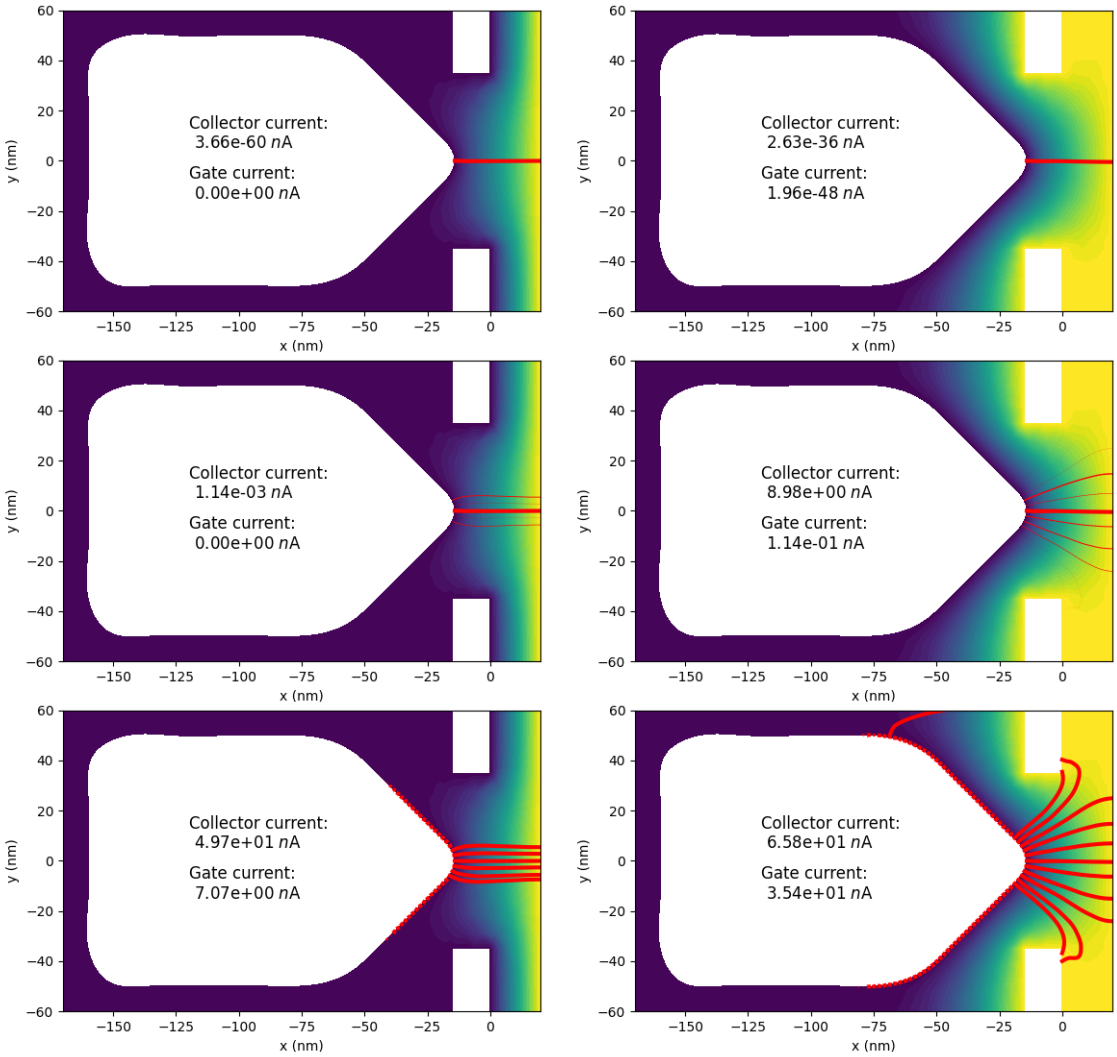

Figure 4-1: Results of initial device design simulated in FEniCS with the emitter at 0 V, collector at 15 V, and (left column) gate off (v_{gate} =0 V) and (right column) gate on (v_{gate} = 15 V)and emission model (top) Fowler-Nordheim, (middle) enhanced Fowler-Nordheim, (bottom) Fowler-Nordheim + Schottky.

	Gate on	Gate off
$i_{\text{collector}}$ (nA)	8.98	1.14×10^{-3}
i_{gate} (nA)	0.114	0.00

Cost	
Switch	1.27×10^{-5}
Leakage	0.013

Table 4.2: Simulated currents (left) and the corresponding cost functions (right) for transistor with the initial design assuming enhanced Fowler-Nordheim emission with an artificial enhancement factor of $\gamma = 7$.

	Gate on	Gate off		Cost	
$i_{\text{collector}}$ (nA)	65.8	49.7		Switch	0.76
i_{gate} (nA)	35.4	7.07		Leakage	0.54.

Table 4.3: Simulated currents (left) and the corresponding cost functions (right) for transistor with the initial design assuming enhanced Fowler-Nordheim + Schottky emission.

geometry, the Schottky component of the emission dominates.

4.4 Runtime

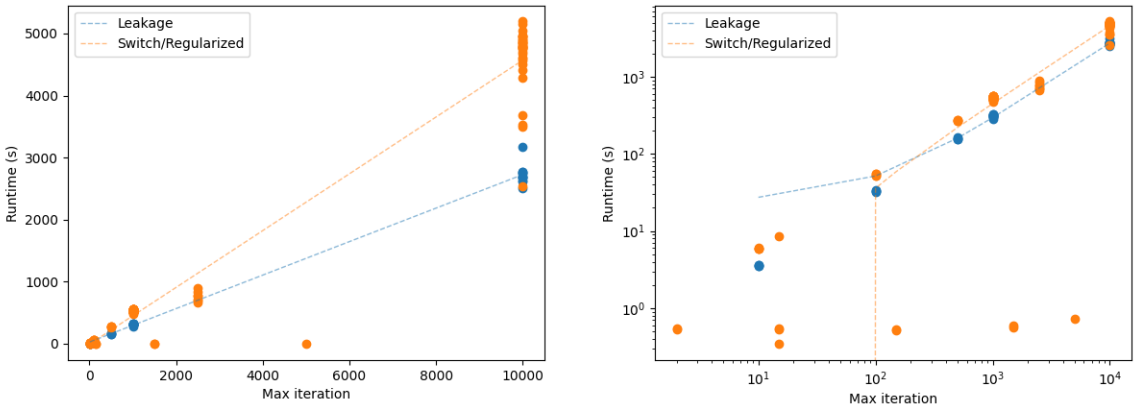

Figure 4-2: Runtime for optimization of a three-terminal device plotted against the number of iterations in the optimization. The leakage cost function requires only one electrostatic simulation, while the switch cost and the regularized cost functions each require two simulations. Therefore, we analyze the runtime of the optimizer under the switch cost and the regularized cost separately from the runtime of an optimization under the leakage cost.

When analyzing the runtime of the optimization, it is important to differentiate between cost functions that require only one simulation and cost functions that require two simulation runs. The optimization is much slower per iteration compared to the optimizations discussed in the previous chapter because of the trajectory tracking and the usage of two simulations in some cost functions. The runtime is roughly linear in the number of iterations. Intuitively, the slope of the linear fit to the runtime of the regularized cost and switch cost is about twice as steep as the slope of the fit for the leakage cost.

4.5 Switch Results

For each one of the emission mechanisms considered, we ran 6-10 rounds of optimization to minimize the ratio of off-current to on-current in the collector. We used the geometry displaced in fig. 4-1 as the initial geometry and used the reduced parameterization for the decision variables.

4.5.1 Fowler-Nordheim emission

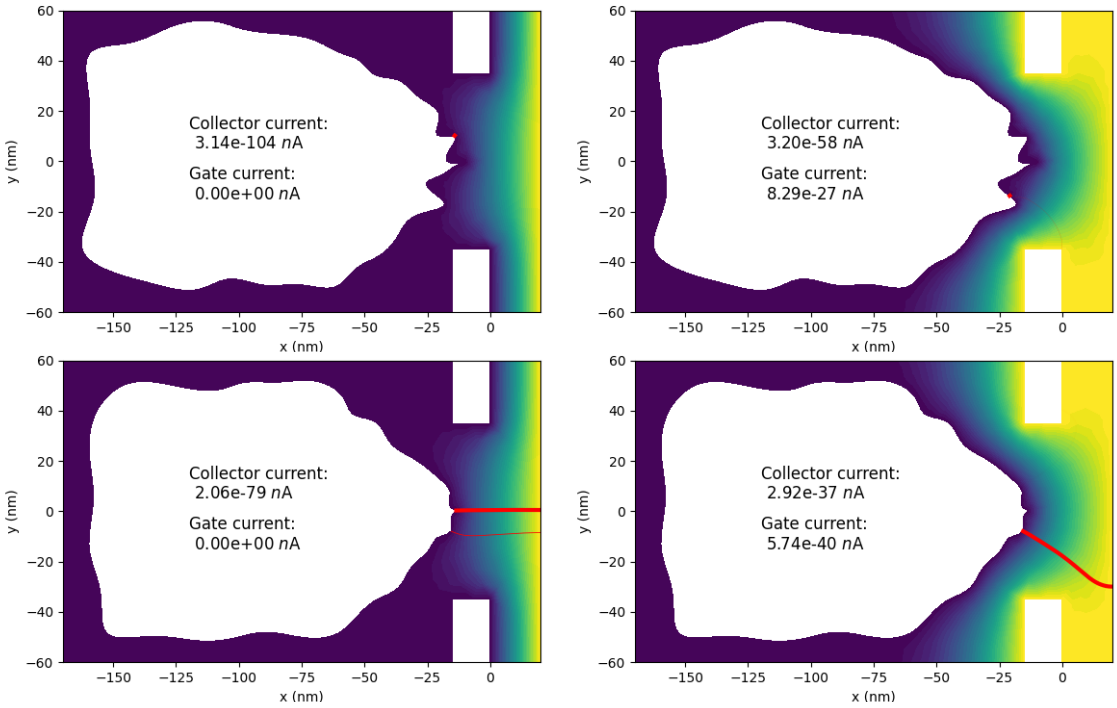

Figure 4-3: Results of optimizing the NVCT with switch cost function assuming Fowler-Nordheim emission and maximum iteration of 10000. In both cases the cost function is improved by reducing the off current.

The results of the optimization with the switch cost function assuming a pure Fowler-Nordheim emission look promising at first. For the two cases displayed in fig. 4-3, we see cost function values of 9.8×10^{-47} and 7×10^{-43}. That is 13 and 9 orders of magnitude better than the initial geometry's performance respectively. The figure stated here is misleading however, because the advantage is achieved by pushing down the off current rather than increasing the on current. The effective currents are still zero, making the figures of merit are undefined.

Putting absolute accuracy aside, we can appreciate the geometrical features that the optimizer selected for in the relative improvement. In the top sub-figure of fig. 4-3, the side sharp tips exactly in line with the left edge of the gate suppress the electric field seen by the tips when the gate is off.

4.5.2 Enhanced Fowler-Nordheim emission

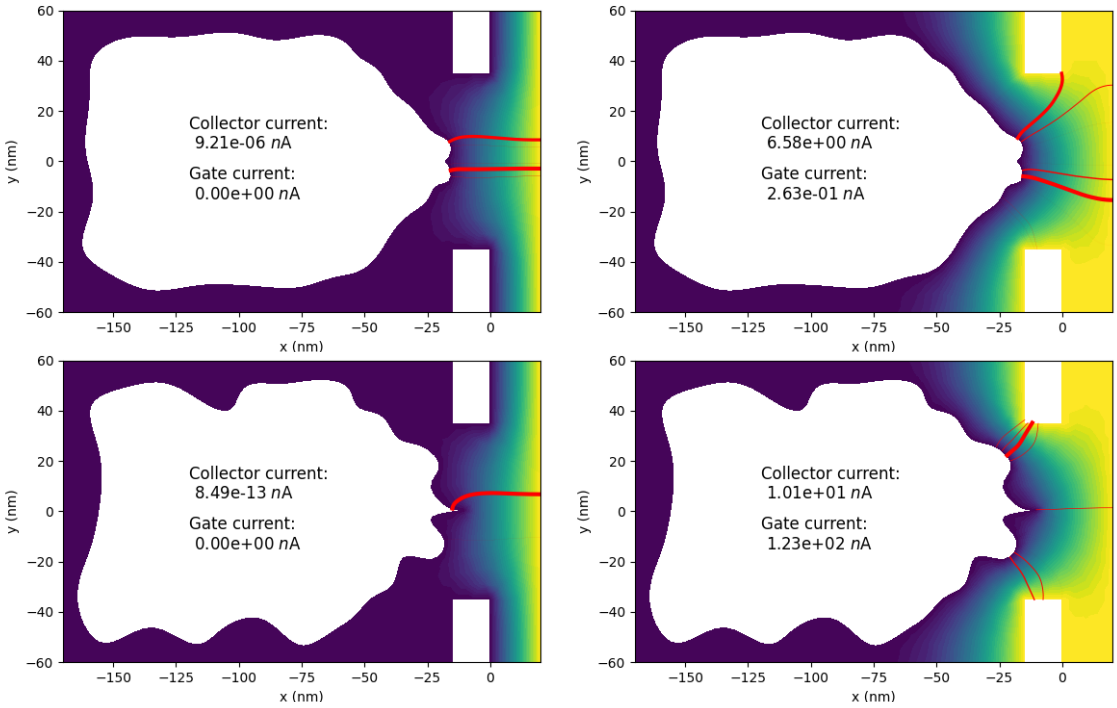

Figure 4-4: Results of optimizing the NVCT with switch cost function assuming enhanced Fowler-Nordheim emission and maximum iteration of (top row) 1000 and (bottom row) 10,000. (left column) gate off and (right column) gate on.

The results from optimizing the device for the switch cost function with enhanced Fowler-Nordheim emission looks promising. For the examples shown in fig. 4-4, we see off-current to on-current ratios of 1.4×10^{-6} and 8.3×10^{-14} for the 1000 iteration and the 10,000 iteration examples respectively. Here too, the algorithm achieves a better objective value by reducing the off current, but without significantly eliminating the on current too.

Interestingly, the optimizer favors a feature of two symmetric bumps instead of a tip, a feature that can be seen in the top row of fig. 4-4 among other simulations that are not shown here. When the optimizer does find a tip, it keeps the bumps symmetrically above

and below it.

4.5.3 Fowler-Nordheim + Schottky emission

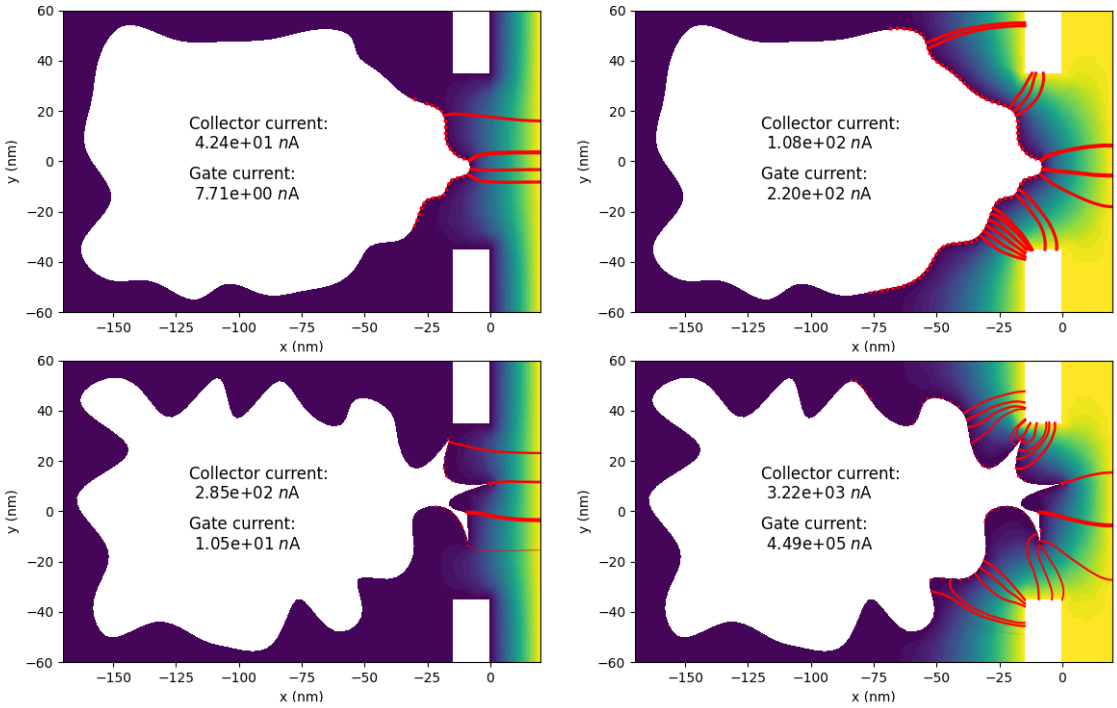

Figure 4-5: Results of optimization of switch cost function assuming Fowler-Nordheim + Schottky emission for (top) 1000 iterations and (bottom) 10,000 iterations. (left column) gate off and (right column) gate on

For the example of 1,000, iterations we see an off/on current ratio of 0.39, a 48% decrease from the initial value of 0.76. For the example of 10,000 iterations we see an off/on current ratio of 0.088, an 88% decrease from the initial ratio. It seems like the optimizer is insisting on increasing the on current, at the expense of making the off current also somewhat higher. The way the optimizer increases the collector current when the gate is on is by creating features that lie exactly in the space between the top and bottom gate. When the gates are on, the features see the potential gradient and emission is encouraged, but because the features are at the same x coordinate as the edge of the gate, the electrons will be swept to the collector instead of being swept into the gate. Those features also increase the emission when the gate is off, but at a rate lower than the rate that still improves the objective value.

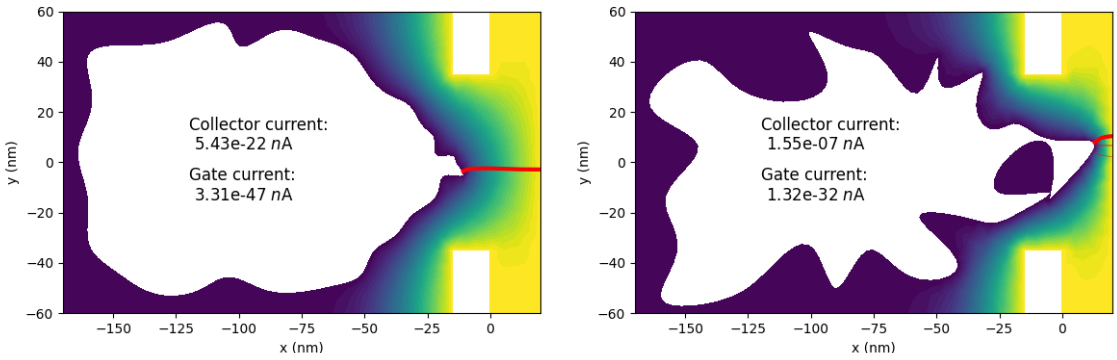

Figure 4-6: Results of minimizing the gate leakage cost assuming Fowler-Nordheim emission for (left) 1000 iterations and (right) 10,000 iterations. Both structures achieve cost function values that are 13 orders of magnitude smaller than the initial cost.

4.6 Leakage Results

For each one of the emission mechanisms considered, we ran a few rounds of optimization to minimize the leakage current stolen by the gate. We used the geometry displaced in fig. 4-1 as the initial geometry and used the reduced parameterization for the decision variables.

4.6.1 Fowler-Nordheim

In the example that was run for 10,000 iterations (right sub-figure of fig. 4-6), the optimizer manages to reduce the fraction of current proportionally consumed by the gate by pushing the device edge to the right, past the gate. This design allows for gate current of 8×10^{-26} times the collector current without decreasing the collector current to a point where it is no longer detectable. That is a 13-order-of-magnitude reduction in the cost function with respect to the initial value. However, from inspecting the figure, we can assume that the on/off current ratio is close to one. In the example that converged in 1000 iterations (left of fig. 4-6), the device became sharp only in the center space between the two gates causing emission to be present at a location where the electrons will get swept to the collector. The gate to collector current ratio is 6.2×10^{-26}, a similar cost function value as the result in the right sub-figure, yet here the collector current is essentially zero (an electron emitted once every 9,000 years).

4.6.2 Enhanced Fowler-Nordheim emission

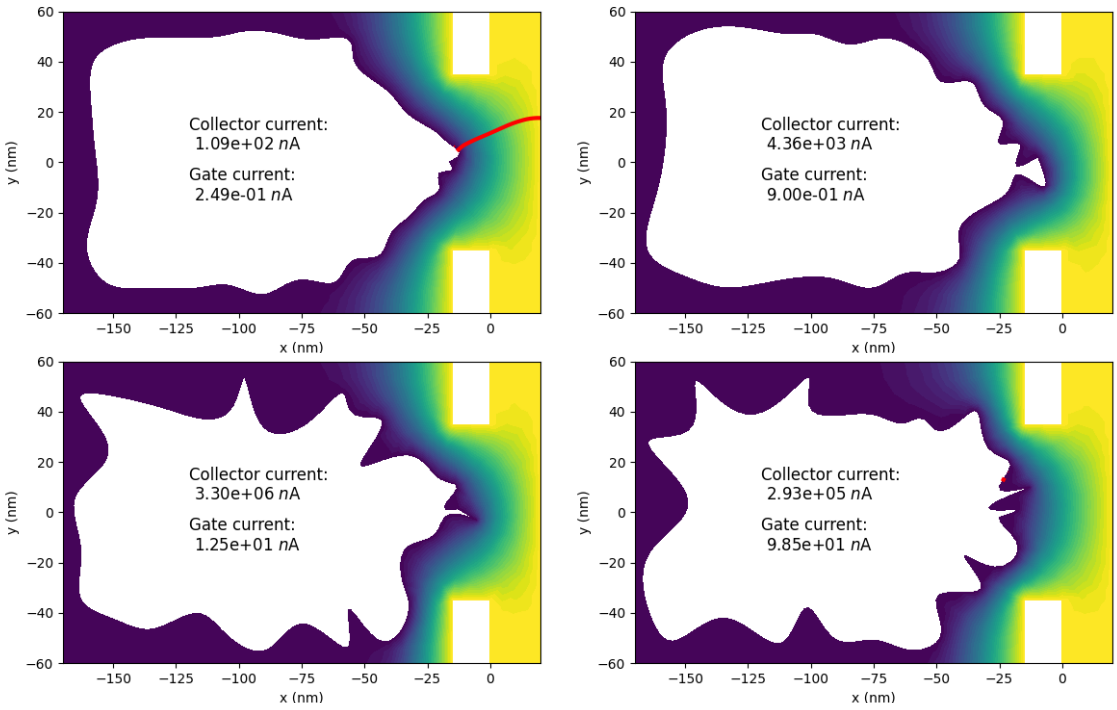

Figure 4-7: Gate on simulations of results of an optimization of gate leakage with enhanced Fowler-Nordheim emission assuming an artificial enhancement factor of $\gamma = 7$. (top row) Results of optimizations with maximum iteration = 1,000. (bottom row) Maximum iteration = 10,000.

In fig. 4-7, we show results of an optimization assuming enhanced Fowler-Nordheim emission. We see gate/collector ratio cost values of (left to right, top to bottom) of 2.3×10^{-3}, 2.06×10^{-4}, 3.8×10^{-6}, and 3.4×10^{-4}. With an initial cost of 0.013, we see improvements of 5.65×, 63×, 3.421×10^{4}×, and 38× respectively.

An interesting feature that emerges here is the series of emitter dips in the emitter sides near the gate corners. A large field enhancement is seen near the gate corners because they are simulated as perfectly sharp, and the points on the emitter closest to the gate corner are most likely to emit current that will be absorbed by the gate. To decrease the emission that is absorbed by the gates, the optimizer pushes away the side of the emitter closest to the gate corner, causing those segments of the emitter to be flat or cave in. Concavity shields the points from the large electric field and flatness reduces the field enhancement, resulting in less emission at the proximity of the gate. The collector current is made higher

by introducing sharp features in the space between the gates, at a spot where the emitted beam will be swept directly to the collector.

4.6.3 Fowler-Nordheim + Schottky

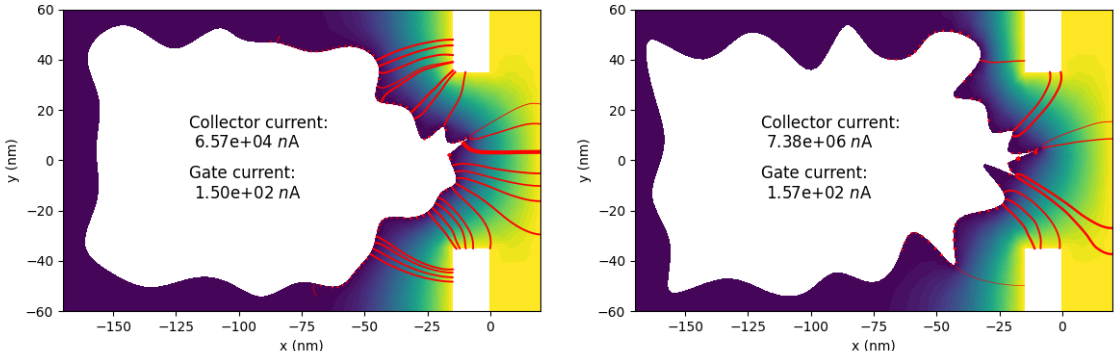

Figure 4-8: Gate on simulations of results of an optimization of gate leakage with Fowler-Nordheim + Schottky emission. (left) optimizations stopped after 1,000 iterations. (right) 10,000 iterations.

In fig. 4-6 we show the results of an optimization assuming Fowler-Nordheim + Schottky emission. Despite the difficulties, the optimizer manages to push down the cost function by creating sharp features that generate a large current that then gets swept to the collector. The gate current also gets increased as a result of secondary features, but the overall cost decreases.

Schottky emission has an exponential of a square root dependence on field enhancement, a weaker dependence than Fowler-Nordheim exhibits. Additionally, Schottky assumes an amount of thermal emission even for zero electric field. For those reasons devices with Schottky emission are generally difficult for the optimizer when it comes to gate-to-collector current ratio reduction. The introduction of sharp features to help raise the collector current has a more muted effect and all points near the gate are emitting current that is being absorbed by it, even when the electric field there is low.

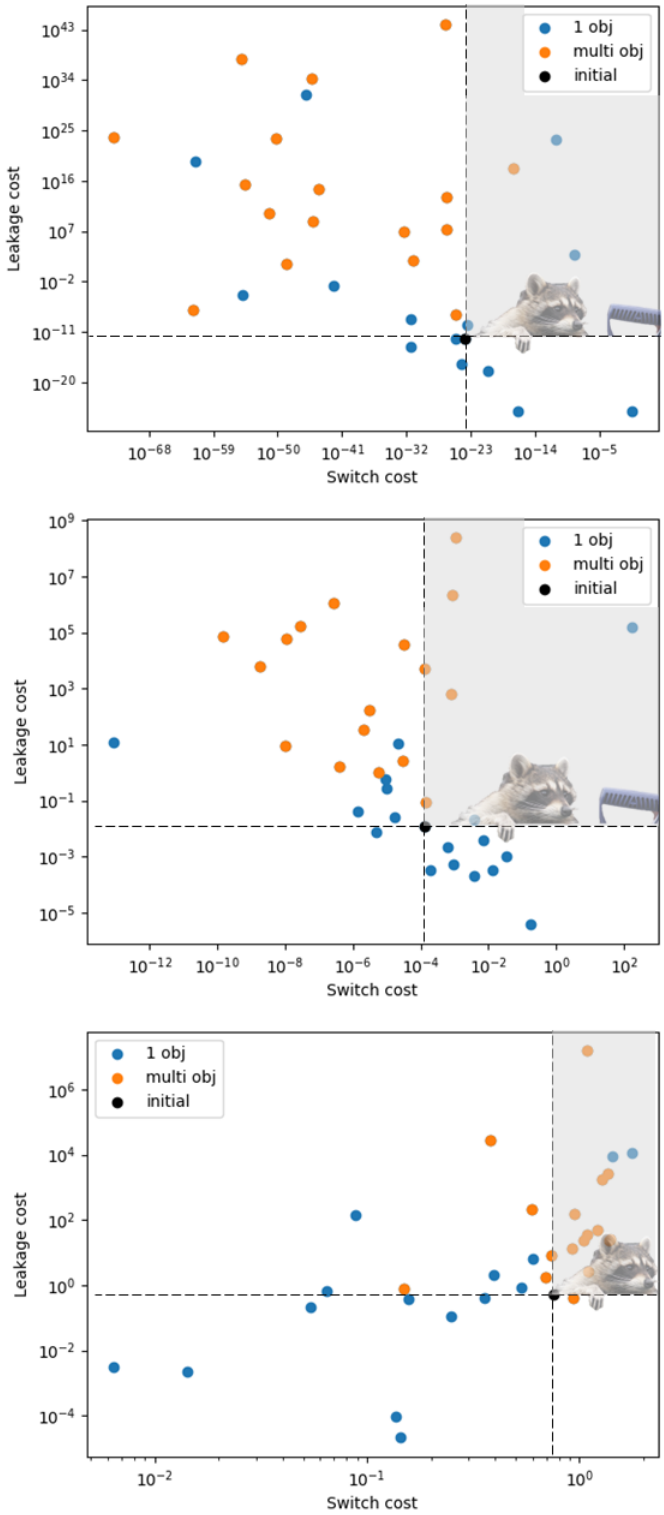

Figure 4-9: Optimizations plotted as function of their performance on switch cost and gate cost with (top) Fowler-Nordheim emission, (middle) enhanced Fowler-Nordheim, and (bottom) Fowler-Nordheim + Schottky. In each plot, the performance of the initial geometry is plotted in black, the results of optimizing only one cost function are in blue, and the results of regularized optimization are in orange. The raccoons are overlaid on the trash region of each sub-figure for demonstration.

4.7 Regularized cost result

Regularized cost functions are designed to balance and manage the trade-off between multiple objectives that might be working against each other. It is common to visualize a collection of optimizations with n objectives as points in $\mathbb{R}^n$ with each dimension representing performance on one objective, but more sophisticated visualizations exist for high number of objectives [26]. The Pareto front is the region in which there is an active trade-off between the objectives; in order to do better on one objective, you must sacrifice performance on another [17]. To visualize the benefit of our regularized optimization, we created fig. 4-9, in which we plotted the results from each optimization as a point in 2D where the x axis represents the switching cost (eq. (4.1)) and y axis represents the gate cost (eq. (4.2)). In each subplot, the initial solution is plotted in black with dashed lines through it dividing the space into regions with cost lower and higher than the initial cost in each dimension. The results of optimization with a single objective are plotted in blue, while the results of the regularized optimization are plotted in orange. The top right region is shaded to represent its irrelevance due to the fact that points in it suffer from a worse performance on both cost functions compared to the initial solution. In theory, that region should be empty because the optimizer has no incentive to return a geometry that preforms worse on both cost functions. However, we make approximations in calculating the cost in order to speed up the optimizations. Those approximations are valid in most cases, but do sometimes lead the optimizer astray. This issue does not invalidate the approximations altogether, but it means that the results of the optimizer cannot be trusted blindly. A more sensitive and complete simulation of the results needs to be run in order to verify them.

When allowed to converge fully, the regularized optimization points trace out the Pareto front, which is the curve bounding the optimization. Here, however, several limitations prevented that convergence from occurring. Those limitations along with the limitations that cause points to appear in the upper right corners of (trash region) the sub-figures of fig. 4-9 are discussed in section 4.8. Despite the limitations, we can see some interesting characteristics of the optimization from analyzing fig. 4-9.

Each sub-figure in fig. 4-9 visualizes the results of a set of optimizations run under the

assumption of a different emission mechanism. Broken lines are drawn through the point representing the initial geometry, cutting each sub-figure into four regions.

The upper-right region is defined as the trash region (raccoons are overlaid for demonstration). Points in the the trash region represent geometries that suffer from worse performance on both objective functions compared to the initial geometry.

The diagonally opposed upper-left and lower-right corners are defined as the trade-off regions. They represent solutions, that in order to achieve better performance on one metric, had to sacrifice performance on another metric. For emission models of Fowler-Nordheim and enhanced Fowler-Nordheim, we see that most points are in the union of the upper-left and lower-right regions.

The lower left region is the optimal region. There, the geometries perform better on both metrics when compared to the initial geometry. The optimizations with Fowler-Nordheim has a few points in that region, none of which were generated from a regularized optimization. The optimization with enhanced Fowler-Nordheim has one point in that region. Surprisingly, the Fowler-Nordheim + Schottky emission model has many points in the optimal region. Interestingly, none of the points in the optimal region were generated by a regularized optimization.

4.8 Limitations

In addition to the limitations discussed in section 3.6, which apply here as well, gated devices experience limitations unique to them.

4.8.1 Step size

Calculating the electric field lines is done iteratively by calculating the potential gradient, taking a step in the direction of the gradient, and repeating until a stopping condition is reached. See algorithm 3 for full algorithm. Assuming a step size ϵ, we have

$$(x_{new}, y_{new}) = \epsilon \nabla u(x, y) + (x, y)$$

The step size taken will affect the accuracy of the traced field line. Taking a smaller step size will result in a more accurate trace at the price of requiring more iterations to reach the end of the trajectory and increasing the runtime. In the optimization loop, we selected step size of $\epsilon = 0.4$ nm for a balance of accuracy and computational speed. When evaluating the results of the optimization, we use $\epsilon = 0.1$ nm for higher precision.

4.8.2 Bisection search

In the bisection search algorithm, we find the index of the last trajectory to end up on the collector and assume that all trajectories with indices smaller than it end up on the collector and that all trajectories with indices larger than it end up at the gate. We know that electric field lines do not intersect, so no trajectories with smaller index will end up on the gate. However, some of the trajectories do end up looping back into the emitter or flying off the edge of the simulation. Depending on their index, they will be counted as collector or emitter current, even though they should be excluded, as we do properly in the post optimization evaluation of the results. This limitation could be the main contribution to the disparity between the cost function evaluation happening during the optimization and the post optimization evaluation.

4.8.3 Particle tracking

The full physical model is a lot more complicated, it involves particles getting emitted, interacting with each other, and gaining momentum as they accelerate across the potential difference, and influencing the electric fields around them. Simulating the accurate particle-particle and particle-field interactions is computationally expensive, but it will be more accurate. Likely, the charge in the particles will alter the field seen at the emitter and suppress emission somewhat. By ignoring the space charge effects, we do not see the saturation at high voltages and we get probably more current than we should.

4.8.4 Gate emission

In our modeling we assumed that only the emitter emits current and that the collector and the gates collect currents. However, the gate is capable of emitting current, especially when there is a potential difference between it and the collector. This means that we were ignoring potential current seen at the collector due to gate emission when the gate is at low voltage.

4.9 Conclusion and outlook

In this chapter we explored two figures of merit to optimize in a transistor. The first is the switching behavior quantified as the ratio of the collector current when the gate is turning the device on to the collector current when the gate is turning the device off. The second is the gate leakage current which is quantified as the ratio of the gate current to the collector current. We optimized devices with these figures of merit using the framework described in chapter 2. Although the optimizations had limitations, we discovered useful features like the exact x coordinate to create sharp features that see an electric field influenced by the gate, emit current that is absorbed by the collector.

In future iterations, we should seek to optimize the gate and collector geometries in addition to the emitter. Additionally, we should verify the final results with a full particle in cell simulator.

Chapter 5

Conclusions, impact, and outlook

5.1 Conclusions

In this thesis we built a framework to simulate and optimize electron emitting nano vacuum channel diodes and transistors. We used FEniCS, an open source finite element solver with additional layers of home-brewed code to account for emission and particle tracking. We formulated cost functions to optimize the current emission from the diode and the switching behavior and gate leakage for the transistor. We ran the optimization for different parameterizations, different cost functions, different maximum iterations, and different emission models. We showed that un-intuitive device designs can be discovered by means of a relatively simple optimizations. The geometrical features that emerge from the optimization heavily depend on the emission mechanism assumed, which is yet to be fully determined in PNVCTs. For the diode, we notice a tip and bulb feature emerge form optimizing the current while assuming enhanced Fowler-Nordheim emission and Schottky + Fowler-Nordheim emission. For enhanced Fowler-Nordheim + Schottky we see a tendency towards particularly spiky geometries. For gated devices, we notice a symmetric double rounded tip emerge for the off/on figure of merit when assuming enhanced Fowler-Nordheim emission. We also notice the inward concavity near the gate for the minimum gate leakage figure of merit. In both the costs of transistor we see sharp tips and other features that encourage emission emerge at the exact x coordinate of the right edge of the gate, where the features see the field caused by the gate, but the current produced at the features is not pulled by the gate.

Instead, it makes it way to the collector.

Many simplifying assumptions and approximations were made in order to speed up the simulation, making it fast enough so that an optimization with thousands or tens of thousands of iterations can run in an acceptable time. The first approximation made was using a two dimensional electrostatic simulations instead of a three dimensional one. This approximation is accurate when the gap and the features of the emitter are much smaller than the height of the device in real life. However, for bigger gaps we might get factor differences and errors of 40%. An additional time saving simplification was the mesh resolution used in the electrostatic simulation. Instead of using an incredibly fine mesh, we used a resolution of 20 instead of 1,000 which would have given 2× more accurate results but also would have taken 250× longer to compute. An additional assumption that applies only to gated devices was the heuristic to particle tracking used to approximate the current at the collector and at the gate. We cannot provide a quantification of the error caused by this approximation because of lack of access to particle in cell simulations to use to calculate a more accurate figure to use for benchmarking.

The bisection search used to speed up the summation of the relevant particle trajectories can introduce errors. The search algorithm finds the first and last trajectories to arrive at the collector. It assumes that all the trajectories indexed by numbers in the range of the trajectories found end up at the collector. All other trajectories are assumed to end up at the gate. However, there are trajectories that get reflected back to the emitter or fly off the edge of the simulation which arbitrarily get added to the collector or gate current, erroneously inflating the collector or gate currents. In most cases, the optimized result for the approximate model maps to cost value that is better than the initial cost in a finer and more accurate simulation. However, there are geometries returned by the optimizer that when evaluated on a finer simulation, perform worse than the initial geometry. This by no means invalidates the optimization process or the approximate simulations, but it does establish that the results need to be verified with a finer simulation before moving on to the fabrication.

5.2 Impact

The work described in this thesis is impactful for several reasons. The optimization is useful at recommending geometries that are likely to be optimal. Those geometries can be fabricated, saving the device designers time, money, and effort on fabrication and testing. Additionally, the results of the optimization deepen our understanding of the device physics and the models that are used to describe it. When unexpected geometrical features arise from the optimization, we are confronted with new information that causes us to rethink our modeling and leads to a better understanding of the model. Finally, this work establishes a framework for an optimization process that is completely open source and publicly accessible. Device designers will be able to download our code, apply the relevant changes, and explore optimization at no cost and without needing to acquire academic licenses for commercial software packages.

5.3 Next steps

Some suggestions for immediate futures steps to improve the optimizer are included in chapters 3 and 4 as suggestions. Those suggestions include forcing symmetry on the parameterization, implementing a dynamic step size optimizer, and optimizing the shapes of the collector and emitter. Additional ideas for optimization improvement are given in the next sections.

5.3.1 Optimization under voltage sweep

The devices modeled and optimized in this work are operated under a range of voltages. Diodes are operated under a one dimensional collector-emitter voltage. Transistors have a 2D voltage sweep range, comprising of the sweep in collector-emitter voltage and the sweep in gate-emitter voltage. While the electric field solution is linear, the current emission and the particle flight in space are not linear in the voltage difference. Therefore, a more complete optimization framework will consider optimizing and evaluating a device over its full operating range. This will require many more simulations per optimization, but might

ultimately lead to a more informed optimization with overall better results.

5.3.2 Fabrication constraints

While the optimizer might attempt to create infinitely sharp corners and infinitely narrow features, the fabrication is physically limited by the grain size of the material that is deposited to make the device. To prevent infinitely sharp corners from occurring, we used a spline interpolation between the parameter points, which prevents sharpness, but still allows features with small radii of curvature. To prevent features with a small radii of curvature, we can directly penalize occurrences of radii below a certain threshold and add the penalizing term directly to the cost function. Alternatively, we can add a minimum radius of curvature as a constraint of the optimization.

Very narrow features are not physically impossible to fabricate, but are not robust because of the fabrication variations and the tendency of narrow structures to be fragile and break off. To discourage the optimizer from gravitating towards narrow and non robust structures, we can add an additional term to the cost function penalizing the surface area to volume ratio. The penalizing term will have a lowest value when the object is spherical and highest when the shape irregular and windy.

The features described in this section are not yet implemented, but can be easily implemented with help from already existing python packages such as Scipy and Shapely [41, 56].

5.4 Outlook

This work highlights the value and power of shape optimization and sets the stage for extending its usage to a variety of related projects. We are working on applying shape optimization to fabrication aware design, adjoint optimization, and optical applications.

5.4.1 Fabrication-aware design

The nano fabrication process suffers not only from hard physical limitations such as the atomic size or grain size of the material used, but also from limitations arising from the state of fabrication technology.

NVCT devices are fabricated by layering a resist on the substrate, and then exposing the resist to an electron beam at predetermined locations for some duration of time. Then, the resist is put through development which breaks it down at the locations that were exposed beyond some threshold. Then, the material for the devices is deposited into the vacancies created by the resist breakdown. Eventually, the resist mask is washed away and the deposited structure remains [45]. Additional steps can be applied to create an undercut, but we will focus on the other parts of the process. The electron typically is assumed to have a Gaussian spatial profile, which along with the threshold process for the resist development and the finite grain size of deposited material, almost guarantee that the layout attempted won't match the final shape produced very well, especially in regions of very fine features. There is a lot of art and acquired expertise in the process of tinkering a layout given to a fabrication tool to get a desired target shape with sharp features at the other side. We wish to include consideration for the fabrication process in the optimization, which can be done in a several ways.

We could run the optimization as before, and after the final iteration, reverse engineer a layout that will produce the closest shape to the target when placed in a nano fabrication tool. Going from a target design to the layout is an inverse design problem which is a particular flavor of optimization. This method will be the fastest of the methods proposed here, but it is problematic in some cases. For example, if the optimizer generates a target shape that cannot be fabricated, the closest fabricable shape might have a completely different performance which is considerably sub-optimal.

Another method would be to bake a fabrication simulation into every iteration of the optimization. This will make the "decide if to keep" step evaluate the cost function on the shape that emerges on the other size of the fabrication simulation. The benefits of this method are that the optimizer will seek to optimize the the fabricated shape, but will return

the layout for it once converged ion addition to not returning shapes that are unfabricable. This method promises the most accuracy and usefulness, but is also incredibly slow and will require coding an involved simulator with many steps.

The third approach acts like a compromise between the two methods suggested above. We can add a layer that applies a Gaussian filter convolution to the parameter shape. The convolution will smooth out harsh features and will simulate the effects of the electron beam without requiring an intense fabrication simulation. It might be less accurate than a method using a full fabrication simulation, but it is a lot faster and likely to produce a result that is close enough to the result of the involved simulator.

5.4.2 Adjoint optimization

Finite difference is an intuitive and straight forward method for calculating a performance gradients, though it is not nearly the most efficient one. Calculating a cost gradient using finite difference requires $n + 1$ simulation solves where n is the number of parameters that are optimized over. The adjoint method can be used to calculate a performance gradient of a system efficiently, in only two system solves. It is used a lot in photonic design [21] and is beginning to get applied in electron emitting and electron path devices [37].

For a system simulation that solves

$$g(\mathbf{x}, p) = 0$$

with parameters p, state vector $\mathbf{x}$, and a merit equation

$$f(\mathbf{x}, p),$$

we derive the performance gradient $\frac{df}{dp}$ with tow system solves as follows

$$g_{\mathbf{x}}^{\top}\lambda = -f_{\mathbf{x}}^{\top} \tag{5.1}$$

$$\frac{df_p}{dp} = \lambda^{\top} g_p \tag{5.2}$$

[35]. In our work, the Laplace equation gets projected into a finite element space and is solved by a matrix equation

$$\partial\Phi\mathbf{a} - L = 0, \tag{5.3}$$

where $\partial\Phi$ is a matrix of the form

$$\partial\Phi = \begin{bmatrix} \frac{\partial\phi_1}{\partial x_1} & \frac{\partial\phi_1}{\partial x_2} \\ \frac{\partial\phi_2}{\partial x_1} & \frac{\partial\phi_2}{\partial x_2} \\ \vdots & \\ \frac{\partial\phi_n}{\partial x_1} & \frac{\partial\phi_n}{\partial x_2} \end{bmatrix}.$$

The ϕ_i are the basis functions for the finite element method. The system variables $\mathbf{x}$ can be thought of as the set of coefficients of the basis functions $\mathbf{a}$, such that

$$U = \sum_j a_j \phi_j. \tag{5.4}$$

The merit function $f(\mathbf{a}, p)$, contains the nonlinear emission that depends on the electric field

$$E = -\sum_j a_j \nabla\phi_j$$

FEniCS allows users to access the deepest levels of code including nodal quantities and basis functions. Thus, we have access to the $\partial\Phi$ matrix and can write an adjoint optimization using the mesh quantities directly. We have done most of the derivation in order to implement this. However, the node ordering is not immediately intuitive and has been preventing us from moving forward.

5.4.3 Adjoint method for particle tracking

The work in chapter 4 relies on a heuristic to particle tracking. We considered the heuristic as a fast approximation to the particle in cell (PIC) simulation. We could however, afford to use a PIC if we can compute the direction of steepest descent efficiently, making the optimizer converge quickly. Calculating an adjoint method to a PIC is unintuitive because

the particle emission and progression in space are highly non-linear. An additional difficulty comes from the fact that the cost functions depend on currents that are defined as a sum of discrete particles arriving at a target. Taking a gradient of a discrete quantity is mathematically undefined. A work by Antenson et al. relies on creative use of the Green identities and Poisson's equation relating charge and potential to circumvent the discrete quantities when calculating the adjoint for a PIC simulation in an electron gun [34]. We can use Antenson's formulation for the particle progression in time, but it does not account for field dependant emission. We consider the option of including a field dependent emission in the formulation and re-deriving the adjoin, or alternatively, alternating between optimizing the electron trajectories with the PIC adjoint and optimizing the emission with the method in section 5.4.2.

5.4.4 Optimization for optical emission

One of the many advantage of NVCTs is that they can be driven optically by ultrafast pulses. Their low capacitance combined with the ballistic transport of the electrons through the channel, allow for almost instantaneous current responses to the driving optical pulses. PNVC device can sample the waveform of an 100THz optical pulse [43], they are sensitive to the carrier envelope phase (CEP), respond to pulses with polarization aligned with the geometry , and suppress pulses with perpendicular polarization. Previous work in ultrafast optics shows that when NVCT devices are operated optically, their response is heavily dependant on their geometry. A work by Buckley et al. compared the resonant responses of several antennas of slightly different geometries, but with the same radius of curvature at the tip. They found a significant differences in the resonant response and CEP sensitivity for the different structures[44]. That work highlights the potential for shape optimization in this area. Shape optimization can be used to design devices with a target resonant frequency response, optimize devices for CEP detection, or to have a tailored polarization response.

The cost function for the shape optimization here will have to account for the time dependence of the exciting electric field and the physics of the light matter interaction. We will have to run a time domain simulation which aggregates the response of the device to the excitation field and calculates the desired figure of merit. MEEP is an open source python

package that specializes in optical simulations. It has a built in adjoint method which can be used for local shape optimization of a structure [54].

Appendix A

Code

A.1 Simulation code

This appendix chapter contains simulation code referenced in the thesis text.

```
# import relevant libraries
from fenics import *
from dolfin import *

# Define the solution domain using geometrical objects
emitter = Polygon([Point(coords[0][i], coords[1][i]) for i in
    range(len(coords[0]))])
gate_top = Rectangle(Point(-20, 30), Point(-10, ymax))
gate_bottom = Rectangle(Point(-20, ymin), Point(-10, -30))
domain = Rectangle(Point(xmin, ymin), Point(xmax, ymax)) - emitter - gate_top -
    gate_bottom

# Create the mesh
mesh = generate_mesh(domain, 20)
```

```
# select a function space and define trial functions and test functions
function_space = FunctionSpace(mesh, 'CG', 1)
```

```
trial_fxn = TrialFunction(function_space )
test_fxn = TestFunction(function_space )
```

```
#  variational method
a = inner(grad(trial_fxn), grad(test_fxn))*dx
L = Constant(0.0)*test_fxn*dx
```

```
# boundary conditions
out = DirichletBC(function_space, Constant(15), out_boundary)
em = DirichletBC(function_space, Constant(0), on_emitter)
gate = DirichletBC(function_space, Constant(17), on_gate)

bcs = [out, em]
```

```
def on_emitter(x, on_boundary):
    return on_boundary and (intersect_surface(np.transpose(out2), x, tol) or
        intersect_surface_2(np.transpose(out2), x, tol))

def intersect_surface(outline_points, point_star, tol):
    """ used for determaining points on the inner boundary for boundary conditions.
    Takes in a test point and an array of points which define a surface.
    The code linearly iterates through the outline array and checks whether the
        test point is on the line connecting point i with point i+1.
    Returns a boolean value
    TODO: find a better and more efficient way to calculate this
    """
    x_star, y_star = point_star
    is_on = False
    for i in range(len(outline_points[:, 0]) -1):
        if abs(outline_points[i, 0] - outline_points[i+1, 0]) < tol:
            if abs(x_star - outline_points[i, 0]) < tol and ((y_star <=
                outline_points[i,1] and y_star >= outline_points[i+1,1]) or (y_star
```

```
                >= outline_points[i,1] and y_star <= outline_points[i+1,1])):
16                  is_on = True
17          elif abs(outline_points[i, 1] - outline_points[i+1, 1]) < tol:
18              if abs(y_star - outline_points[i, 1]) < tol and ((x_star <=
                    outline_points[i,0] and x_star >= outline_points[i+1,0]) or (x_star
                    >= outline_points[i,0] and x_star <= outline_points[i+1,0])):
19                  is_on = True
20          elif (x_star <= outline_points[i,0] and x_star >= outline_points[i+1,0])
                or (x_star >= outline_points[i,0] and x_star <= outline_points[i+1,0]):
21              m = (outline_points[i,1] - outline_points[i+1,1])/(outline_points[i,0]
                    - outline_points[i+1,0])
22              b = outline_points[i,1] - m*outline_points[i,0]
23              if abs(y_star - m*x_star - b) < tol:
24                  is_on = True
25      return is_on
26
27  def intersect_surface_2(outline, point_star, tol):
28      poly = [(outline[i, 0], outline[i, 1]) for i in range(outline.shape[0])]
29      line = shapely.geometry.Polygon(poly).buffer(tol)
30      point = shapely.geometry.Point(point_star[0], point_star[1]).buffer(tol)
31      return line.intersects(point)
```

```
1   # assemble the system into matrices with boundary conditions
2   A, b = assemble_system(a, L, bcs)
3
4   # define solver parameters
5   solver = KrylovSolver('cg', 'ilu')
6
7   # allocate a function space function and vector for the solution
8   u = Function(function_space)
9   U = u.vector()
10
```

```
# solve
solver.solve(A, U, b)
```

```
V_g = VectorFunctionSpace(mesh, 'CG', 1)
v = TestFunction(V_g)
w = TrialFunction(V_g)
a = inner(w, v)*dx
L = inner(grad(u), v)*dx
grad_u = Function(V_g)
solve(a == L, grad_u)
```

```
def fowler_nordheim_emission(E_strength, area, phi):
    """ Takes in an electric fielsd magnitude, area, and work function. Determains
        the current due to fowler nordheim emission in that area
    """
    afn = 1.5414e-6 # eV v^-2
    bfn = 6.8309 # eV^(-3/2)V/nm
    nu = 1
    J = afn*(np.power(E_strength, 2)/phi)*np.exp(np.divide(-nu*bfn*np.power(phi,
        3/2), E_strength, out=np.zeros_like(E_strength), where=E_strength!=0))
    return J*area
```

```
def find_ppoints(x, y, epsilon=1/20):
    """ The code takes in an array of x values and an array of y values. Together
        they describe an outline of a shape.
    The code uses the cross product to generate the outward pointing normal for
        every point on the outline.
    The outward normal is used to find a new point that is in the simulation
        domian, just outside the shape.
    This is used to find a point just outside the emitter to test the e field in
        because the gradient isn't defined on the boundary (apperantly)
    """
```

```
    x_new = []
    y_new = []
    for i in range(len(x)-1):
        x_vec = x[i+1] - x[i]
        y_vec = y[i+1] - y[i]
        norm_factor = np.sqrt(x_vec**2 + y_vec**2)/epsilon
        x_vec /= norm_factor
        y_vec /= norm_factor
        x_new.append( x[i] + y_vec)
        y_new.append(y[i] - x_vec)
    return x_new, y_new
```

```
def find_ppoints(x, y):
    """ The code takes in an array of x values and an array of y values. Together
        they describe an outline of a shape.
    The code uses the cross product to generate the outward pointing normal for
        every point on the outline.
    The outward normal is used to find a new point that is in the simulation
        domian, just outside the shape.
    This is used to find a point just outside the emitter to test the e field in
        because the gradient isn't defined on the boundary (apparently)
    """
    new_points = np.zeros((len(x)-1, 2))
    x_vec = x[1:] - x[:-1] # x_vec = x[i+1] - x[i]
    y_vec = y[1:] - y[:-1] # y_vec = y[i+1] - y[i]
    norm_factor = 10*np.sqrt(np.power(x_vec, 2) + np.power(y_vec, 2))
    x_vec = np.divide(x_vec, norm_factor)
    y_vec = np.divide(y_vec, norm_factor)
    new_points[:, 0] = x[:-1] + y_vec
    new_points[:, 1] = y[:-1] - x_vec
    return new_points
```

```
new_points = find_ppoints(x_coords, y_coords)
magnitude = np.zeros_like(new_points[:, 0])
area = np.zeros_like(new_points[:, 0])
for i in range(len(new_points[:,0])):
    grad_at_x = grad_u(Point(new_points[i, 0], new_points[i, 1])) #Point(x[i],
        y[i])
    dot_product = (new_points[i, 0] - x_coords[i])*grad_at_x[0] + (new_points[i,
        1] - y_coords[i])*grad_at_x[1]
    if dot_product >0:
        # if the dot product is negative, the current is pointing outward and can
            be counted
        magnitude[i] = np.sqrt((grad_at_x[0])**2 + (grad_at_x[1])**2)*1e9
            # don't need an else because the value will be zero by default
    if i<len(new_points[:, 0]) - 1:
        seg1 = np.sqrt((new_points[i, 0] - new_points[i-1, 0])**2 + (new_points[i,
            1] -new_points[i-1, 1])**2)
        seg2 = np.sqrt((new_points[i+1, 0] - new_points[i, 0])**2 +
            (new_points[i+1, 1] -new_points[i, 1])**2)
        area[i] = 1e-9*(seg1+seg2)/2
    else:
        seg1 = np.sqrt((new_points[i, 0] - new_points[i-1, 0])**2 + (new_points[i,
            1] -new_points[i-1, 1])**2)
        seg2 = np.sqrt((new_points[0, 0] - new_points[i, 0])**2 + (new_points[0,
            1] -new_points[i, 1])**2)
        area[i] = 1e-9*(seg1+seg2)/2

total_current = np.sum(fowler_nordheim_emission(magnitude, area, 5.3))*40e-9 * 1e6
    # get current as micro amps
```

```
paths = []
for i in range(len(x)):
```

```
    grad_at_x = grad_u(Point(x[i], y[i]))
    ...
    """calculate dot product, magnitude, and area like above"""
    ...
    if dot_product>0 :
        current_path = [(x[i], y[i])]
        x_new = x[i] + grad_at_x[0]
        y_new = y[i] + grad_at_x[1]
        for q in range(400):
            if x_new >= xmax: # reached collector
                current_path.append((x_new, y_new))
                collector_currents[i] = magnitude[i]
                break
            elif gated_device and reached_gate([x_new, y_new]): # reached gate
                current_path.append((x_new, y_new))
                gate_currents[i] = magnitude[i]
                break
            elif y_new >= ymax or y_new <= ymin or x_new <= xmin or
                intersect_surface(np.transpose(out2), [x_new, y_new], 5e-1): #
                esceped through the side or looped back to emitter
                current_path.append((x_new, y_new))
                break
            elif np.sqrt((grad_at_x[0])**2 + (grad_at_x[1])**2) <= 0.1:
                current_path.append((x_new, y_new))
                if np.sqrt((x[i]-x_new)**2 + (y[i] - y_new)**2) < 10:
                    break
                else:
                    prev_x, prev_y = current_path[-2]
                    momentum_x = x_new - prev_x
                    momentum_y = y_new - prev_y
                    grad_at_x = grad_u(Point(x_new, y_new))
                    x_new = x_new + momentum_x
```

```
                    y_new = y_new + momentum_y
            else:
                current_path.append((x_new, y_new))
                grad_at_x = grad_u(Point(x_new, y_new))
                x_new = x_new + grad_at_x[0]
                y_new = y_new + grad_at_x[1]
        paths.append(current_path)

def reached_gate(x):
    return (x[0] >= -15 and x[0] <= 0) and (x[1] >= 35 or x[1] <= -35)
```

```
## for y > 0
high = 60
low = 0
middle = round((high + low)/2)
found = False
iterations_of_search = 0
skipped = 0
while (high - low) > 1 and not found:
    i = middle
    x_star = new_points[i, 0]
    y_star = new_points[i, 1]
    grad_at_x = grad_u(Point(x_star, y_star))
    trajectory_end = are_we_there_yet(grad_u, x_coords[i], y_coords[i], x_star,
        y_star, grad_at_x, xmin, xmax, ymin, ymax, out2, 0, num_iter = 300)
    iterations_of_search += 1
    if trajectory_end == "gate" or trajectory_end == "edge-l" or middle == high:
        high = middle - skipped
        middle = round((high + low)/2)
        skipped = 0
    elif trajectory_end == "collector":
        low = middle
```

```
        middle = round((high + low)/2)
        skipped = 0
    elif trajectory_end == "edge-r" or trajectory_end == "zero":
        middle += 1
        skipped += 1
        if middle == low:
            found = True
    elif middle >= high or middle <= low:
        found = True
    if iterations_of_search > 10:
        print("stuck!!")

## for y < 0
high2 = len(new_points[:, 0]) - 1
low2 = high2 - 60
middle2 = round((high2 + low2)/2)
found2 = False
iterations_of_search2 = 0
skipped = 0
while (high2 - low2) > 1 and (not found2):
    i = middle2
    x_star = new_points[i, 0]
    y_star = new_points[i, 1]
    grad_at_x = grad_u(Point(x_star, y_star))
    trajectory_end = are_we_there_yet(grad_u, x_coords[i], y_coords[i], x_star,
        y_star, grad_at_x, xmin, xmax, ymin, ymax, out2, 0, num_iter = 400)
    iterations_of_search2 += 1
    if trajectory_end == "gate" or trajectory_end == "edge-l":
        low2 = middle2
        middle2 = int((high2 + low2)/2)
        skipped = 0
    elif trajectory_end == "collector" or middle2 == high2:
```

```
        high2 = middle2 - skipped
        middle2 = int((high2 + low2)/2)
        skipped = 0
    elif trajectory_end == "edge-r" or trajectory_end == "zero":
        middle2 += 1
        skipped += 1
        if middle2 == low2:
            found2 = True
    elif middle2 >= high2 or middle2 <= low2:
        found2 = True
    if iterations_of_search2 > 10:
        print("stuck!!")
```

```
plt.figure()
plot(u, vmin= min_v, vmax=max_v)
plt.show()
```

```
max_current = max(emission)
        min_current = min(emission)
        for i, path in enumerate(paths):
            x_path = [tup[0] for tup in path]
            y_path = [tup[1] for tup in path]
            width = 3*(emission[i] - min_current + 0.2)/max_current
            plt.plot(x_path, y_path, 'r', linewidth=width)
```

A.2 Optimization code

This appendix chapter contains the optimization code and the utility scripts referenced in the thesis text.

```
from optimizer.simulated_annealing import simulated_annealing
```

```
import optimizer.objective_2terminal as ug_objective
import optimizer.objective_gated as g_objective
import utils.plot_currents as plotter

if params["optimization"]["device"] == "ungated":
    save_directory = "ungated"
    params["optimization"]["save directory"] = save_directory
    solver = simulated_annealing(x0, ug_objective.objective,
        params["optimization"]["max iteration"], params["optimization"]["initial
        temp"], upper_bound = ub, lower_bound = lb, parameters = params) #
        initialize
else:
    save_directory = "gated"
    params["optimization"]["save directory"] = save_directory
    solver = simulated_annealing(x0, g_objective.objective,
        params["optimization"]["max iteration"], params["optimization"]["initial
        temp"], upper_bound = ub, lower_bound = lb, parameters = params) #
        initialize

final = solver.run_annealing(upper_bound = ub, lower_bound = lb, parameters =
    params) # solve
```

```
params["optimization"]["run time"] = end - start
params["optimization"]["error rate"] =
    solver.error_number/params["optimization"]["max iteration"]

np.savetxt(save_directory + "/final_solution_{}.csv".format(fname), final,
    delimiter=",")
np.savetxt(save_directory + "/accepted_cost_{}.csv".format(fname),
    accepted_cost_arr, delimiter=",")
np.savetxt(save_directory + "/best_cost_{}.csv".format(fname), best_cost_arr,
    delimiter=",")
```

```

# plot geometry, merits, and current distribution
solver.plot_final_geometry(save_directory + "/" + fname + "geometry")
solver.plot_merits(save_directory + "/" + fname + "merits")

p = plotter.plot_currents(params, final)
p.plot_trajectories(save_directory + "/" + fname)

#solver.plot_final_geometry()
with open("opt_log2.txt", 'r') as f:
    database = json.load(f)
    database.append(params)
with open("opt_log2.txt", 'w') as f:
    json.dump(database, f)
```

```
class simulated_annealing:
    def __init__(self, x0, cost, num_iterations, initial_T, upper_bound=None,
        lower_bound=None, parameters=None):
        self.num_iterations = num_iterations
        self.current_iteration = 0
        self.converged = False
        self.initial_T = initial_T
        self.current_T = initial_T
        self.const_pert_size = parameters["optimization"]["const pert"]
        self.initial_solution = x0
        self.current_solution = x0
        self.try_solution = x0
        self.best_solution = x0
        self.initial_objective = cost(x0, parameters["geometry"]["xmin"],
            parameters["geometry"]["xmax"], parameters["geometry"]["ymin"],
            parameters["geometry"]["ymax"])
        self.try_objective = self.initial_objective
```

```
        self.current_objective = self.initial_objective
        self.best_objective = self.initial_objective
        self.upper_bound = upper_bound
        self.lower_bound = lower_bound
        self.cost = cost
        self.T_change = parameters["optimization"]["temperature scale"]
        self.try_cost_arr = np.zeros(num_iterations)
        self.accepted_cost_arr = np.zeros(num_iterations)
        self.best_cost_arr = np.zeros(num_iterations)
        self.error_number = 0
        self.run_id = parameters["optimization"]["run_num"]
        self.save_directory = parameters["optimization"]["save directory"]
        self.smooth_pert = parameters["optimization"]["perturb smoothly"]
        self.symmetry = parameters["optimization"]["symmetry"]
        self.log_errors = parameters["optimization"]["log errors"]
        self.geomtry_params = parameters["geometry"]
```

```
def perturb(self):
    """
    Inputs: self
    Outputs: None
    For each parameter point, p(i), this method computes 2 normal vectors: (p(i
        +1) - p(i)) x z and (p(i) - p(i-1)) x z
    where z is the unit vector in the direction out of the plane.
    The average of the normal vectors is used to move the point p(i) to a new
        location p(i)*.
    Each point is moved along the line normal to the surface through that point by
        a random amount drawn from a uniform distribution.
    If the random movement tries to push a point beyond a boundary, the algorithm
        resets it to the boundary.
    """
    new_points = np.zeros_like(self.current_solution)
```

```
        delta_x1 = np.zeros_like(self.current_solution)
        delta_x2 = np.zeros_like(self.current_solution)
        x_vec = self.current_solution[1:, 0] - self.current_solution[:-1, 0] # x_vec =
            x[i+1] - x[i]
        y_vec = self.current_solution[1:, 1] - self.current_solution[:-1, 1] # y_vec =
            y[i+1] - y[i]

        if self.const_pert_size:
            epsilon = 1/2
        else:
            epsilon = np.sqrt(self.current_T)/5

        norm_factor = np.divide(epsilon, np.sqrt(np.power(x_vec, 2) + np.power(y_vec,
            2)))

        delta_x1[:-1, 0] = np.multiply(norm_factor, self.current_solution[1:, 1] -
            self.current_solution[:-1, 1]) # norm_factor*(y[i+1] -y[i])
        delta_x1[:-1, 1] = - np.multiply(norm_factor, self.current_solution[1:, 0] -
            self.current_solution[:-1, 0]) # norm_factor*(x[i+1] -x[i])
        delta_x1[-1, :] = delta_x1[0, :]
        delta_x2[1:, 0] = np.multiply(norm_factor, self.current_solution[1:, 1] -
            self.current_solution[:-1, 1]) # norm_factor*(y[i+1] -y[i])
        delta_x2[1:, 1] = - np.multiply(norm_factor, self.current_solution[1:, 0] -
            self.current_solution[:-1, 0]) # norm_factor*(x[i+1] -x[i])
        delta_x2[0, :] = delta_x2[-1, :]
        q = np.random.uniform(low=-0.5, high=0.5, size=(delta_x1.shape[0],))
        q[-1] = q[0]
        if self.smooth_pert:
            ## kernel size hard coded and can be reset. This might actually be a
                disaster considering that the mean is zero
            kernel_size = 3
            kernel = np.ones(kernel_size) / kernel_size
```

```
        q = np.convolve(q, kernel, mode='same')
    # to set back of full device to not change, uncomment the line below. Notice
        that this only works for the original geometry with 88 points.
    #q[21:-21] = 0
    new_points[:, 0] = self.current_solution[:,0] + np.multiply(delta_x1[:, 0]/2 +
        delta_x2[:, 0]/2, q)
    new_points[:, 1] = self.current_solution[:,1] + np.multiply(delta_x1[:, 1]/2 +
        delta_x2[:, 1]/2, q)

    ##  implement constraints
    if self.upper_bound is not None:
        new_points[new_points>self.upper_bound] =
            self.upper_bound[new_points>self.upper_bound]
    if self.lower_bound is not None:
        new_points[new_points<self.lower_bound] =
            self.lower_bound[new_points<self.lower_bound]
    # register perturbation
    self.try_solution = new_points
```

```
def decide_if_to_keep(self, parameters):
    """ Decide whether or not to keep the perturbation
    Inputs: self, dict: parameters
    Outputs: None
    This function attempts to call the cost function on the current proposed
        solution which is saved in self.try_solution
    If delta cost < 0: accept solution with no further checks.
    If delta cost >=0: draw from acceptance probability and then accept or discard.
    If except block is reached, the error count will increase and if the debugger
        mode is activated, the error will be logged.
    """
    try:
        self.try_objective = self.cost(self.try_solution,
```

```
            self.geomtry_params["xmin"], self.geomtry_params["xmax"],
            self.geomtry_params["ymin"], self.geomtry_params["ymax"])
        delta_cost = self.try_objective - self.current_objective
        self.try_cost_arr[self.current_iteration] = self.try_objective
        if delta_cost <0:
            # goes downhill -> accept
            self.current_solution = self.try_solution
            self.current_objective = self.try_objective
            if self.current_objective < self.best_objective:
                self.best_objective = self.current_objective
                self.best_solution = self.current_solution
        else:
            prob = np.exp(-delta_cost/self.current_T) # is this good enough? We'll
                find out
            r = np.random.uniform(0, 1)
            if r <= prob:
                # accept with this probability
                self.current_solution = self.try_solution
                self.current_objective = self.try_objective
    except Exception as e:
        self.try_cost_arr[self.current_iteration] = float("nan")
        self.error_number +=1
        if self.log_errors:
            print(e)
            print(self.error_number)
            elements = [True, False]
            probabilities = [0.3, 0.7]
            dec = np.random.choice(elements, 1, p=probabilities)[0]
            if dec:
                np.savetxt(self.save_directory +
                    "/errors/err_geom_run{}_error{}.csv".format(self.run_id,
                    self.error_number), self.try_solution, delimiter=',')
```

```
    self.accepted_cost_arr[self.current_iteration] = self.current_objective
    self.best_cost_arr[self.current_iteration] = self.best_objective
```

```
def run_annealing(self, upper_bound=None, lower_bound=None, parameters=None):
    """
    start the annealing process
    TODO: update iter number, temperature
    TODO: Set bounds/ constraints """

    if upper_bound is not None:
        self.upper_bound = upper_bound
    if lower_bound is not None:
        self.lower_bound = lower_bound

    for i in range(self.num_iterations):
        # update iteration
        self.current_iteration = i
        # update temperature (newton cooling maybe?)
        if "lin" in self.T_change:
            self.current_T = self.initial_T*(1-i/self.num_iterations)
        elif "exp" in self.T_change:
            self.current_T = self.initial_T*np.exp(-i*10/self.num_iterations)
        elif "inv" in self.T_change or "over" in self.T_change:
            self.current_T = self.initial_T/(1 + i)

        # perturb
        self.perturb()
        self.decide_if_to_keep(parameters)
    return self.best_solution
```

```
def plot_final_geometry(self, fname):
    """
```

```
    Inputs: self, string: fname
    Outputs: None

    Plots the final geometry (spline interpolation of points) and saves the figure
        as fname.png
    """
    x_coords = self.best_solution [:, 0]
    y_coords = self.best_solution[:, 1]
    tck, u = spint.splprep([x_coords, y_coords], s=0)
    unew = np.arange(0, 1.005, 0.005)
    out = spint.splev(unew, tck)

    out[0][-1] = out[0][0]
    out[1][-1] = out[1][0]

    plt.figure()

    plt.plot(out[0], out[1], x_coords, y_coords, 'rx')
    plt.plot([self.geomtry_params["xmin"], self.geomtry_params["xmax"]],
        [self.geomtry_params["ymin"], self.geomtry_params["ymin"]], 'k')
    plt.plot([self.geomtry_params["xmin"], self.geomtry_params["xmax"]],
        [self.geomtry_params["ymax"], self.geomtry_params["ymax"]], 'k')
    plt.plot([self.geomtry_params["xmin"], self.geomtry_params["xmin"]],
        [self.geomtry_params["ymin"], self.geomtry_params["ymax"]], 'k')
    plt.plot([self.geomtry_params["xmax"], self.geomtry_params["xmax"]],
        [self.geomtry_params["ymin"], self.geomtry_params["ymax"]], 'k')
    plt.savefig(fname + ".png")
    plt.show()
    plt.close()

def plot_merits(self, fname):
    """
```

```
30      Inputs:
31      Outputs:
32
33      Plots the merits over the iterations
34      """
35      plt.figure()
36      plt.semilogy(range(self.num_iterations), -1*self.accepted_cost_arr,
            range(self.num_iterations), -1*self.best_cost_arr)
37      plt.legend(['accepted cost', 'best cost'])
38      plt.savefig(fname + ".png")
39      plt.show()
40      plt.close()
```

Appendix B

Tables

B.1 Config

This section contains parameter tables for the config.yml file.

B.2 Simulated annealing

This section contains parameter tables for the simulated annealing class.

parameter	values	description
device	String: ungated/-gated	Which type of device is being optimized.
cost	String: cost function	Cost function description. Options are: -current (applies for ungated only), switch, leakage, and regularized (apply for gated only).
max iteration	Integer: 1-inf	How many iterations the optimizer will run for. A good value is at least 1000.
initial geom	String: filename	Name of file from which to select initial geometry. Values include initial_rounded, initial_reduced
temperature scale	string	How should temperature decrease throughout the simulated annealing optimization. Supported options are linear, exponential, and inverse.
initial temp	Float: default 100	Initial temperature in the simulated annealing optimization
emission symmetry	Boolean	Whether symmetry should be imposed across the horizontal axis. Default value is False.
log errors	Boolean	Whether geometries that crash FEniCS should be saved along with the error messages for debugging. Default value is False.
const pert	Boolean: default True	Whether the size of the perturbation should remain constant throughout the optimization. Setting the parameter to False will result
perturb smoothly	Boolean: default False	Whether or not a smoothing filter should be applied to the perturbation.
start run number	Integer: 0-inf	ID number for the first run of optimizer.
optimization runs	Integer: 1	The number of identical optimizations to run with the run of main.py
comment	String	Any comment the use deems worthy of appearing in the matadata of the optimization run.
show plots	Boolean	Whether or not main.py should show the plots of the final geometry and cost values of each optimization.
lambda	Float: 0.0-1.0	When regularized optimization is run, this parameter controls the scaling of the two cost functions with respect to each other.

Table B.1: Optimization parameters found in config.yml. These can be changed by a user to customize the optimization to their needs.

parameter	values	description
x_gap	float: default 20	Gap in nm between emitter and collector in the x direction. Descriptive.
y_gap:	float: 50	Gap in nm between 2 gates in the y direction. Relevant for gated devices. Descriptive.
w	float: 100	Width in nm of emitter base. Descriptive.
wg	float: 30	Width in nm of gates. Descriptive.
l	float: 100	Length in nm of emitter along the x dimension. Descriptive.
lg	float: 100	No idea, but doesn't matter beacuse it's just descriptive.
h	float: 50	Height in nm of emitter triangle. Descriptive.
hg	float: 50	
xmin	float: -170	The smallest x value edge of the simulation.
xmax	float: 20	The largest x value edge of the simulation.
ymin	float: -60	The smallest y value edge of the simulation.
ymax	float: 60	The largest y value edge of the simulation.
margin	float: 5	How close to the edge of the simulation can the emitter reach.

Table B.2: Geometry parameters in config.yml. We do not advise the user to change these parameters.

parameter	type	description
num_iterations	int	Number iterations for the optimizer
current_iteration	int	The current iteration of the optimizer
converged	Boolean	Has the optimizer converged
initial_T	float	Initial temperature
current_T	float	Current temperature in the optimization
const_pert_size	Boolean	Whether perturbation size depends on temperature
initial_solution	2xn float array	Initial geometrical parameters
current_solution	2xn float array	Current geometrical parameters
⋮	⋮	⋮

Table B.3: Parameters expected by the simulated annealing class.

Bibliography

[1] B. J. Yokelson and W. Ulrich. "Engineering multistage diode logic circuits". In: *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics* 74.4 (Sept. 1955). Conference Name: Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics, pp. 466–475. ISSN: 2379-674X. DOI: 10.1109/TCE.1955.6372399.

[2] R. H. Bergman. "Tunnel Diode Logic Circuits". In: *IRE Transactions on Electronic Computers* EC-9.4 (Dec. 1960). Conference Name: IRE Transactions on Electronic Computers, pp. 430–438. ISSN: 0367-9950. DOI: 10.1109/TEC.1960.5219881.

[3] T.E. Yingst et al. "High-power gridded tubes—1972". In: *Proceedings of the IEEE* 61.3 (Mar. 1973). Conference Name: Proceedings of the IEEE, pp. 357–381. ISSN: 1558-2256. DOI: 10.1109/PROC.1973.9035.

[4] Keng-Tung Cheng and Niels Olhoff. "An investigation concerning optimal design of solid elastic plates". In: *International Journal of Solids and Structures* 17.3 (1981), pp. 305–323. ISSN: 0020-7683. DOI: https://doi.org/10.1016/0020-7683(81)90065-2. URL: https://www.sciencedirect.com/science/article/pii/0020768381900652.

[5] G. Baccarani, M.R. Wordeman, and R.H. Dennard. "Generalized scaling theory and its application to a $\frac{1}{4}$ micrometer MOSFET design". In: *IEEE Transactions on Electron Devices* 31.4 (1984), pp. 452–462. DOI: 10.1109/T-ED.1984.21550.

[6] Martin Philip Bendsøe and Noboru Kikuchi. "Generating optimal topologies in structural design using a homogenization method". In: *Computer Methods in Applied Mechanics and Engineering* 71.2 (1988), pp. 197–224. ISSN: 0045-7825. DOI: https://

doi.org/10.1016/0045-7825(88)90086-2. URL: https://www.sciencedirect.com/science/article/pii/0045782588900862.

[7] J.-M. Shyu et al. "Optimization-based transistor sizing". In: *IEEE Journal of Solid-State Circuits* 23.2 (Apr. 1988). Conference Name: IEEE Journal of Solid-State Circuits, pp. 400–409. ISSN: 1558-173X. DOI: 10.1109/4.1000.

[8] Shun-ichi Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* 5.4 (1993), pp. 185–196. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(93)90006-O. URL: https://www.sciencedirect.com/science/article/pii/092523129390006O.

[9] Y.Y. Lau, Youfan Liu, and R. K. Parker. "Electron emission: From the Fowler-Nordheim relation to the Child-Langmuir law". In: *American Institute of Physics* (1994).

[10] COMSOL Multiphysics. "Introduction to COMSOL multiphysics®". In: *COMSOL Multiphysics, Burlington, MA, accessed Feb* 9 (1998), p. 2018.

[11] M.M. Spuhler et al. "A very short planar silica spot-size converter using a nonperiodic segmented waveguide". In: *Journal of Lightwave Technology* 16.9 (1998), pp. 1680–1685. DOI: 10.1109/50.712252.

[12] Steven J. Cox and David C. Dobson. "Maximizing Band Gaps in Two-Dimensional Photonic Crystals". In: *SIAM Journal on Applied Mathematics* 59.6 (1999), pp. 2108–2120. ISSN: 00361399. URL: http://www.jstor.org/stable/118418 (visited on 01/17/2023).

[13] V.L. Granatstein, R.K. Parker, and C.M. Armstrong. "Vacuum electronics at the dawn of the twenty-first century". In: *Proceedings of the IEEE* 87.5 (1999), pp. 702–716. DOI: 10.1109/5.757251.

[14] R.K. Parker et al. "Vacuum electronics". In: *IEEE Transactions on Microwave Theory and Techniques* 50.3 (2002), pp. 835–845. DOI: 10.1109/22.989967.

[15] Martin P. Bendsøe and Ole Sigmund. *Topology Optimization - Theory, Methods, and Applications*. English. Germany: Springer Verlag, 2003. ISBN: 3-540-42992-1.

[16] S. Boyd et al. *Convex Optimization*. Berichte über verteilte messysteme pt. 1. Cambridge University Press, 2004. ISBN: 9780521833783. URL: https://books.google.com/books?id=mYm0bLd3fcoC.

[17] P. Ngatchou, A. Zarei, and A. El-Sharkawi. "Pareto Multi Objective Optimization". In: *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. 2005, pp. 84–91. DOI: 10.1109/ISAP.2005.1599245.

[18] W. P. Kang et al. "Nanodiamond Lateral VFEM Technology for Harsh Environments". In: *IEEE Transactions on Nuclear Science* 54.4 (2007), pp. 1061–1065. DOI: 10.1109/TNS.2007.892117.

[19] Chris A. Mack. "Fifty Years of Moore's Law". In: *IEEE Transactions on Semiconductor Manufacturing* 24.2 (2011), pp. 202–207. DOI: 10.1109/TSM.2010.2096437.

[20] Richard G. Forbes. "Development of a simple quantitative test for lack of field emission orthodoxy". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 469.2158 (2013), p. 20130271. DOI: 10.1098/rspa.2013.0271. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.2013.0271. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2013.0271.

[21] Christopher M. Lalau-Keraly et al. "Adjoint shape optimization applied to electromagnetic design". In: *Opt. Express* 21.18 (Sept. 2013), pp. 21693–21701. DOI: 10.1364/OE.21.021693. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-21-18-21693.

[22] Jesse Lu and Jelena Vučković. "Nanophotonic computational design". In: *Optics Express* 21.11 (June 3, 2013). Publisher: Optica Publishing Group, pp. 13351–13367. ISSN: 1094-4087. DOI: 10.1364/OE.21.013351. URL: https://opg.optica.org/oe/abstract.cfm?uri=oe-21-11-13351 (visited on 01/16/2023).

[23] Martin Alnæs et al. "The FEniCS Project Version 1.5". en. In: *Archive of Numerical Software* Vol 3 (2015), Starting Point and Frequency: Year: 2013 . DOI: 10.11588/ANS.2015.100.20553. URL: http://journals.ub.uni-heidelberg.de/index.php/ans/article/view/20553.

[24] A. Kyritsakis and J. P. Xanthakis. "Derivation of a generalized Fowler–Nordheim equation for nanoscopic field-emitters". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2174 (Feb. 8, 2015). Publisher: Royal Society, p. 20140811. DOI: 10.1098/rspa.2014.0811. URL: https://royalsocietypublishing.org/doi/10.1098/rspa.2014.0811 (visited on 11/24/2022).

[25] Alexander Y. Piggott et al. "Inverse design and demonstration of a compact and broadband on-chip wavelength demultiplexer". In: *Nature Photonics* 9.6 (June 2015). Number: 6 Publisher: Nature Publishing Group, pp. 374–377. ISSN: 1749-4893. DOI: 10.1038/nphoton.2015.69. URL: https://www.nature.com/articles/nphoton.2015.69 (visited on 01/16/2023).

[26] Tea Tušar and Bogdan Filipič. "Visualization of Pareto Front Approximations in Evolutionary Multiobjective Optimization: A Critical Review and the Prosection Method". In: *IEEE Transactions on Evolutionary Computation* 19.2 (2015), pp. 225–245. DOI: 10.1109/TEVC.2014.2313407.

[27] S A Guerrera and A I Akinwande. "Nanofabrication of arrays of silicon field emitters with vertical silicon nanowire current limiters and self-aligned gates". In: *Nanotechnology* 27.29 (June 2016), p. 295302. DOI: 10.1088/0957-4484/27/29/295302. URL: https://dx.doi.org/10.1088/0957-4484/27/29/295302.

[28] Jin-Woo Han, Dong-Il Moon, and M. Meyyappan. "Nanoscale Vacuum Channel Transistor". In: *Nano Letters* 17.4 (2017). PMID: 28334531, pp. 2146–2151. DOI: 10.1021/acs.nanolett.6b04363. eprint: https://doi.org/10.1021/acs.nanolett.6b04363. URL: https://doi.org/10.1021/acs.nanolett.6b04363.

[29] Hans Petter Langtangen and Anders Logg. *Solving PDEs in Python*. Springer, 2017. ISBN: 978-3-319-52461-0. DOI: 10.1007/978-3-319-52462-7.

[30] Alexander Y. Piggott et al. "Fabrication-constrained nanophotonic inverse design". In: *Scientific Reports* 7.1 (May 11, 2017). Number: 1 Publisher: Nature Publishing Group, p. 1786. ISSN: 2045-2322. DOI: 10.1038/s41598-017-01939-2. URL: https://www.nature.com/articles/s41598-017-01939-2 (visited on 01/16/2023).

[31] Hsien-Ching Lo et al. "Transistor Optimization with Novel Cavity for Advanced FinFET Technology". In: *2018 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)*. 2018 International Conference on Simulation of Semiconductor Processes and Devices (SISPAD). ISSN: 1946-1577. Sept. 2018, pp. 210–213. DOI: 10.1109/SISPAD.2018.8551703.

[32] Sean Molesky et al. "Inverse design in nanophotonics". In: *Nature Photonics* 12.11 (Oct. 2018), pp. 659–670. DOI: 10.1038/s41566-018-0246-9. URL: https://doi.org/10.1038/s41566-018-0246-9.

[33] Shruti Nirantar et al. "Metal–Air Transistors: Semiconductor-Free Field-Emission Air-Channel Nanoelectronics". In: *Nano Letters* 18.12 (Dec. 12, 2018), pp. 7478–7484. ISSN: 1530-6984, 1530-6992. DOI: 10.1021/acs.nanolett.8b02849. URL: https://pubs.acs.org/doi/10.1021/acs.nanolett.8b02849 (visited on 01/05/2023).

[34] Thomas M. Antonsen, David Chernin, and John J. Petillo. "Adjoint approach to beam optics sensitivity based on Hamiltonian particle dynamics". In: *Physics of Plasmas* 26.1 (Jan. 2019), p. 013109. DOI: 10.1063/1.5079629. URL: https://doi.org/10.1063/1.5079629.

[35] Andrew M. Bradley. *PDE-constrained optimization and the adjoint methods*. Oct. 2019.

[36] Seyedali Mirjalili. "Genetic Algorithm". In: *Evolutionary Algorithms and Neural Networks: Theory and Applications*. Cham: Springer International Publishing, 2019, pp. 43–55. ISBN: 978-3-319-93025-1. DOI: 10.1007/978-3-319-93025-1_4. URL: https://doi.org/10.1007/978-3-319-93025-1_4.

[37] Lars Thorben Neustock et al. "Inverse Design Tool for Ion Optical Devices using the Adjoint Variable Method". In: *Scientific Reports* 9.1 (July 2019). DOI: 10.1038/s41598-019-47408-w. URL: https://doi.org/10.1038/s41598-019-47408-w.

[38] Lucia B. De Rose, Axel Scherer, and William M. Jones. "Suspended Nanoscale Field Emitter Devices for High-Temperature Operation". In: *IEEE Transactions on Electron Devices* 67.11 (Nov. 2020), pp. 5125–5131. ISSN: 0018-9383, 1557-9646. DOI: 10.1109/TED.2020.3019765. URL: https://ieeexplore.ieee.org/document/9189941/ (visited on 01/05/2023).

[39] Georg Gaertner, Wolfram Knapp, and Richard G. Forbes, eds. *Modern Developments in Vacuum Electron Sources*. Vol. 135. Topics in Applied Physics. Cham: Springer International Publishing, 2020. ISBN: 978-3-030-47290-0 978-3-030-47291-7. DOI: 10.1007/978-3-030-47291-7. URL: http://link.springer.com/10.1007/978-3-030-47291-7 (visited on 11/28/2022).

[40] Sunae So et al. "Deep learning enabled inverse design in nanophotonics". In: *Nanophotonics* 9.5 (May 1, 2020). Publisher: De Gruyter, pp. 1041–1057. ISSN: 2192-8614. DOI: 10.1515/nanoph-2019-0474. URL: https://www.degruyter.com/document/doi/10.1515/nanoph-2019-0474/html (visited on 01/16/2023).

[41] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[42] Ranajoy Bhattacharya et al. "Long term field emission current stability characterization of planar field emitter devices". In: *Journal of Vacuum Science &amp Technology B* 39.5 (Sept. 2021), p. 053201. DOI: 10.1116/6.0001182. URL: https://doi.org/10.1116/6.0001182.

[43] Mina R. Bionta et al. "On-chip sampling of optical fields with attosecond resolution". In: *Nature Photonics* 15.6 (Apr. 2021), pp. 456–460. DOI: 10.1038/s41566-021-00792-0. URL: https://doi.org/10.1038/s41566-021-00792-0.

[44] Drew Buckley et al. "Nanoantenna design for enhanced carrier–envelope-phase sensitivity". In: *J. Opt. Soc. Am. B* 38.9 (Sept. 2021), pp. C11–C21. DOI: 10.1364/JOSAB.424549. URL: https://opg.optica.org/josab/abstract.cfm?URI=josab-38-9-C11.

[45] A Nardi et al. "Nanoscale refractory doped titanium nitride field emitters". eng. In: *Nanotechnology* 32.31 (2021), pp. 315208–. ISSN: 0957-4484.

[46] Hugo Serra, Rui Santos-Tavares, and Nuno Paulino. "Transistor-level optimization methodology for the complete design of switched-capacitor filter circuits". In: *International Journal of Circuit Theory and Applications* 49.1 (2021). _eprint: https://onlinelibrary.wiley.com

pp. 94–113. ISSN: 1097-007X. DOI: 10.1002/cta.2891. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2891 (visited on 01/10/2023).

[47] Peter R. Wiecha et al. "Deep learning in nano-photonics: inverse design and beyond". In: *Photonics Research* 9.5 (May 1, 2021), B182. ISSN: 2327-9125. DOI: 10.1364/PRJ.415960. URL: https://opg.optica.org/abstract.cfm?URI=prj-9-5-B182 (visited on 01/16/2023).

[48] Cherq Chua, Yee Sin Ang, and Lay Kee Ang. "Tunneling injection to trap-limited space-charge conduction for metal-insulator junction". In: *Applied Physics Letters* 121.19 (Nov. 7, 2022), p. 192109. ISSN: 0003-6951, 1077-3118. DOI: 10.1063/5.0124748. URL: https://aip.scitation.org/doi/10.1063/5.0124748 (visited on 11/23/2022).

[49] Francesco Mezzadri and Xiaoping Qian. "Density gradient-based adaptive refinement of analysis mesh for efficient multiresolution topology optimization". In: *International Journal for Numerical Methods in Engineering* 123.2 (2022). _eprint: https://onlinelibrary.wiley.com/
pp. 465–504. ISSN: 1097-0207. DOI: 10.1002/nme.6863. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6863 (visited on 12/05/2022).

[50] Marco Turchetti et al. "Electron Emission Regimes of Planar Nano Vacuum Emitters". In: *IEEE Transactions on Electron Devices* 69.7 (July 2022). Conference Name: IEEE Transactions on Electron Devices, pp. 3953–3959. ISSN: 1557-9646. DOI: 10.1109/TED.2022.3175706.

[51] A Demenko, J K Sykulski, and R M Wojciechowski. "2D versus 3D electromagnetic field modelling in electromechanical energy converters". In: (), p. 2.

[52] *Electric field enhancement equations*. URL: http://www.nessengr.com/technical-data/electric-field-enhancement/.

[53] *Lorentz: Integrated Engineering Software*. URL: https://www.integratedsoft.com/products/Lorentz.

[54] *MEEP*. URL: https://meep.readthedocs.io/ (visited on 01/12/2023).

[55] Rob A. Rutenbar. "Simulated Annealing Algorithms: An Overview". In: *IEEE Circuits and Devices Magazine* ().

[56] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects*. toblerity.org, 2007–. URL: `https://github.com/Toblerity/Shapely`.