With the support of the Erasmus+ Programme of the European Union

INTERMATHS
ERASMUS MUNDUS JOINT MASTER DEGREE

UAB Universitat Autònoma de Barcelona

UNIVERSITÀ DEGLI STUDI DE L'AQUILA

TU WIEN TECHNISCHE UNIVERSITÄT WIEN

Erasmus Mundus Joint Master Degree "InterMaths" Interdisciplinary Mathematics

Specialisation: Computational Fluid Dynamics in Industry (TUW, Vienna)

Máster Universitario Erasmus Mundus en "Matemáticas Interdisciplinarias" UNIVERSITAT AUTÒNOMA DE BARCELONA (UAB)	*Laurea Magistrale nella classe LM-44 "Mathematical Modelling"* UNIVERSITÀ DEGLI STUDI DELL'AQUILA (UAQ)	*Master in "Interdisciplinary Mathematics"* TECHNISCHE UNIVERSITÄT WIEN (TUW)

Master's Thesis Title

Automated FEM model generation for the simulation of microheaters.

Supervisor
Associate Prof. Dr.techn. Lado Filipovic

[redacted]

Candidate
Elio Skënderaj

[redacted]

Co-advisor
Dr. Daniel Tscharnuter

[illegible]

Student ID (UAQ): 280214

Academic Year 2022/2023

Automated FEM model generation for the simulation of microheaters

MASTERARBEIT

zur Erlangung des akademischen Grades

Master of Science

im Rahmen des Studiums

Interdisciplinary Mathematics

eingereicht von

Elio Skënderaj
Matrikelnummer 12143578

an der Fakultät für Mathematik und Geoinformation

der Technischen Universität Wien

Betreuung: Associate Prof. Dr.techn. Lado Filipovic, Fakultät für Elektrotechnik und Informationstechnik
Mitwirkung: Dr. Daniel Tscharnuter, KAI Kompetenzzentrum Automobil- und Industrieelektronik GmbH

Wien, 24. Oktober 2023 [illegible] 19.10.2023 [redacted]

Elio Skënderaj Lado Filipovic

Automated FEM model generation for the simulation of microheaters

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Interdisciplinary Mathematics

by

Elio Skënderaj
Registration Number 12143578

to the Faculty of Mathematics and Geoinformation

at the TU Wien

Advisor: Associate Prof. Dr.techn. Lado Filipovic, Fakultät für Elektrotechnik und Informationstechnik
Assistance: Dr. Daniel Tscharnuter, KAI Kompetenzzentrum Automobil- und Industrieelektronik GmbH

Vienna, 24th October, 2023	[redacted]	19.10.2023 [redacted]
	Elio Skënderaj	Lado Filipovic

Declaration of Authorship

Elio Skënderaj

I hereby declare that I have written this Master Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 24th October, 2023

[redacted]

Elio Skënderaj

Acknowledgements

Firstly, I would like to express my gratitude to both of my supervisors Dr.Daniel Tscharnuter and Prof.Lado Filipovic for accepting me in this project and continuously supporting me throughout the development phase. Your advice and feedback was essential in the completion of this work. I would also like to thank Dr. Sebastian Moser for the extensive explanations on the experimental aspect. Many thanks go also to all of my KAI colleagues for creating a friendly and collaborative working environment.

Secondly, I would like to thank the InterMaths EMJMD Consortium for allowing me to be part of this diverse and challenging study program. The semesters spent at University of L'Aquila and TU Wien were truly formative and enriching academically and personally.

Lastly, I am particularly grateful to my family and close friends, for their constant support throughout my studies.

Villach, October 2023

Multicharm

Year 2: 1.5.2022-30.4.2023 : This work was funded by the Austrian Research Promotion Agency (FFG, Project No. 898207).
Year 3: 1.5.2023-30.4.2024 : This work was funded by the Austrian Research Promotion Agency (FFG, Project No. 905107).

Abstract

The rapid growth of the semiconductor industry has seen an increase in demand for reliability tests which are essential for ensuring that the devices reach their expected lifetimes, and the overall well functioning of the devices is achieved. Experimental tests give plenty of insight into topics of interest, but nowadays with access to high performance computing facilities, it is possible to perform multiphysics simulations which are more cost efficient compared to their physical testing counterparts. However, there is a significant effort placed into developing and then refining the simulation models, so that they mimic reality up to a desired accuracy.

In multiphysics simulations there are typically three well-known routines involved: pre-processing, solving, and post-processing. The aim of this work is to provide an automated workflow which goes through the pre-processing routine which, in most engineering/-physics applications, is the most tedious and time consuming component.

The pre-processing consists of the following tasks: Defining the geometry, meshing, defining the material properties, and specifying the initial and the boundary conditions. In most cases the latter two are derived from experiments or literature values, but the first two tasks are always problem-specific and can be of considerable difficulty. There are general good practices that one should follow, but in real engineering problems it is up to the simulation engineer to make compromises between modelling assumptions, meshing and the desired simulation accuracy compared to the experimental data.

In this work, the finite element method is applied to analyse the temperature response in non-commercial microelectronic devices, which typically require many repetitive manual tasks to be executed during pre-processing. The design of these devices is first drawn in the graphic design system (GDS) format, which is an industry standard for microelectronic devices. In the scope of this thesis, a Python framework is developed to automate all the pre-processing parts of these designs and minor automation is also introduced to solving and the post-processing. For demonstration purposes the developed methods are applied to generate and validate models of test chips against experimental data. The model generation is fully automated, and is up to 50 times faster than the previous manual routine. Adjustments in the geometry pre-processing enhance the solving runtime, by making it up to 35 times faster than the previous solving methodology. The Python framework is scaled to work on a family of different device models and architectures which share certain similarities.

Contents

CHAPTER 1

Introduction and motivation

Microelectronic devices are small electronic components which are typically made using semiconductor materials. These devices typically have structural components which are just a few micrometers in size, and are designed to perform specific electrical functions. Microelectronic devices are used in a wide range of applications, from consumer electronics such as smartphones, computers, and digital cameras to medical devices, electrical vehicles, industrial equipment, etc.

The structural design of a microelectronic device typically involves the integration of several layers of structural thin films, each with a specific function, on a substrate material. The substrate material is usually made of a semiconductor, such as silicon, and serves as the base layer for the device. The layers of thin films are deposited and etched to a desired pattern on the substrate using a variety of processing techniques. These layers can include metal interconnects, dielectric layers for insulation, and active regions containing semiconductors doped with impurities to create specific electrical properties.

The active regions of the microelectronic device typically contain p-n junctions, which are formed by the intentional introduction of impurities to the semiconductor material. The p-n junction acts as a one-way valve for electrical current, allowing it to flow from the p-doped region to the n-doped region, with a small reverse bias on the counter direction. This property is essential for enabling logic and switching operations in the device.

1.1 Polyheater test devices

Purpose

Power electronic devices are designed to handle higher power levels compared to other microelectronic devices. During their active operation they often experience thermal cycling, which can be categorized into active power cycling and thermal overload/short circuit pulsing. Active power cycling refers to the scenario where the device is subjected to repeated ON and OFF cycles, while short circuit pulsing can occcur as a consequence of an electrical fault. Both types of thermal cycling can cause the device to heat up and cool down repeatedly, which creates thermo-mechanical stress and may lead to degradation and eventually, failure of the device. Stress testing engineers use thermal cycling tests to evaluate how well a device can withstand these thermal stresses and to predict its lifespan and durability under harsh conditions.

To study the thermo-mechanical fatigue of metallizations in power semiconductors, stress tests are conducted on commercially available devices. This approach replicates real-world conditions with identical chip layouts and materials while emulating environmental and electrical conditions in the lab. Detectable failures, such as local melting or plastic deformations, enable reliability assessment and identification of vulnerable chip areas.

The major issue of this approach is that most of commercial devices of interest are designed to work efficiently, not to help assess the heat and stress they experience during use. This means that it is not easy to change certain factors like how long a heating pulse lasts because the device's architecture does not allow it. Moreover commercial devices are packaged into a protective enclosure to withstand the environmental threats (physical damage, dust, and moisture). These packages are designed in a way to ensure the right electrical contact and thermal management; however, the ambient-safe packaging restricts the ability of real-time monitoring of stress-testing scenarios.

To bypass the difficulties mentioned above polysilicon microheaters, also known as polyheaters, were introduced in [1]. These are simplified test chips which can be subjected to a precise and controlled amount of heat loading in order to analyse the thermo-mechanical fatigue of the materials which are present in commercial semiconductor devices.

In order to understand the thermo-mechanical fatigue of the materials, we only need to look at a bi-layer system, as shown on Figure 1.1. This system is composed of a thin copper layer on top of a much thicker silicon substrate. Each material has a different coefficient of thermal expansion (CTE), with CTE_{Si} being $2.56 \cdot 10^{-6} K^{-1}$ and CTE_{Cu} being $17 \cdot 10^{-6} K^{-1}$, so almost 7 times higher when compared to silicon. As the temperature changes, the resulting mechanical stress occurs due to the difference in CTEs. Specifically, during the heat-up and cool-down cycles, the copper layer is subjected to compression and tension respectively. Repeated stress cycles can cause damage to the metallization layer and lead to plastic deformation in the copper surface. Motivated by this observation in a simple example, the same concepts are implemented in the polyheater designs through a more complex layered structure.

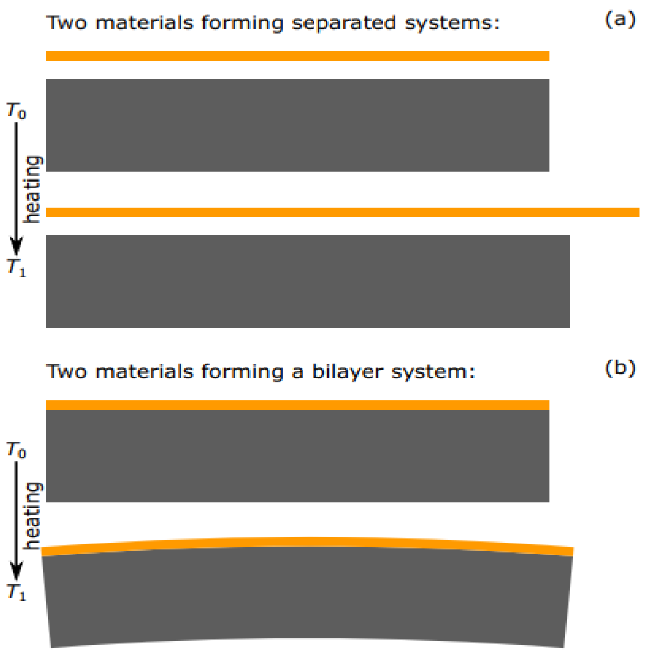

Two materials forming separated systems:
(a)
T_0
heating
T_1
Two materials forming a bilayer system:
(b)
T_0
heating
T_1

Figure 1.1: (a) The thick silicon substrate film in gray and the thinner copper film in orange are considerd as separate systems that can expand individually.
(b)Both Si and Cu are considered as a bilayer with adhesion boundary conditions. Warping with a radius of curvature can be observed. Used with permission from [1].

Layer design

The polyheaters are devices that provide a high temperature at a specific location as a result of Joule heating. These devices vary a lot in their design and allow for different types of research to be done on them; however, they share some structural properties. Each model has lateral dimensions of 3.3 mm × 1.9 mm. The bottom layer of the chip is a 120µm-thick silicon substrate. A thin layer of oxide is put on top of the substrate for electrical insulation. Subsequently layers of polysilicon, aluminium and Cu with insulating nanometer-thin layers inbetween are deposited. In areas where there has to be electrical contact to the polysilicon layer there are small metal interconnecting channels known as vias. The polysilicon layer acts as a resistive heater to the entire device and can be subjected to sub-millisecond pulse durations of Joule heating. In the aluminium layer, referred as M1 in the layout, various types of sensors are included, which range in design depending on the model type and the type of test they are used to extract measurement data for, as shown on Figure 1.2. For our investigation the relevant application of these layers are as temperature sensors in a meander shape localized in the aluminium(M1) layer as depicted in Figure 1.3 in cross-sectional and top view.

Setup and calibration

The electrical actuation of the chip is realized through the copper pads located on the metallisation layer, as explained in [2]. Temperature sensing devices located on the M1 layer have Ohmic behaviour, meaning that resistance is proportional to the temperature. This behaviour is represented by the temperature coefficient of resistance $\alpha_{TCR,T_{ref}}$ for a specific material at a given reference temperature T_{ref}. Using these parameters, a linear model in [2], [3] is defined as:

$$R(T) = R_{T_{ref}} \cdot (1 + \alpha_{TCR,T_{ref}} \cdot (T - T_{ref})) \tag{1.1}$$

In order to calculate resistance in this way both $\alpha_{TCR,T_{ref}}$ and $R_{T_{ref}}$ have to be known for the specified temperature T_{ref}. Most of these values can be found in literature for standard reference temperatures and established materials. For any other T_{ref} of interest, both values of $\alpha_{TCR,T_{ref}}$ and $R_{T_{ref}}$ can be calculated using experiments and line fitting models.

The calibration procedure for the temperature is as follows: First the $\alpha_{TCR,T_{ref}}$ of interest must be chosen, then $R_{T_{ref}}$ is calculated using Eq.(1.1) where $R(T) = R_{T_{room}}$. Inserting the calculated $R_{T_{ref}}$ results in Eq.(1.1) :

$$R(T) = R_{T_{room}} \cdot \frac{1 + \alpha_{TCR,T_{ref}} \cdot (T - T_{ref})}{1 + \alpha_{TCR,T_{ref}} \cdot (T_{room} - T_{ref})} \tag{1.2}$$

1.2 Modelling and simulation of polyheater devices.

With the rapid development of miniaturization techniques, there are new microelectronic devices being fabricated with smaller features and this means that power densities

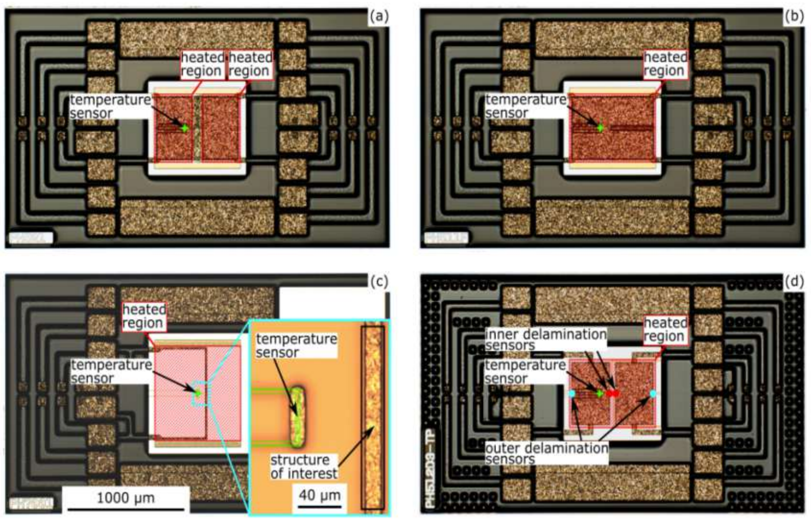


Figure 1.2: Light microscopy images of 4 different polyheater models each having different heating configurations on the polysilicon layer. Different configurations of temperature sensors are depicted in the specified areas of the 'M1' layer. Image taken with permission from [1].

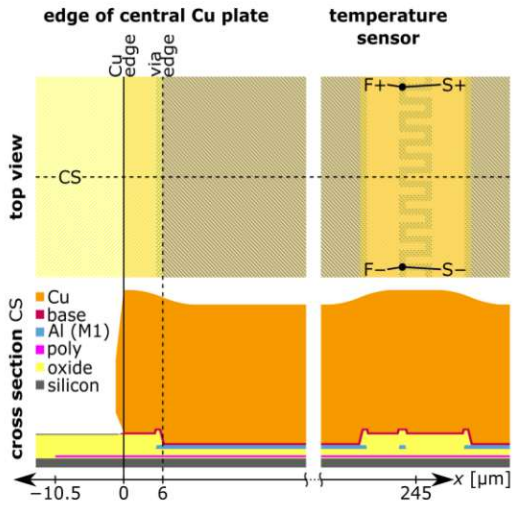


Figure 1.3: Top view and their cross-sectional counterparts of a polyheater device. The image taken with permission from [1], is provided just for schematic reasons and does not actually refer to any model studied here.

inside a functioning device are constantly increasing. There is a plethora of research into optimisation methods of heat dissipation in microelectronic devices, see [4], [5], [6]. The accumulation of thermal stresses in very short time intervals has several negative consequences such as: chip bowing, delamination, thermo-mechanical fatigue of metal layers, etc. All these phenomena impact the lifetime of the device and could potentially lead to its ultimate failure.

In order to investigate the aforementioned phenomena on the polyheater test devices many experimental investigations have been carried out in the past. However all the data extracted from an experiment is limited to the extent of sensing devices which are present on the device under test. For example, in the context of experimental thermal analysis, it is possible to obtain the temperature values only from the region where temperature sensors are present, as depicted in Figure 1.2. If the temperature values of the Cu metallization layer need to be known this information cannot be measured through experiments.

When performing a FEM simulation of the thermal response of a polyheater test chip in the post-processing of the simulation data, the temperature profiles can be extracted and visualized in any region of the chip, providing a better understanding about how heat is distributed through the different material layers described in Figure 1.3.

CHAPTER 2

Background

2.1 Heat conduction in solids

Heat flows from the hotter parts of a medium to the colder parts in three different ways: conduction, which it is realised through physical contact; convection-advection, where the transport property and relative motion of the medium play a major role; and radiation, in which heat is transferred through electromagnetic radiation. In solids, heat transfer takes place mainly through conduction. Convection-advection is not present and heat transfer through radiation is negligible. A quick overview of the theory in isotropic materials will be given below and building from that some results about anisotropic heat transfer will be provided.

Isotropic materials

Mathematical modelling of heat transfer in solids is done considering their physical properties along different directions. Usually the simplest mathematical models are derived from first principles taking some ideal assumptions into account. Assume a homogeneous, isotropic solid body with constant mass density ρ. Isotropic means that the material properties of the body do not depend on the direction they are measured.

In the model, denote the interior of the body by Ω and by $u(x, y, z, t)$ the temperature of the body at a point (x, y, z) at the time t. Furthermore, we assume that $u(x, y, z, t)$ is C^2 (second derivative is continuous) with respect to the spatial variables (x, y, z) and C^1 (first derivative is continuous) with respect to the time variable t.

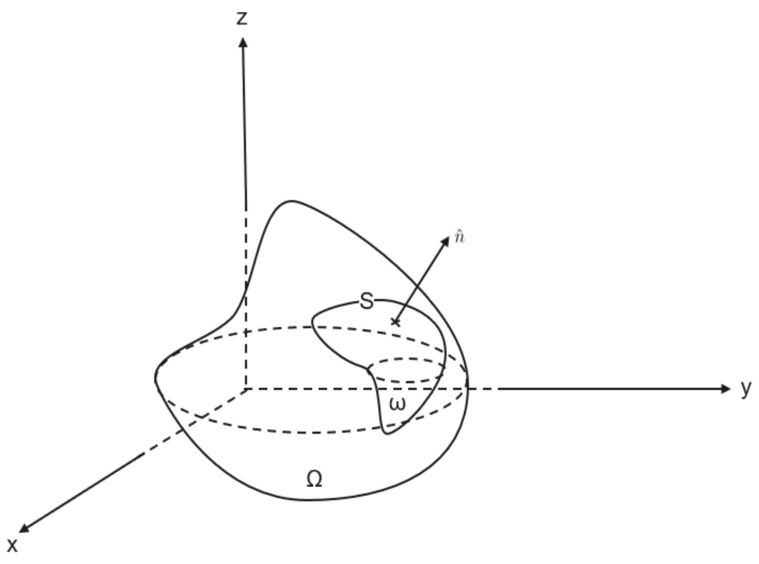


Figure 2.1: Solid domain definition.

Define S to be a smooth surface inside Ω as seen on 2.1 and let $\hat{n}$ be be a unit normal vector on S. In a defined time interval $[t_1, t_2]$, the amount of heat q which passes through S on the side identified by direction $\hat{n}$, assuming there are no external heat sources or sinks, is given by [7]:

$$q = -\int_{t_1}^{t_2} \iint_S k(x, y, z) \frac{\partial u}{\partial \hat{n}} \, d\sigma \, dt \tag{2.1}$$

In Eq.(2.1), $k(x, y, z)$ is a positive function which represents the thermal conductivity of the body at a point (x, y, z) and, since we assumed an isotropic material, the value of k does not depend on the direction of the normal $\hat{n}$ to the surface S at the point (x, y, z).

Consider a smaller subregion ω inside the domain Ω bounded by a smooth closed surface with a unit normal vector $\hat{n}$ pointing outwards. The change in the amount of heat inside the subregion A from time t_1 to time t_2 is given by:

$$\iiint_\omega c(x, y, z)\rho(x, y, z)[u(x, y, z, t_{2)} - u(x, y, z, t_{1)}] \, dx \, dy \, dz \tag{2.2}$$

In the equation above $c(x, y, z)$ is the specific heat and $\rho(x, y, z)$ is the density of the body at the point (x, y, z). From the law of conservation of thermal energy, this change of heat in ω must be equal to the amount of heat which enters into A through its boundary S in the time interval $[t_1, t_2]$ and which given by:

$$q = \int_{t_1}^{t_2} \iint_S k(x,y,z) \frac{\partial u}{\partial \hat{n}} \, d\sigma \, dt \tag{2.3}$$

By equalling Eq.(2.2) and Eq.(2.3) we obtain:

$$\iiint_\omega c(x,y,z)\rho(x,y,z)[u(x,y,z,t_{2)} - u(x,y,z,t_1)] \, dx \, dy \, dz = \int_{t_1}^{t_2} \iint_S k(x,y,z) \frac{\partial u}{\partial \hat{n}} \, d\sigma \, dt \tag{2.4}$$

By knowing that:

$$u(x,y,z,t_{2)} - u(x,y,z,t_1) = \int_{t_1}^{t_2} \frac{\partial u}{\partial t}(x,y,z,t) dt \tag{2.5}$$

and, $\partial u / \partial n = \nabla u \cdot n$, the divergence theorem applied to the vector field $V = k\nabla u$ gives:

$$\iint_S k \frac{\partial u}{\partial \hat{n}} \, d\sigma = \iiint_\omega \nabla \cdot (k\nabla u) \, dx \, dy \, dz \tag{2.6}$$

In this way equation Eq.(2.4) becomes:

$$\int_{t_1}^{t_2} \iiint_\omega [c\rho \frac{\partial u}{\partial t}] \, dx \, dy \, dz \, dt = \int_{t_1}^{t_2} \iiint_\omega \nabla \cdot (k\nabla u) \, dx \, dy \, dz \, dt \tag{2.7}$$

or

$$\int_{t_1}^{t_2} \iiint_\omega [c\rho \frac{\partial u}{\partial t} - \nabla \cdot (k\nabla u)] \, dx \, dy \, dz \, dt = 0 \tag{2.8}$$

By assumption of the regularity, the integrand on 2.8 is continuous and since this is valid for all subregions ω and all intervals $[t_1, t_2]$, it follows that the integrand must be zero for all (x, y, z) in Ω and for all t. Therefore we have:

$$c\rho \frac{\partial u}{\partial t} - \nabla \cdot (k\nabla u) = 0 \tag{2.9}$$

The partial differential equation above describes heat conduction in an isotropic body that has no external heat sources/sinks. In the case that there are heat sources/sinks, in the right-hand side of Eq. 2.9 there is an additional forcing term $g(x, y, z, t)$ that

is a heat source if it has a positive sign, or a heat sink otherwise. For isotropic and homogenous bodies ρ, k, and c are in linear dependence of temperature and are all treated as constants.
In order for Eq.(2.9) to have an unique solution, the initial and boundary conditions must be supplied from physical considerations. Specifying the temperature distribution in the closure of the body domain (set of the body domain, with its limit points added) at a given time t_0 is known as an initial condition:

$$u(x, y, z, t_0) = \phi(x, y, z), \qquad (x, y, z) \in \overline{\Omega} \tag{2.10}$$

Depending on the type of heating, boundary conditions need to be specified as well:

a) Prescribed surface temperature

The temperature over the boundary can be constant, or a function of time, or position, or both, defined in Eq.(2.11). This type of boundary condition, also known as Dirichlet boundary coundition, is used mostly in idealized scenarios and rarely in real engineering problems.

$$u(x, y, z, t) = f(x, y, z, t), \qquad (x, y, z) \in \partial\Omega, \qquad t > t_0 \tag{2.11}$$

b) Adiabatic boundary condition

The adiabatic boundary condition refers to the case when there is no heat exchange accross the boundary of the system i.e. there is no heat flux. In practical terms, adiabatic boundaries are often encountered in situations where the boundary is well-insulated or thermally isolated, preventing heat from flowing in or out of the system through that boundary. It is expressed as:

$$\frac{\partial u}{\partial \hat{n}} = 0, \qquad (x, y, z) \in \partial\Omega, \qquad t > t_0 \tag{2.12}$$

where $\frac{\partial}{\partial \hat{n}}$ denotes the directional derivative in the direction of the outward normal to the boundary surface.

c)Prescribed flux along the surface

The prescribed heat flux boundary condition in the heat equation can be defined mathematically as:

$$q = -k\frac{\partial u}{\partial \hat{n}} \tag{2.13}$$

where q is the heat flux and k is the thermal conductivity of the material, . The negative sign indicates that heat flows from high temperature to low temperature. This boundary condition is often used when the temperature at the boundary is not known or is not important for the problem being solved. It specifies the rate at which heat is flowing through the boundary by defining the amount of heat per unit area per unit time.

d) Radiation boundary condition

Radiation, or linear heat transfer, at the surface is imposed if the flux across the surface is proportional to the temperature difference between the boundary and the surrounding environment, which is given by:

$$H(u - u_0) \tag{2.14}$$

where u_0 is the temperature of the medium and H and K are both constants. The radiation boundary condition is then defined as:

$$K\frac{\partial u}{\partial \hat{n}} + H(u - u_0) = 0, \qquad \text{or} \qquad \frac{\partial u}{\partial \hat{n}} + h(u - u_0) = 0 \tag{2.15}$$

where $h = H/K$.

As $h \to 0$, the boundary condition (2.15) tends to the adiabatic boundary condition and as $h \to \infty$ it tends to the prescribed surface temperature boundary condition.

In general, the radiation boundary condition is used when there is transfer of heat by electromagnetic waves between two bodies which are at different temperatures and separated by a distance. It describes the rate at which energy is radiated away from a surface, and is based on the Stefan-Boltzmann law,[8], which states that the rate of energy radiated per unit area is proportional to the fourth power of the absolute temperature:

$$q = \epsilon\sigma(u^4 - u_\infty^4) \tag{2.16}$$

Here, q is the heat flux, ϵ is the emissivity of the surface, σ is the Stefan-Boltzmann constant, u is the temperature of the surface, and u_∞ is the temperature of the surrounding environment.

The radiation boundary condition is particularly important in applications involving high-temperature processes, such as combustion, where radiation is a major mode of heat transfer. It is also used in problems involving the transfer of heat in space or in vacuum, where there is no direct contact between the two bodies.

e) Boundary condition over the surface of separation of two media with different conductivities

Denote by u_1 and u_2 the temperatures of each medium and k_1, k_2 their respective thermal conductivities. According to [9] the flux is continous over the surface of separation between the media:

$$k_1 \frac{\partial u_1}{\partial \hat{n}} = k_2 \frac{\partial u_2}{\partial \hat{n}} \tag{2.17}$$

where $\partial/\partial n$ is the directional derivative along the normal to the surface of separation.

In the case that the media are adhesive to each other, e.g., they are soldered along their surface of separation, in addition to boundary condition (2.17) we have:

$$u_1 = u_2 \tag{2.18}$$

For other cases, when the contact is not so close, the boundary condition through the surface of separation is defined as:

$$-k_1 \frac{\partial u_1}{\partial \hat{n}} = H(u_1 - u_2) \tag{2.19}$$

Anisotropic materials

Anisotropic materials are materials which exhibit different physical or mechanical properties in different axial directions. In anisotropic materials the components of the heat flux q depend on a linear combination of temperature gradients along the spatial directions in the following way:

$$-q_x = k_{11}\frac{\partial u}{\partial x} + k_{12}\frac{\partial u}{\partial y} + k_{13}\frac{\partial u}{\partial z} \tag{2.20}$$

$$-q_y = k_{21}\frac{\partial u}{\partial x} + k_{22}\frac{\partial u}{\partial y} + k_{23}\frac{\partial u}{\partial z} \tag{2.21}$$

$$-q_z = k_{31}\frac{\partial u}{\partial x} + k_{32}\frac{\partial u}{\partial y} + k_{33}\frac{\partial u}{\partial z} \tag{2.22}$$

or in a more compact way:

$$q_i = -\sum_{j=1}^{3} k_{ij}\frac{\partial u}{\partial x} \qquad i = 1, 2, 3 \tag{2.23}$$

Thus, in anisotropic materials, the heat flux vector q is not always normal to the isothermal surfaces passing through a specific point. The thermal conductivity k of an anisotropic body has 9 components k_{ij}, called conductivity coeffiecients which are the entries of a second order tensor $\mathbf{k}$:

$$\mathbf{k} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \tag{2.24}$$

and in the anisotropic case, Eq.(2.9) can be written for a rectangular coordinate system as in [10]:

$$k_{11}\frac{\partial^2 u}{\partial x^2} + k_{22}\frac{\partial^2 u}{\partial y^2} + k_{33}\frac{\partial^2 u}{\partial z^2} + (k_{12}+k_{21})\frac{\partial^2 u}{\partial x \partial y} + (k_{13}+k_{31})\frac{\partial^2 u}{\partial y \partial z} + \ldots$$
$$+ (k_{23}+k_{32})\frac{\partial^2 u}{\partial y \partial z} + g(x,y,z) = \rho c \frac{\partial u}{\partial t} \tag{2.25}$$

where $g(x,y,z)$ denotes an external force term. In a real engineering problem Eq.(2.25) is subjected to initial and boundary conditions similarly to the Eq.(2.9). Various simplifications can be performed by considering structural symmetry especially in crystals. This reduces the number of unknowns in the thermal conductivity tensor $\mathbf{k}$ and simplifies further Eq.(2.25) as shown in [9], [10]. However, in most cases the conductivity coefficients k_{ij} depend on the temperature itself, making the problem nonlinear and very difficult to solve.

2.2 Numerical methods

In many engineering applications, there is a nonlinear problem defined over a highly complex geometrical domain, where a closed form solution cannot be found. In this regard, numerical methods are used to find approximate solutions to these problems. The most widely used method in solid mechanics is the the finite element method, mostly due to the fact that it is quite robust and it can handle very complicated geometries.

Semi-discretization of the heat equation

In order to solve Eq.(2.9) numerically we will use a FEM solver based on the Galerkin method. Since we need to study the time evolution of the temperature we will perform a semi-discretization in the spatial variables. By choosing accordingly a finite element subspace $V_h \subseteq V$ and writing the steady-time variant of Eq.(2.9) with Dirichlet initial conditions in its weak form we will solve the following:
Find $u : [0,T] \to V_h$ such that:

$$(u_h'(t), v_h)_{L^2(\Omega)} + a(u_h(t), v_h) = (f(t), v_h)_{L^2}(\Omega) \qquad \forall v_h \in V_h, \forall t \in (0,T] \tag{2.26}$$

with initial conditions:

$$(u_h(0), v_h)_{L^2(\Omega)} = (u_0, v_h)_{L^2(\Omega)} \qquad \forall v_h \in V_h \tag{2.27}$$

By choosing a basis $\{\phi_1, ..., \phi_N\}$ of V_h and expanding the solution with respect to this basis:

$$u_h(x,t) = \sum_{i=1}^{N} u_i(t)\phi_i(x) \tag{2.28}$$

In the weak formulation we use the basis functions as test functions: $v = \phi_j$ which leads to the mass matrix M and stiffness matrix A of the form:

$$M = ((\phi_j, \phi_i)_{L^2(\Omega)})_{i,j=1,...,N} \qquad A = (a(\phi_j, \phi_i))_{i,j=1,...,N} \tag{2.29}$$

and the load vector depends on time:

$$f\hat{(t)} = ((f(t), \phi_j)_{L^2(\Omega)})_{i,j=1,...,N} \tag{2.30}$$

The problem will be reduced to a system of time-dependent ordinary differential equations:

$$Mu'(t) + Au(t) = f\hat{(t)}, \qquad u(0) = u_0 \tag{2.31}$$

which are solvable by implicit time-stepping methods such as: Newmark-beta method, generalized alpha method etc.

Meshing and different element types

The Galerkin method described above can be applied to a variety of strongly formulated problems on any arbitrarily complicated geometry of interest. Before assembling the system of time-dependent ODEs described in Eq.(2.31) a discretisation of the geometry must be made using finite elements. The basic definition as given in [11] is:

Definition: *A finite element is a triple (T, V_T, Ψ_T), where:*

1. *T is a bounded set.*
2. *V_T is a finite dimensional function space on T of dimension $N_T \in \mathbb{N}$.*
3. *$\Psi_T = (\psi_T^1, ..., \psi_T^{N_T})$ is a set of linearly independent functionals on V_T.*

When V_T is a space of linear polynomials, the most common elements are shown in Figure 2.2. The set of linearly independent functionals can be point evaluation functionals over the basis functions and these give the Lagrange elements or point evaluation functionals over the first derivative of the basis functions; these are called Hermite elements. However, the point evaluation can be in the nodes of the set T, but also in the middle of its edges or on other positions, giving rise to a diverse class of finite element methods.

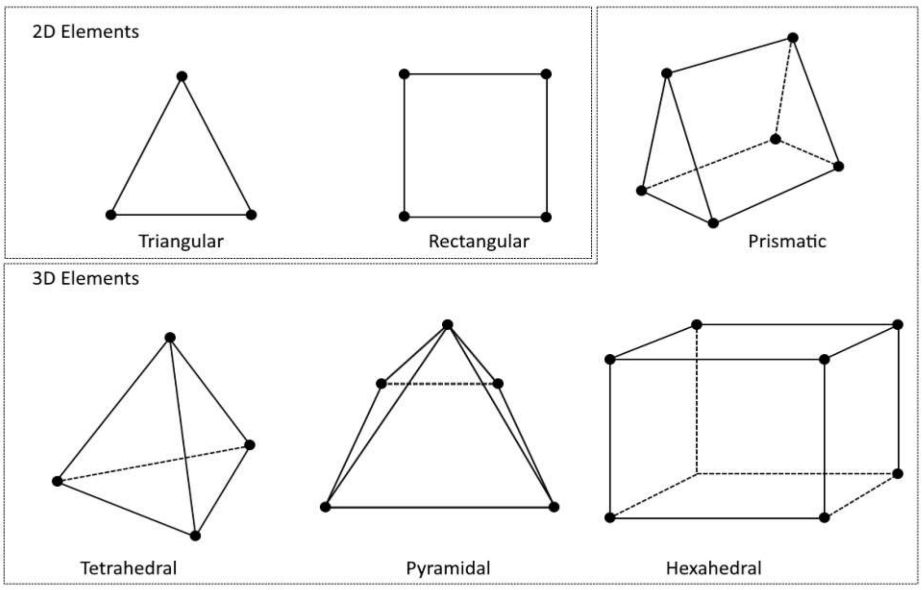


Figure 2.2: Types of different linear two-dimensional and three-dimensional elements used in common FEM commercial software.

When it comes to meshing, there are various possibilities on structuring the elements with each other. Block meshing can be used on different sections of the CAD geometry and elements of mixed type can be used to capture geometrical details in the best way possible, while still ensuring mesh conformality. Mesh adaptivity is another possibility in FEM because it allows mesh refinement in a defined region of interest through a refinement algorithm. Due to limitations in computing capabilities, this allows for a more efficient distribution of computing power where it is needed the most.

2.3 GDS layouts

GDS II is a binary database file format which has become the standard in the electronic design automation (EDA) industry for exchanging data related to integrated circuit(IC) layout artwork [12]. It represents planar geometric shapes, text labels, layer numbers; and other layout-related information in a hierarchical format. This means that repetitive structures can be defined once and then referenced multiple times in the layout, reducing the file size.

However, the GDS II format supports only *weakly simple polygons* which are polygons whose segments are permitted to intersect, but not cross, depicted in Figure 2.3. This means that shapes with holes can be supported only through their weakly simple variant, whereby the boundary of the inner hole shape must be connected to the boundary of the outer enclosing shape.

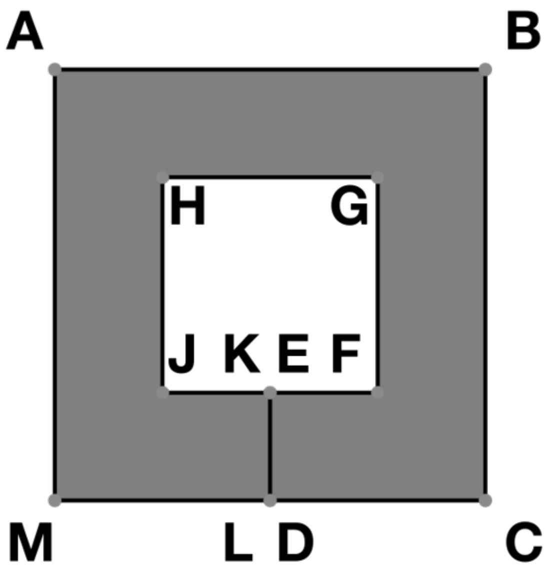


Figure 2.3: Weakly simple polygon.

The data contained in GDS II files can be used to recreate the entire design or parts of it, which is useful for sharing layouts, transferring the design between different tools, or creating photomasks. Originally designed for controlling integrated circuit photo mask plotting, GDS II quickly became the industry's default format for transferring IC layout data between design tools of different vendors, as they often used incompatible and proprietary data formats [13]. GDS II was developed by Calma Company for its layout design software. The legacy format "Graphic Data System" ("GDS") and "GDS II", are currently owned by Cadence Design Systems, Inc. Typically, GDS II files are the final output of the IC design cycle and are provided to silicon foundries for IC fabrication. The default length unit in GDS II format is μm and the file extension is .gds.

Viewing GDS layouts with KLayout

KLayout is an open-source software which was initially created and still maintained by Matthias Koefferlein for the purpose of mask layout editing in the microelectronics industry [14]. It has since grown in popularity and is now widely used by engineers, designers, and researchers for creating, viewing and editing layouts for integrated circuits, printed circuit boards, and micro-electromechanical systems (MEMS). KLayout is designed to be highly customizable through the use of plugins, which has contributed to its widespread adoption in industry and academia.

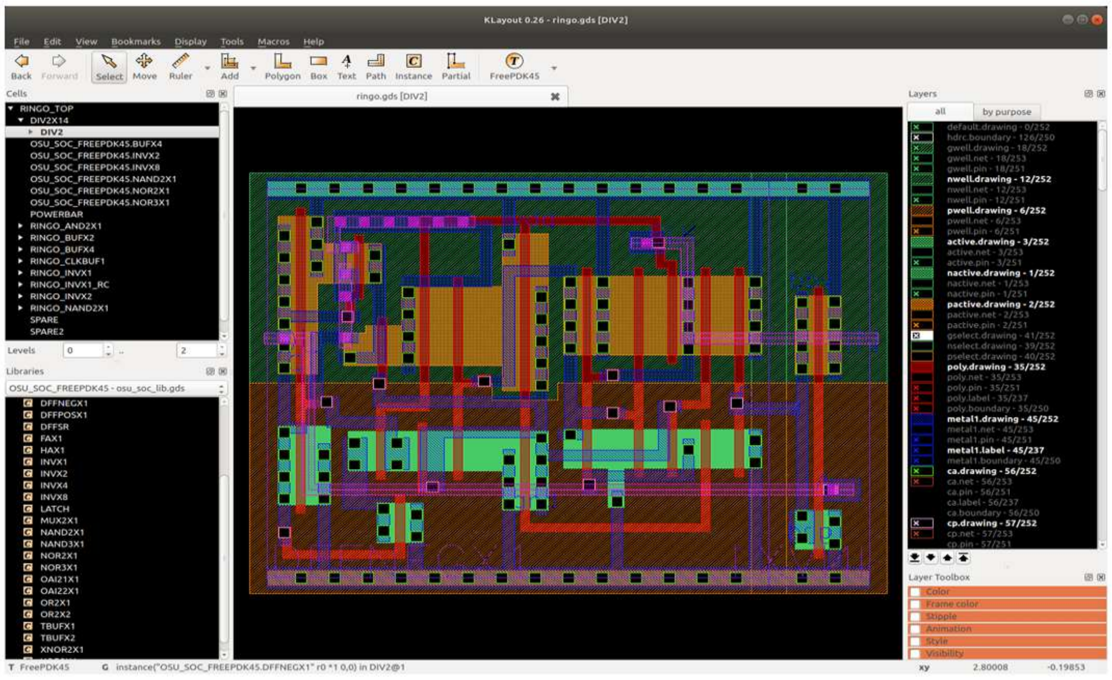


Figure 2.4: KLayout main window.

It is possible to view *.gds* files and their hierarchical tree-structure in KLayout' graphical user interface GUI. In Figure 2.4, the panel on the left shows the cell hierarchy and inside each cell there could be subcells which could also have further subelements in the hierarchy. The cell shown by default is the one that is the highest on the hierarchy, but custom ones can be selected to be displayed as well. Cells are important to define the structure and to reference different models or components which might be designed, for fabrication reasons in a single big *.gds* file.

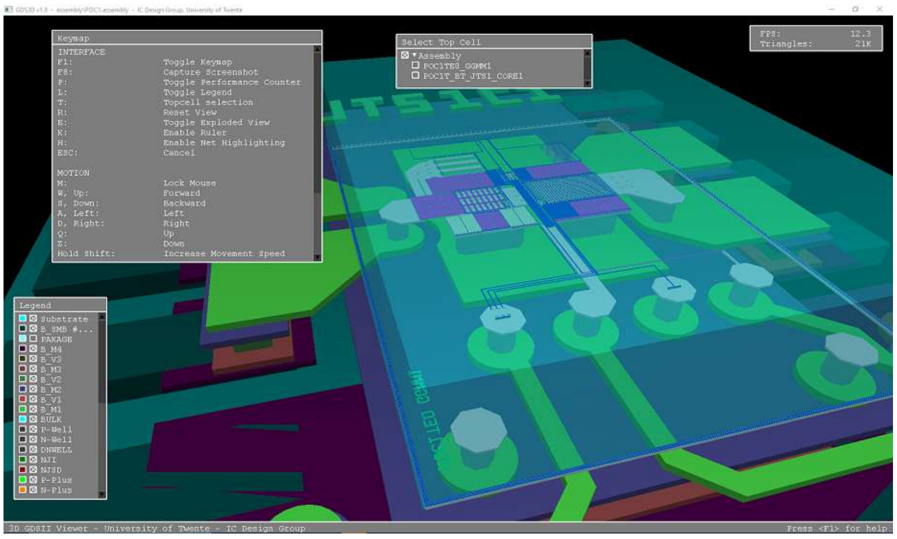
The panel on the right shows the layer list and each layer contains a list of 2-Dimensional (2D) drawn shapes. Layers are represented by the pair *(layer_number, datatype)*, where the *layer_number* is an integer between 0 and 65535 and the *datatype* is an integer between 0 and 255. These numbers have no predefined meaning and can be used by the design engineer to organize the layout in a structured manner. By default, all layers are displayed using different colours to display overlapping regions in a distinct way. The panel in the center is the canvas where the current layout is displayed.

2.4 Review of current state-of-the-art

In the past (Hoffmann et al., 2022) in [15] used 2D half models of polyheaters to perform transient thermal analysis FEM simulations. This information was then applied to calibrate the numerical fatigue damage models. The approach was successful, but more insights can be achieved when working with 3-Dimensional (3D) CAD models of polyheaters. There are several alternatives when it comes to 3D rendition of 2D .gds layouts, some of them are open source, but there are also commercial applications as described in the following.

GDS3D

GDS3D is an open source software developed at the University of Twente [16] for extruding 2D .gds layouts into 3D rendered objects. The user can navigate around the 3D visualization and enable or disable the extruded layers. It allows to export 3D objects into OneLab as .geo files, but it still has problems handling vias which can be present on the layout.



```
Keymap
INTERFACE
F1:                 Toggle Keymap
F8:                 Capture Screenshot
P:                  Toggle Performance Counter
L:                  Toggle Legend
T:                  Topcell selection
R:                  Reset View
E:                  Toggle Exploded View
K:                  Enable Ruler
H:                  Enable Net Highlighting
ESC:                Cancel

MOTION
M:                  Lock Mouse
W, Up:              Forward
S, Down:            Backward
A, Left:            Left
D, Right:           Right
Q:                  Up
Z:                  Down
Hold Shift:         Increase Movement Speed
```

Figure 2.5: Interface of GDS3D

GDS2POV

GDS2POV is another open source software focused mostly on 3D visualization of .gds layouts [17], based on POV-Ray which is an open- source, cross-platform ray tracer. However, this tool doesn not allow to export the 3D objects in any suitable CAD format which is useful to conduct finite element analyses.

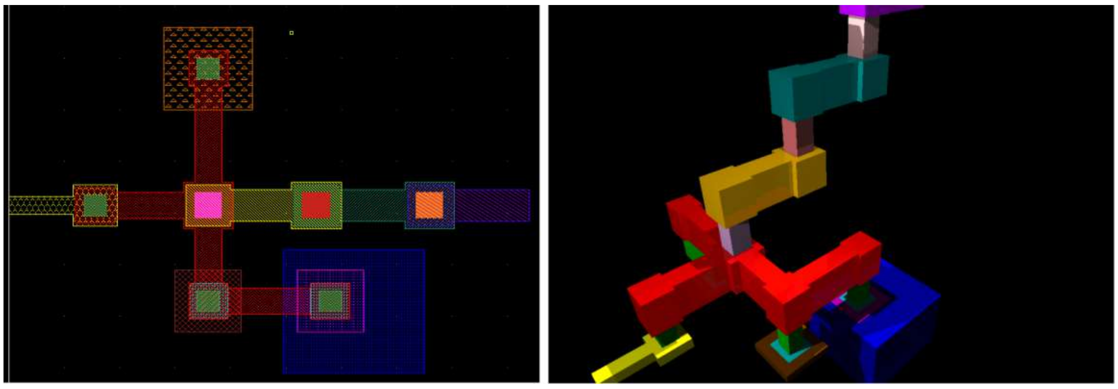


Figure 2.6: Functionality of GDS2POV.

gds3xtrude

gds3xtrude [18] is an open-source project in a very early phase developed by Thomas Kramer to convert .gds layouts into .scad objects based on OpenSCAD [19], which is a free software for creating solid 3D CAD objects through its scripting language. It allows to define the layer stacking procedure and the material sequence metadata through its own data structure, but in the end, the .scad format is not suitable to be meshed and it has to be converted to other CAD formats, Figure 2.7. During the conversion the material sequence metadata is not transferred and, in the end, we are left with an unlabelled 3D object.

Gds2Mesh

Gds2Mesh is a 3D technology computer aided design (TCAD) model construction tool developed by Cogenda [20], a company specialized in simulation software for semiconductor devices. The 3D CAD models are constructed by using a GDSII mask layout as input, with the ability to customize and predefine process rules. This software allows boolean operations between 2D shapes or 3D objects and, in the end, the output is a high-quality tetrahedral mesh as claimed on their product description. It is not known how well does the software perform on devices with very thin structures or layers, which we will

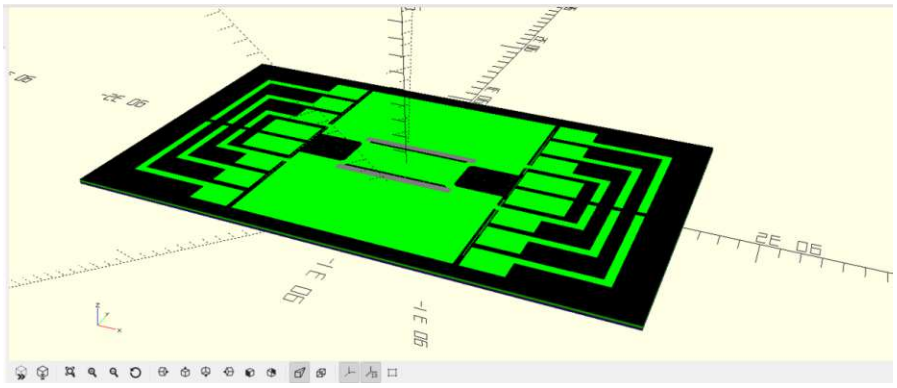


Figure 2.7: The .scad output of an initial test done on a polyheater .gds layout up to an intermediate layer.

investigate later on. The license of Gds2Mesh is mostly targeted to industrial enterprises and thus it is quite cost-intensive.

COMSOL Multiphysics® and Ansys Mechanical

COMSOL Multiphysics® and Ansys Mechanical are the most well known multiphysics suites used in industry, but also in academia. In regard to our task, currently COMSOL Multiphysics® supports the feature of creating 3D geometries from .gds files through its ECAD module [21]. However if the layer stacking procedure of the geometry itself is quite complex, this involves plenty of manual work which is not desirable. Ansys Design Parametric Language (APDL) [22] supports a similar workflow by first converting the .gds file into an .anf (Ansys neutral format) file and then proceeding with the layer stacking procedure similarly as in COMSOL Multiphysics®. In both cases the bigger the manual workload is, the higher the probability is to have human errors.

Coventor SEMulator3D

The SEMulator3D tool from Coventor is an advanced process and device emulation tool, [23] for the development and optimization of semiconductor fabrication processes, and for the design and analysis of semiconductor devices. It is mostly focused on providing a virtual fabrication environment, where the user can emulate and visualize each step in the semiconductor fabrication process, from photolithography to etching and deposition. This allows the user to optimize their processes, reduce costs, and improve yield and performance. After reading .gds input files, regarding CAD models for FEM analyses, this software allows to export 3D models to FEM solvers for additional analyses. However there is not much reference on this part due to the main focus being on process emulation. The license is aimed for enterprise industries and distributed on a yearly basis.

ViennaPS

ViennaPS, [24] is a header-only C++ process simulation library, developed at Institute of Microelectronics at TU Wien, which includes surface and volume representations, a ray tracer, and physical models for the simulation of microelectronic fabrication processes. Regarding .gds layouts ViennaPS has a module that can import these, and perform subsequent process simulations over these geometrical masks. It is possible to specify the layers to be extruded by their z-coordinates and also perform boolean operations over them in order to follow the layer stacking ruleset. For labelling, it is possible to specify material names for each volume, and this also assigns a color to each material volume. The 3D built-in models are exported in a .vtk [25] format which is already meshed by definition. As it is mostly focused on process emulation, it does not handle well the extrusion and meshing of structures with huge difference in spatial scales.

2.5 Workflow

In this work, the main goal is to generate 3D meshed CAD models, which are ready to undergo finite element analyses from provided .gds layout files of polyheater chips in a fully automated manner. As mentioned previously the .gds layouts were created for the purpose of defining all the geometrical shapes and details of the layout masks needed for the fabrication of the devices. Using .gds layouts to create 3D meshed CAD models for FEM analyses is a new and unconventional practice which saves time during CAD modelling as the CAD geometry is not being built from scratch, but from its different cross sections along its z-axis.

We are provided with a .gds layout which is used for the actual fabrication of the polyheater chips. This layout contains many small geometrical details that are crucial for the electrical functionality of the fabricated models, but are of negligible importance for the simulation models used in the thermal analysis. The idea is to remove or simplify these certain geometrical features from all the models automatically and then use these simplified models to build 3D extruded and meshed geometries, which are ready for thermal finite element analysis (FEA). The validity of of the automatically created CAD models will be checked using a transient thermal analysis, where FEM results will be compared to available experimental data obtained through the setup mentioned in Section 1.1.

CHAPTER 3

Simplification methodology

In the pre-processing phase of multiphysics simulations, significant time and effort is placed into setting up the geometry, meshing, specifying initial and boundary conditions, defining the physics of the problem, assigning material properties, etc. Usually, the imported CAD models come with many small and complex geometrical features which will need finer meshing, or a decrease in the meshing quality which will result in longer simulation times or incorrect simulation results, respectively. Motivated by this, CAD engineers usually, manually remove or simplify these small geometrical features which are of negligible importance. In more complex CAD designs this is simply not manageable. There has been significant effort placed into developing new algorithms which perform automatic simplifications or defeaturing of application-specific geometries. Thakur et al. [26] have written a great summary of several fully-automatic or semi-automatic simplification techniques used in a wide variety of applications, categorized as follows: techniques using surface entity based operators, volume entity based operators, explicit feature based operators, and dimension reduction operators. Lee et al. [27] showed that it is possible to use low-pass filters from signal processing to do geometrical featuring , while Owen and Shead in [28] employed machine learning techniques for CAD defeaturing.

Our approach consists as follows: we will use gdspy [29], an open source Python module which allows to read and write .gds files to convert all the geometrical data read through gdspy into a compatible data format to be read and manipulated through the Shapely Python Package [30]. Shapely is an extensive computational geometry library written by Sean Gillies and based on the famous C/C++ library GEOS [31]. All of the 2D simplifications over the .gds layouts will be carried out using the Shapely functions, then the simplified layouts will be written into new files using gdspy.

3.1 Reading with gdspy

The .gds file used in this study is developed internally and contains all the necessary details and information for the fabrication of 93 distinct polyheater models. The models are named according a specific ruleset which takes into account different design specifications which make each model unique. In the layout we are working on, the polyheater designs are placed in an array-like structure as shown in Figure 3.1. The hierarchy of the layout consists of a parent cell that contains subcells for each model. Each model subcell contains lower hierarchy subcells which include: model layout, border sealrings, and text subcells which serve for labelling purposes for different components and for the model naming. The layers are in the *(layer_number, datatype)* format as mentioned in Section 2.3 and they consist of: boxes, polygons, paths, or texts. These objects are the elementary entities which are included in the cell of a .gds drawing. Their definition in the gdspy context is given in [29].

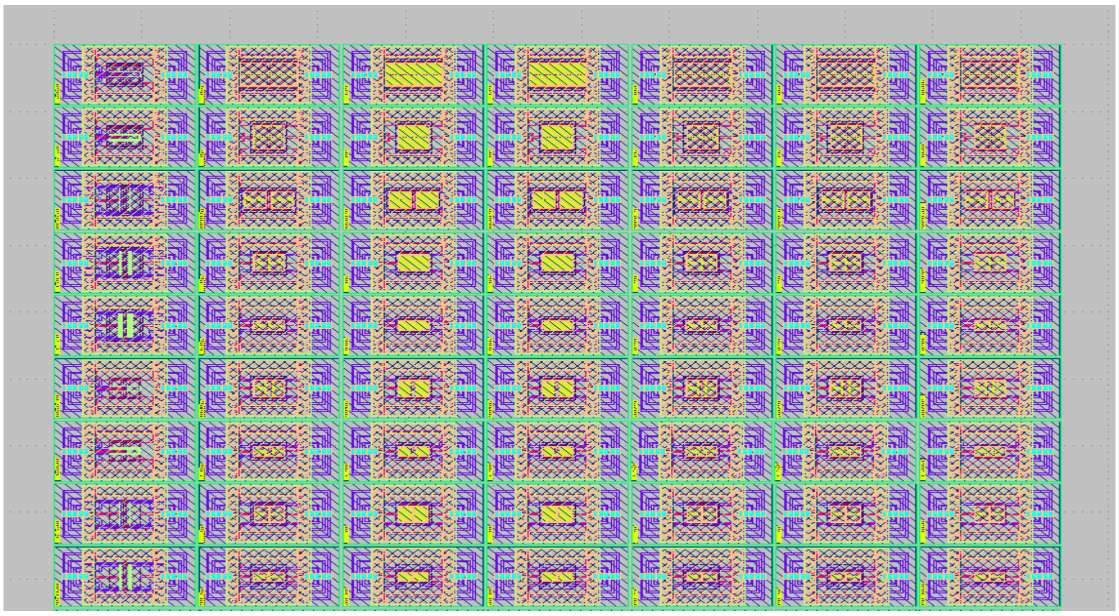


Figure 3.1: GDS layout used for fabrication.

In order to create 3D CAD geometries for each polyheater model in Figure 3.1, first we should crop each model from the array-like structure and save it in a separate .gds file. This can be performed easily through gdspy 's reading capabilites.

First we create a folder named *cropped_layouts*. Then, we load the large array-like .gds file through:

```
        big_lib = gdspy.GdsLibrary(infile=gds_file_path)
        dict_big_lib = big_lib.cells
```

In the context of gdspy, the .gds file is called a library which contains multiple cells. The list of cells of a library is a python dictionary, where the keys are the names of the cells and the values are the cell objects.

After the large layout is read-in, a list with the names of all the 93 models is used to create new files or new libraries using gdspy's `gdspy.GdsLibrary()`, which creates a new empty library. Subsequently, by enumerating through all the list of model names which are the keys of `big_lib.cells`, the cells of interest are filtered by value and added to new empty libraries separately. Any cell containing text or border seals is irrelevant for the CAD models and is filtered out. In the layout there is also a Process Control Monitoring (PCM) device that is used to control the manifacturing process and this is completely disregarded. The separate .gds files are written using: `gdspy.GdsLibrary().write_gds(name.gds)`.

3.2 Filtering by layers

The large array-like layout in Figure 3.1 contains many material layers which are important for fabrication, but are unnecessary from a thermal finite element analysis perspective. For example, there is an imide layer which serves as mechanical protection for the fabricated models, but for the simulation models, its impact is negligible, so it does not need to be included. There are various layers which serve for labelling various areas of the Cu layer, all to be discarded as well, in our modelling.

Therefore the remainder of work is on the cropped .gds layouts from the previous *cropped_layouts* folder. These separate .gds files inherit all the layers as defined on the large array-like .gds layout. Using a python function which takes 2 arguments: the path of the *cropped_layouts* folder and a list of the layers to be kept in the *(layer_number, datatype)* format, we enumerate through all the 93 .gds files on this folder. We create new separate empty libraries using `gdspy.GdsLibrary()` and inside the library create only 1 top hierarchy cell using `gdspy.GdsLibrary().new_cell(name)`. The intention is that each filtered .gds file has only 1 cell which makes the structuring and accessing of the geometric data straightforward. Then, using gdspy 's `get_polygons()` method applied to the loaded `gdspy.GdsLibrary(infile=file_path).cells` dictionary object, we obtain the list of polygons which itself is a python dictionary where the keys consist of *(layer_number, datatype)* pairs and the values are actual gdspy polygon coordinates. In gdspy, polygons can be defined through an ordered set of vertices a in clockwise or a counter-clockwise direction. An example of how a polygon is defined in the gdspy context is given below:

```
# Create a polygon from a list of vertices
points = [(0, 0), (2, 2), (2, 6), (-6, 6), (-6, -6), (-4,
    -4), (-4, 4), (0, 4)]
poly = gdspy.Polygon(points)
```

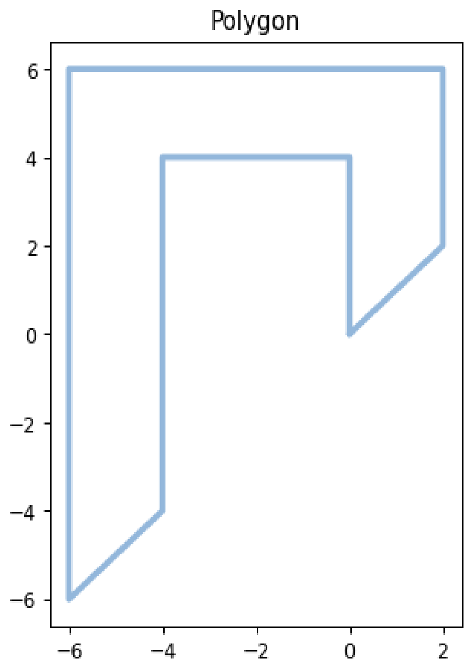


Figure 3.2: Polygon drawn using gdspy.

Then the dictionary containing all the polygons and the respective layers to which they belong, is filtered with respect to the list of layers which must be kept by:

```
filtered_polygons_dict = {k: v for k, v in
    polygons_dict.items() if k in layers_to_keep}
```

Due to design choices in fabrication, the .gds files do not contain a layer which has the box that defines the silicon substrate level. This is important for the simulation models and it is added manually in layer $(1, 0)$ in the single top cell we created.

```
        #add the silicon substrate base layer
        points = [(0, 0), (3220, 0), (3220, 1820), (0, 1820)]
        substrate = gdspy.Polygon(points, layer=1, datatype=0)
        new_cell.add(substrate)
```

Subsequently, all the new filtered .gds files are written with the same `.write_gds(name.gds)` method mentioned above in a new folder titled *filtered_layouts*. For ease of use instead of referring to the layers using numbers in the *(layer_number, datatype)* format we are going to refer them with strings. Now all of the filtered .gds files have the following layers: `['substrate', 'contact', 'M1', 'poly', 'TVia', 'Cu']`. Now after having all the layouts cropped and filtered by layers we can continue with the 2D simplification procedure.

3.3 Simplification procedure in 2D

Due to the fact that gdspy doesn't offer much functionality in terms of computational geometry over 2D shapes, we will use the Shapely library in this regard. Shapely offers a multitude of geometric objects and functionalities required for the simplification task. The basic geometric entities in Shapely are defined as:

```
        point = Point(1, 1)
        line = LineString([(2, 0), (2, 4), (3, 4)])
        ring = LinearRing([(0, 0), (1, 1), (1, 0)])
        polygon = Polygon([(0, 0), (1, 1), (1, 0)])
```

A LinearRing has zero area and non-zero length, while a Polygon has both non-zero area and length. We will have to deal mostly with polygons and need to set some preliminaries first.

One of the functions from Shapely which is quite useful for our task is the `shapely.buffer` function. It computes the buffer of a geometry for positive and negative buffer distance. The positive buffer distance corresponds to dilation, while the negative buffer distance corresponds to erosion in terms of mathematical morphology. Their definitions are as follows [32]:

Dilation

The dilation of polygon A by the structuring element B is:

$$A \oplus B = \bigcup_{b \in B} A_b \tag{3.1}$$

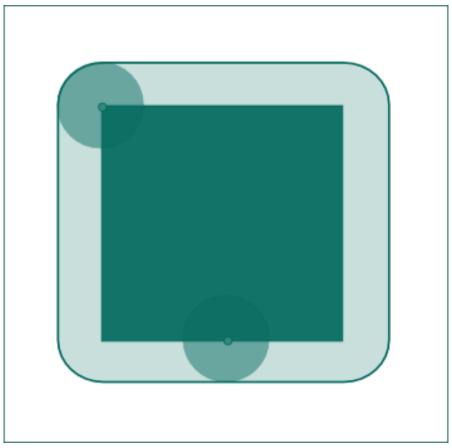
If B has is centered at the origin, as before, then the dilation of A by B can be understood as the locus of the points covered by B when the center of B moves inside A.

Erosion

The erosion of a polygon A by the structuring element B is:

$$A \ominus B = \{z \in E | B_z \subseteq A\} \tag{3.2}$$

where B_z is the translation of B by the vector z, i.e., $B_z = \{b + z | b \in B\}, \forall z \in E$.


(a) Dilation of a square by a disk.

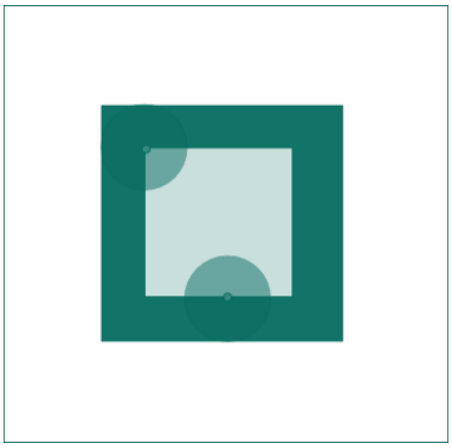

(b) Erosion of a square by a disk.

Figure 3.3: Mathematical morphology operations.

The buffer operation always returns a polygonal result. The negative or zero-distance buffer of lines and points is always empty.

The files from the *filtered_layouts* will be read one by one using gdspy and the `gdspy.Polygon` objects will be translated into Shapely polygon format defined on the snippet above. When working with the Shapely polygons we also need to know the layer they originate from, so we create a `myPolygon` class with the following attributes:

```
        class myPolygon:
            def __init__(self, layer, polygon) -> None:
                self.layer = layer
                self.polygon = polygon
```

Here, the layer is a string as defined above in the layer naming, and polygon is a Shapely `Polygon` object. For model symmetry purposes, all the layouts will be shifted to the origin of the plane. meaning that $(0, 0)$ is at the centroid of a polyheater model.

3.4 Procedure on the 'M1' layer

The 'M1' layer consists of aluminium temperature and delamination sensors. Compared to all the other layers, this particular layer is by far the most diverse in terms of design accross different models. It has many intricate and small geometrical features which are not very important for the thermal simulation and can be simplified.

Motivation

Consider the following polyheater model, where a fraction of M1 and Cu layers are displayed:

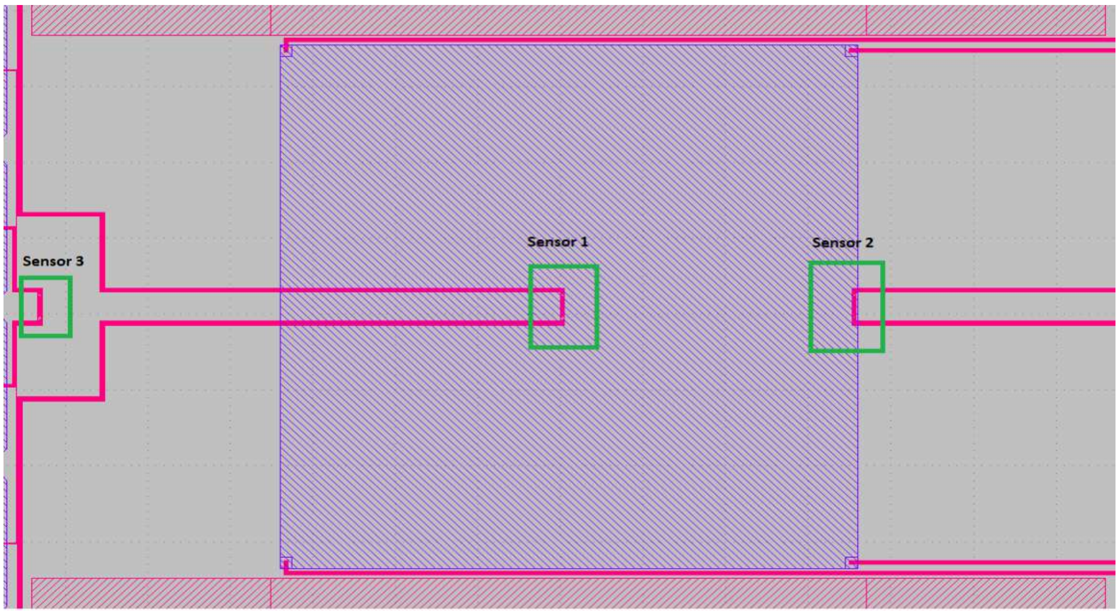


Figure 3.4: Temperature sensors in a meandre structure are highlighted through the green boxes.

Due to very small time scales (short heat pulses of $200\mu s$-$300\mu s$) the heat applied to the main copper plate (the big blue rectangle in the center) needs several milliseconds to be propagated and reach sensor 3 in Figure 3.4. For sensors 1 and 2, since they overlap with the Cu plate, the following can be said regarding their heat footprint in the global scale. The thermal conductivity of copper is approximately 398 $W/(m \cdot K)$ while for aluminium it is 237 $W/(m \cdot K)$. Due to the fact that the thickness of the M1 layer of aluminium is much smaller compared to the thickness of the copper layer (from the design specifications) we can say that the M1 layer is not thermally significant and various simplifications can be applied as shown below. Obviously by filling the empty areas between M1 polygons with aluminium, while in reality there is oxide, we are introducing a small error, but this is negligible in the global scale. Similar simplifications are applied to other M1 structures (Fig. 3.6, 3.7).

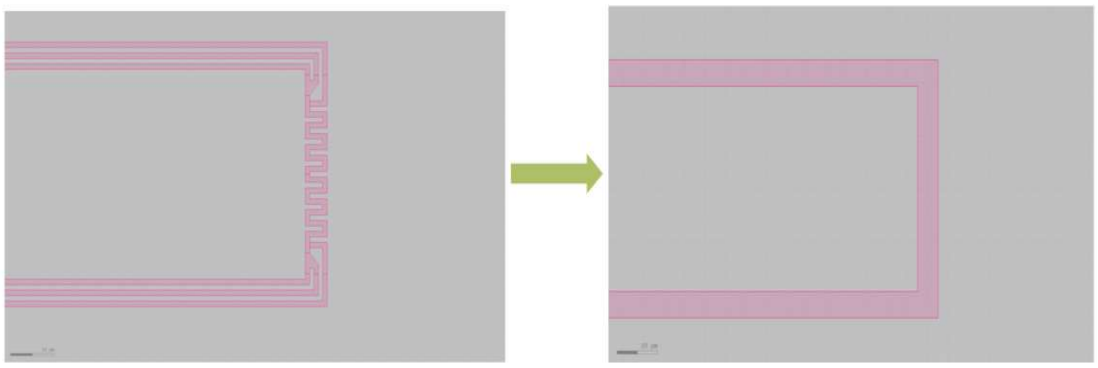


Figure 3.5: Simplification of the meander structure of the temperature sensor.

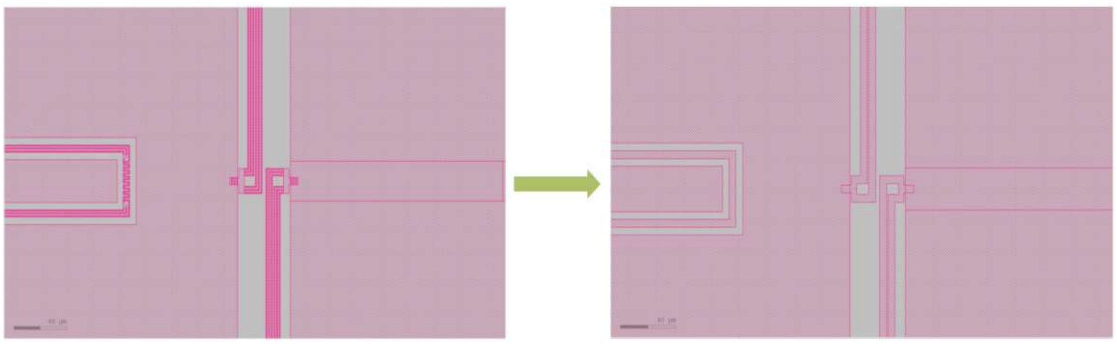


Figure 3.6: Simplification of the delamination sensor.

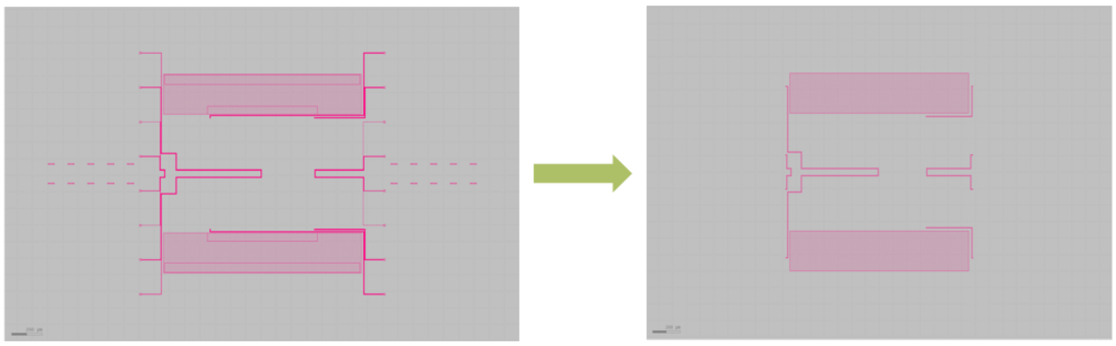


Figure 3.7: Simplification of the connecting lines of M1 layer far away from the heated central area.

Implementation

The implementation of the simplifications pictured in Figures 3.5, 3.6 and 3.7 is performed by splitting the list of myPolygons from the M1 layer according to an area criterion and working with them separately. The meander structure of the temperature sensor is filtered out through a function which checks if a polygon has more than n coordinates with the same x-coordinate. Then, dilation and erosion by the same buffer size gives the result in Figures 3.5 and 3.6. In Figure 3.7 the cutting of the connecting lines is performed using linestring information from the copper pads and the removal of the extremal connecting lines is achieved through one-sided negative and positive buffering. In all buffering operations the `shapely.BufferCapStyle` is chosen to be flat and the `shapely.BufferJoinStyle` is chosen to be mitre in order to preserve the current polygon geometries.

3.5 Procedure on the 'contact' layer

The contact layer consists of very thin tungsten stripes between the poly and M1 layers. These have a fairly easy geometry to be simplified and are the same accross all models.

Motivation and implementation

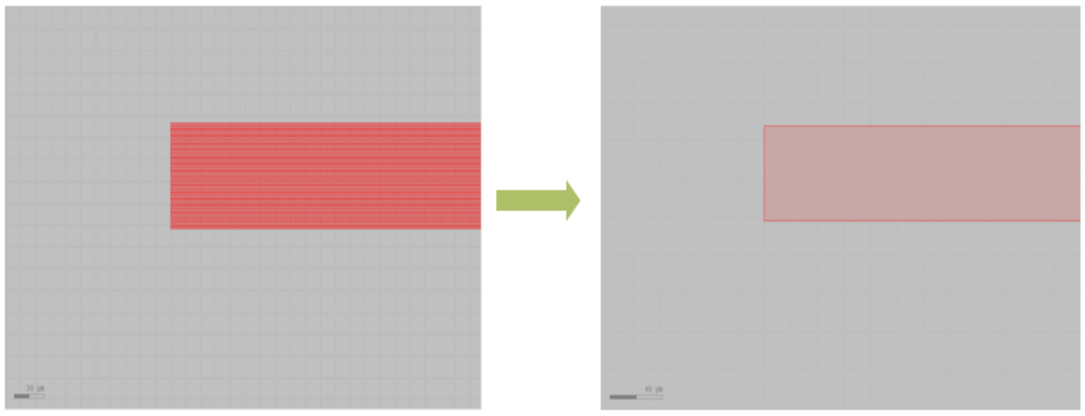


Figure 3.8: Simplification on the contact layer.

The simplification above is simply erosion and dilation by the same size and then merging. By doing this, we are making the thermal contact between poly and M1 layers 50% better than it is in reality; however this doesn't affect the heating of the Cu through the poly plate significantly, since the M1 layer is thermally negligible, as discussed in the previous section.

3.6 Procedure on the ’TVia’ layer

The TVia layer contains many vertical conductive channels of different scales that connect M1 to Cu layer. They are placed on the chip in order to dissipate the heat coming from the underlying poly plate. This layer has the largest number of polygons compared to the other layers as it has a repeating structure of small 3µm × 3µm boxes which make up the majority of the interconnecting channels.

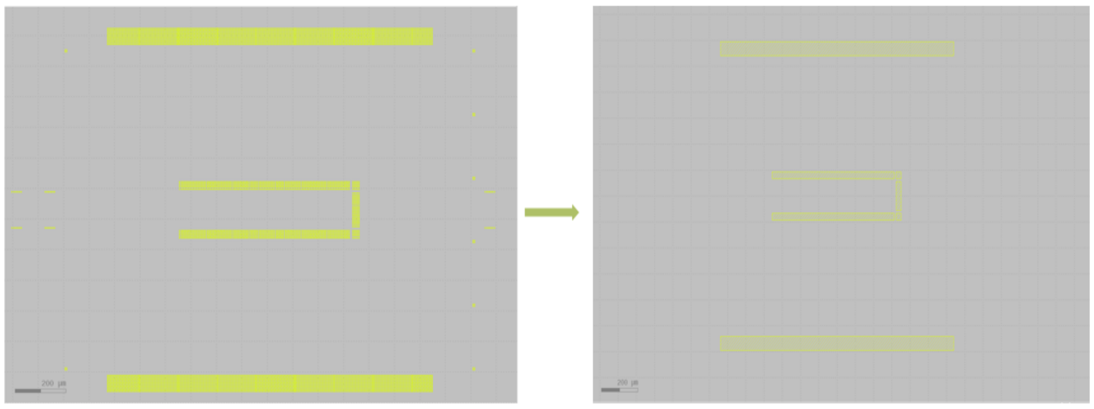


Figure 3.9: Simplification on the TVia layer.

Motivation and implementation

In order to motivate this simplification we must do a simple reasoning on the heat capacity of the entire polyheater device. The heat capacity is an extensive property (its magnitude increases as the size of material volume increases) of materials and it defines the amount of heat to be supplied to an object to increase its temperature by 1 Kelvin. The heat capacity of an object is calculated by multiplying the specific heat capacity (intensive property) to its mass. The heat capacity of our polyheater device in terms of material specific heat capacities would be as follows:

$$\begin{aligned} C_{polyheater} &= C_{Si} + C_{Cu} + C_{Al} + \ldots \\ C_{polyheater} &= c_p^{Si} \cdot m_{Si} + c_p^{Cu} \cdot m_{Cu} + +c_p^{Al} \cdot m_{Al} + \ldots \\ C_{polyheater} &= c_p^{Si} \cdot \rho_{Si} \cdot V_{Si} + c_p^{Cu} \cdot \rho_{Cu} \cdot V_{Cu} + +c_p^{Al} \cdot \rho_{Al} \cdot V_{Al} + \ldots \end{aligned} \tag{3.3}$$

Due to the fact that the silicon and copper volumes make up more than 90% of the entire device volume, the amount of extra dielectric that is being removed due to the merging of the small Tvia holes is fairly small compared to the silicon and copper amounts which dominate in the device. This fairly small volume removal does not affect the heat capacity of the entire polyheater significantly, where C_{Si} and C_{Cu} dominate the heat capacities of all the other remaining materials. The simplification can then be performed as shown on Figure 3.9 by applying dilation and then erosion by the same size and then merging all the polygons of the TVia layer.

3.7 Procedure on the ’Cu’ layer

Motivation

The polygons which are used to draw the copper pads of the Cu layers contain small chamfers in their extremal vertices. They are mostly present on the small 16 copper pads that are used for electrical actuation and the main copper plate in the middle may also have them, based on the polyheater model. As shown in Figure 3.10 we could fill up the empty area described by the half area of the small red square with copper material. This simplification would be valid as the new volume added is a very small percentage compared to the total volume of copper.

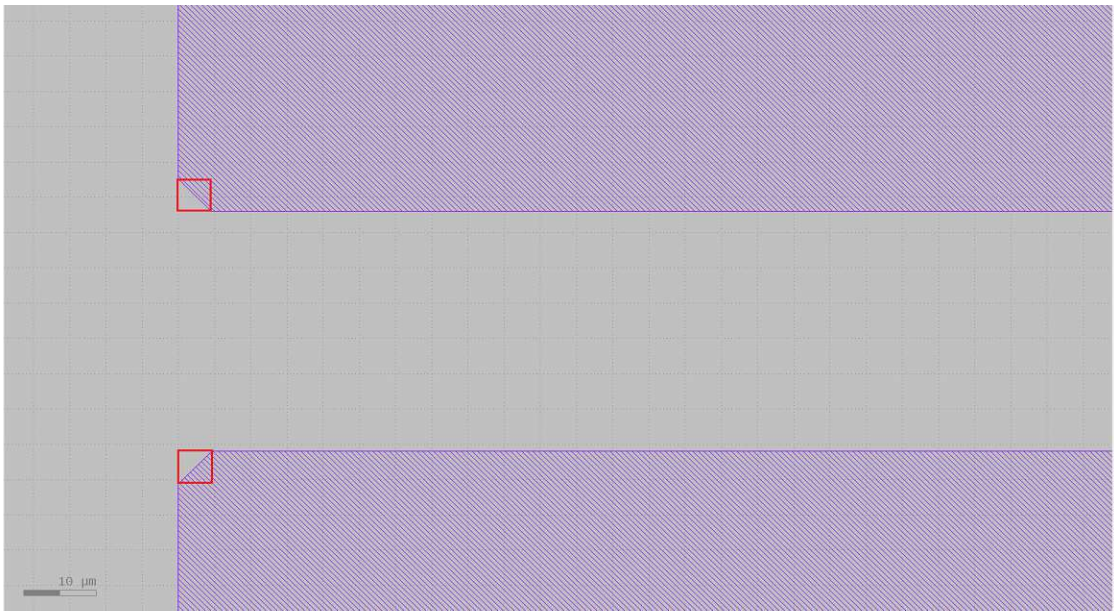


Figure 3.10: Chamfer detection and removal.

Implementation

In all the studied models there are three different chamfer lengths which have been used. They have been found by checking if adjacent edges form a 135° angle. These lengths are the *specified lengths* in the algorithm below. Subsequently, the algorithm is as follows: find the chamfers and use them as a defining diagonal to build the red squares seen in Figure 3.10. Then, use Shapely 's union function to merge the polygon with the chamfer to the newly built squares.

Algorithm 3.1: Algorithm to remove Cu chamfers

```
Data: list of myPolygons
Result: list of myPolygons
new_Cu ← [];
Cu_myPolygons ← myPolygon objects whose layer attribute is 'Cu';
for element in Cu_myPolygons do
    for $i = 0$ to length(element.polygon.exterior.coords)-1 do
        p1 ← Point(element.polygon.exterior.coords[i]);
        p2 ← Point(element.polygon.exterior.coords[i + 1]);
        edge ← p1.distance.p2;
        if edge == specified lengths then
            diagonal ← edge;
            side_length ← diagonal/ $\sqrt{2}$;
            center_x← (p1.x + p2.x)/2;
            center_y← (p1.y + p2.y)/2;
            v1← Point(center_x - side_length / 2, center_y - side_length / 2);
            v2← Point(center_x + side_length / 2, center_y - side_length / 2);
            v3← Point(center_x + side_length / 2, center_y + side_length / 2);
            v4← Point(center_x - side_length / 2, center_y + side_length / 2);
            square←Polygon([v1, v2, v3, v4]);
            union ← element.polygon.union(square)
             new_Cu.append(myPolygon(layer=element.layer, polygon=union)
        else
            new_Cu.append(element)
        end
    end
end
return new_Cu
```

3.8 Algorithm to remove excess collinear vertices

Due to various simplifications which we performed to the layout polygons, most of them will contain extra collinear vertices (Figure 3.11) which serve no purpose and could distort the mesh later on. The following algorithm was devised to remove these excess vertices: check $\vec{a} \times \vec{b}$ where $\vec{a}$ and $\vec{b}$ are formed by each consecutive triple points. If the cross product is 0, then the 3 points are collinear and the middle point is removed.

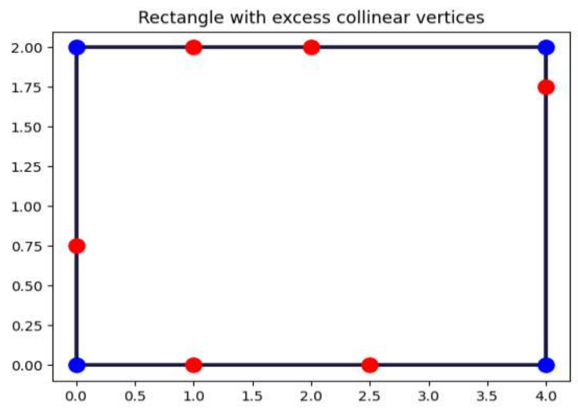


Figure 3.11: Essential vertices are plotted with blue dots, while excess collinear vertices are plotted with red dots.

Algorithm 3.2: Algorithm to remove excess collinear vertices

```
   Data: shapely Polygon object
   Result: shapely Polygon object
 1 new_vertices ← [];
 2 current_vertices ← polygon.exterior.coords;
 3 for $i = 0$ to $length(current\_vertices)$ - $2)$ do
 4     x1,y1 ← boundary_vertices[i];
 5     x2,y2 ← boundary_vertices[i+1];
 6     x3,y3 ← boundary_vertices[i+2];
 7     if $(x2 - x1) \cdot (y3 - y2) - (y2 - y1) \cdot (x3 - x2)) \neq 0$ then
 8         new_vertices.append(Point(x2, y2))
 9     end
10     x1, y1 ← boundary_vertices[prelast index];
11     x1, y1 ← boundary_vertices[last index];
12     x1, y1 ← boundary_vertices[first index];
13     if $(x2 - x1) \cdot (y3 - y2) - (y2 - y1) \cdot (x3 - x2)) \neq 0$ then
14         new_vertices.append(Point(x2, y2))
15     end
16 end
17 return new_vertices
```

CHAPTER 4

Extrusion and meshing

Now that we have a simplification methodology which is valid for all models, the next steps for building a FEM model from a simplified .gds file are 3D extrusion and meshing. It is possible to perform these tasks manually, but we are aiming for an automated procedure for speedup. Automation should be possible here, due to the fact that all the polyheater models have a fairly similar extrusion procedure. The procedure applies a stacking ruleset which is based on the overlap of various layers, as shown in Figure 4.1. The material sequence from bottom to top is as follows: Silicon substrate, Oxide, Polysilicon, Dielectric, Aluminium, Dielectric, Tungsten and Copper.

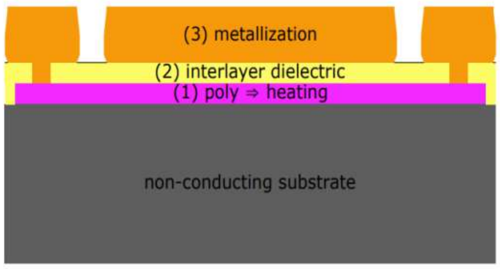


Figure 4.1: Example of the stacking procedure on a schematic model.

Before doing the extrusion, we must consider that in reality the microscopic images of polyheaters have shown that the fabricated 3D models are not flat and exhibit some sort of rough topography (Figure 1.3) which comes from non-idealities in the fabrication processes. Therefore, in an attempt to keep these topography features, we opted at first to perform the 3D modelling using a process simulation tool.

4.1 Tetrahedral meshes in ViennaPS

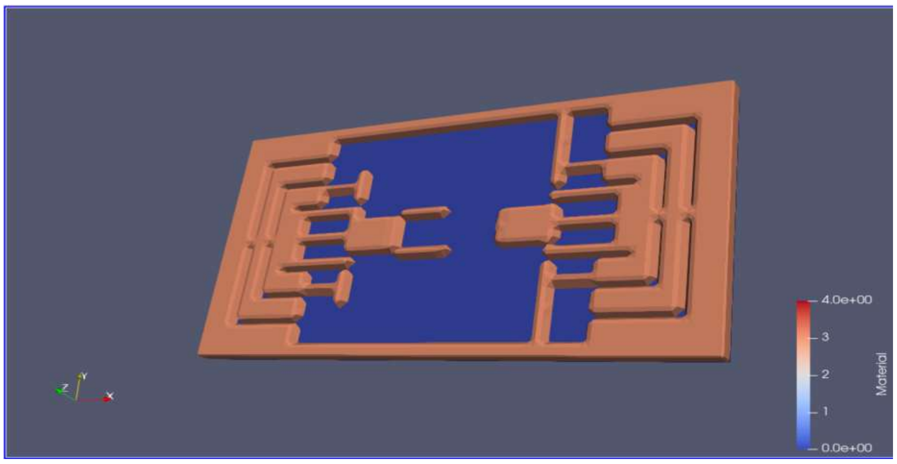


Figure 4.2: .vtk export of an extruded model up to an intermediate layer using ViennaPS.

We attempted to apply ViennaPS (Figure 4.2) for the 3D modelling with the aim of modelling steps in the topography generation with ViennaPS's process emulation based on the level set method. Due to the fact that the layer thicknesses range from nanometers (dielectric layers) up to hundreds of micrometeres (silicon substrate) it is impractical to use a ViennaPS .vtk mesh to resolve the thinnest layers. Resolving these nanometer-scale layers would mean extensive time and memory requirements which make the task unfeasible. This is primarily due to using tetrahedral elements, and applies to all tools which use such meshing techniques (a long survey on the meshing strategy will be given below). ViennaPS is currently under heavy development, and working with very thin structures and dynamic meshes could be an added feature in the future.

4.2 Working with Gmsh

Gmsh [33] is an open-source 3D FEM mesh generator with powerful built-in pre-processing capabilities. It has been written by Christophe Geuzaine and Jean-François Remacle at the University of Liège [34]. Gmsh can be controlled through its GUI, the command line, using text files written in Gmsh's own scripting language (.geo files), or through the C++, C, Python, Julia, and Fortran application programming interface (API). The goal in this work is to exploit Gmsh's powerful parametric input through scripts written using its Python API.

4.2.1 Swept meshes

Tetrahedral meshes on very thin geometries can be very inefficient and of poor quality with a lot of elements that have a very bad aspect ratio. A bad aspect ratio refers to a finite element shape which is significantly elongated or distorted. Usually, this means the ratio of the longest edge of the element to the shortest edge is very large. In numerical analysis, bad aspect ratio elements are not preferred as they cause instabilities, convergence problems, and inaccuracies in the FEM simulations. Eventually swept meshes, another meshing strategy was investigated.

Swept meshing is a meshing technique used on 3D or 2D geometries. For 3D geometries, its main idea consists of extruding a surface mesh throughout the domain. Not every 3D domain can be meshed through swept meshing. In general, 3D domains which have disjoint or discontinuous boundary surfaces cannot be meshed through swept-meshing. In a swept mesh, there can only be hexahedrons and prisms arranged in a structured or a semi-structured manner. In the sweep direction the subvolumes are always structured.

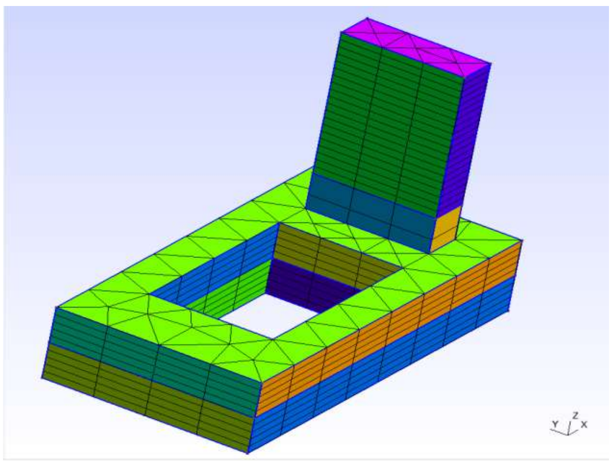
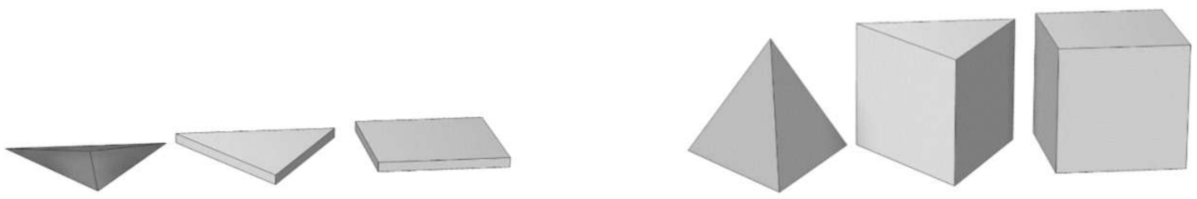


Figure 4.3: Simple example of a swept mesh over a 2D geometry with a hole in Gmsh.

The main advantage of swept meshing is that it uses minimal computing resources (low number of mesh elements) compared to tetrahedral meshes. It produces high quality meshes on thin geometries using only hexahedra or prisms and it is very efficient to generate, as only the 2D mesh of the source surface needs to be generated and that has to be extruded along the inner axis of the geometry until reaching the target surface as shown in Figure 4.3.

The good quality meshes over thin geometries can be attributed to the fact that prisms or hexahedra can be of good skewness quality (cosine of the smallest angle) even when the element is very flat. In other words, they can be stretched or compressed along one direction, and their skewness quality will not be impaired, unlike with tetrahedra [35], as illustrated below:



(a) Almost flat variants. (b) Optimal shapes.

Figure 4.4: A tetrahedron, a prism, a hexahedron in flat and optimal shapes. Skewness parameter is large for the flat tetrahedron, but does not change for the flat prism and hexahedron due to their angles not changing after being flattened.

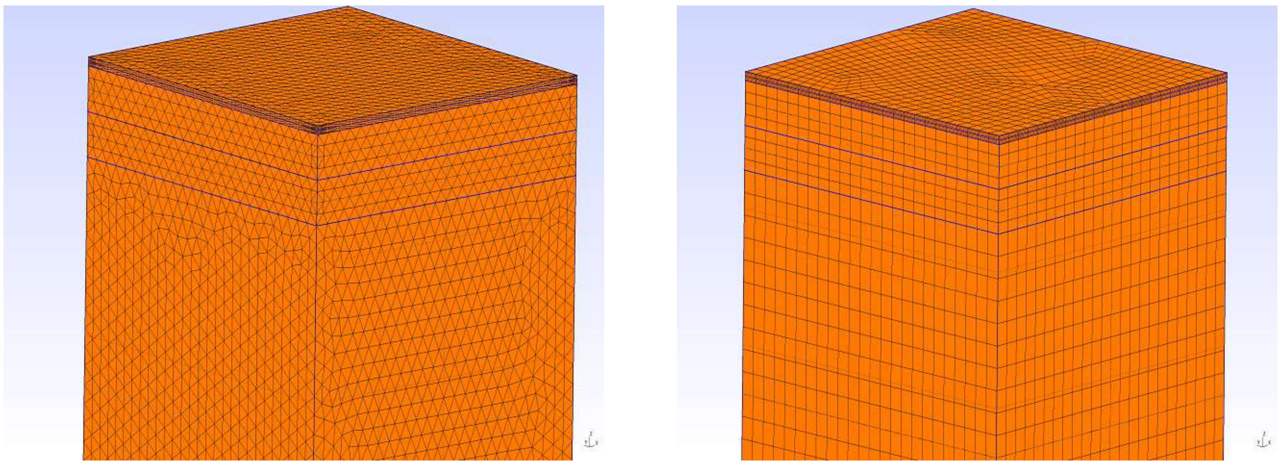
Another aspect ratio parameter of volume versus length should be taken into account for swept elements. There should not be a very high ratio between the swept element volume and the longest/shortest edge of the element.

We note that there is a clash inbetween mesh quality parameters like skewness and volume to length aspect ratio. It is up to the user to find a compromise between these parameters by understanding the problem physics. For example, high aspect ratio meshes should not be used in FEM problems where the solution field has a strong cross gradient, meaning that there are sudden changes in the value of the physical quantity of interest. In our polyheater thermal analysis, we expect that the copper plate will be heated uniformly, so moderate aspect ratios are allowed.

In our study-case, the polyheater layer stacking procedure produces watertight volumes with continuous boundaries and no gaps inbetween. Motivated by swept meshes used for very thin structures for FEM simulations in MEMS devices (Figure 4.5b) we will aim to do the equivalent through our small project gds2Gmsh.

4.3 gds2Gmsh

A model in Gmsh is defined by its boundary representation which means that: a volume is bounded by a set of surfaces, a surface is bounded by a series of curves, and a curve is bounded by two end points. Each of these geometrical entities is defined by a pair:



(a) Tetrahedral mesh. **(b)** Swept hexahedral mesh.

Figure 4.5: Meshes for geometries with thin layers at the top, performed in Gmsh.

`(dimension, tag)`, which specifies their dimension (0 for points, 1 for curves, 2 for surfaces, 3 for volumes) and their tag, a strictly positive global identification number. Entity tags are unique per dimension.

Most Gmsh models are built in a bottom-up manner (first points, then curves, surfaces and volumes) using its GUI (manually) or its APIs (scripted) through the *built-in* or *OpenCASCADE* [36] geometry kernels. In gds2Gmsh we use Gmsh 's Python API to build 3D extruded and swept-meshed models by first extracting all the polygon data from the simplified .gds files by using gdspy and Shapely libraries. The entire implementation is performed in a fully automated way with minimal user input.

4.3.1 Splitting GDS polygons

The first task is to read a simplified .gds layout using the gdspy library and obtain all the GDS polygons, then to translate these polygons in their corresponding Shapely format. These Shapely polygons need to be translated further into the Gmsh format in order to be added to the Gmsh model, which means that they should be added by a script in the bottom-up manner, meaning:points, lines, polygons, areas, volumes.

Recalling the extrusion procedure: It consists on a stacking ruleset which is based on the overlap of various layers which are defined in the end of Section 3.2. Due to swept-meshing the entire 2D area which is the projection of the 3D model at $z = 0$ needs to be meshed, then the meshed areas will be extruded, along with their mesh, to various z-levels according to the layer stacking procedure, and thus 3D material layers will be built.

The layers on the simplified layouts overlap several times with each other, there are even regions where 5 different layers overlap, making the extrusion very difficult to perform according to the layer stacking ruleset. Driven by this, the following approach was applied: Collapse all the 6 layers of a simplified layout into a single layer of split/disjoint polygons which means all overlapping polygons from different layers will be split in non-overlapping polygons by using their line boundaries as splitting boundaries. This can be simply done by using Shapely's `split()` function which allows to split one geometry by another one. In our case one polygon is split through the boundary lines of another intersecting polygon as shown below.

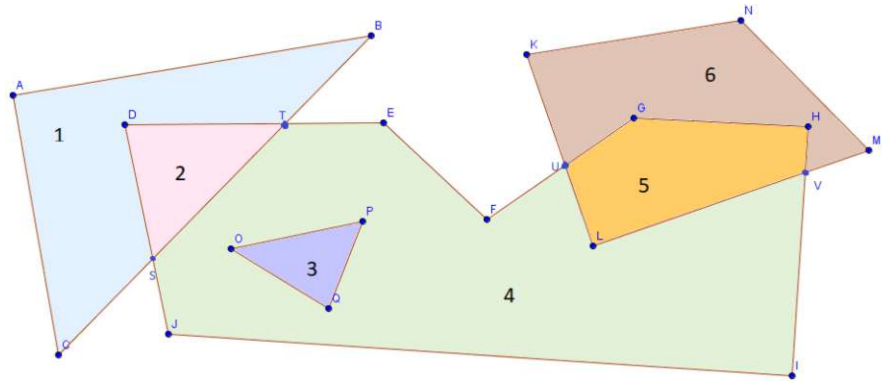


Figure 4.6: Shapely's splitting in action: All the labelled polygons are disjoint. Polygon 4 is disjoint from 3 because it has a hole there.

Differently from the GDS format, both Shapely and Gmsh support polygons with holes. The proper representation of holes and cavities is quite essential for the implementation, as we have many overlapping polygons which contain one-another. This will lead to polygons with holes and separate polygons such as 3 and 4 in Figure 4.6. In Shapely, a simple polygon with 2 holes is defined as follows:

```
        exterior = [(0, 0), (0, 5), (5, 5), (5, 0)]
        hole1 = [(1, 1), (1, 2), (2, 2), (2, 1)]
        hole2 = [(3, 3), (3, 4), (4, 4), (4, 3)]
        polygon = Polygon(shell=exterior, holes=[hole1, hole2])
```

To add a surface with holes in Gmsh, the external shell must be defined first and then the holes. Adding a list of surface tags where the tag of the outer surface shell is the first element and the following ones are the tags of surface holes, is a means to tell Gmsh to add a surface with holes.

Applying the `split()` function to all the polygons of a simplified layout, collapsed in a single layer, is shown in Figure 4.7.

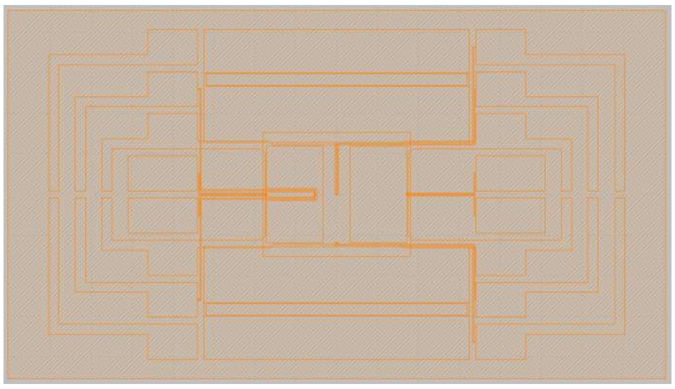


Figure 4.7: Shapely's splitting in a simplified polyheater model.

A new object is then defined:

```
        class mySplit:
           def __init__(self, split_polygon, origin_layers:
              List[str]) -> None:
              self.split_polygon = split_polygon
              self.origin_layers = origin_layers
```

In the above class `split_polygon` is the Shapely `Polygon` object of each split polygon and `origin_layers` is a list with the names of all layers in which this split polygon is included in. The information in `origin_layers` is obtained through checking every split polygon if it is contained in all other original polygons coming from the 6 layers: `['substrate', 'contact', 'M1', 'poly', 'TVia', 'Cu']` by using Shapely 's `.within()` function. The `a.within(b)` returns `True` if geometry `a` is completely inside geometry `b` (even if their boundaries overlap).

4.3.2 Meshing and extruding

Having all the `mySplit` objects for a simplified model, we can start adding them to our Gmsh model through Gmsh's Python API which needs to be initialized:

```
        gmsh.initialize()
        gmsh.model.add("model1")
```

We use the OpenCASCADE geometry kernel within Gmsh as it offers various extrusion functionality. All of the `mySplit` objects's 2D *Polygon* objects to our Gmsh model,

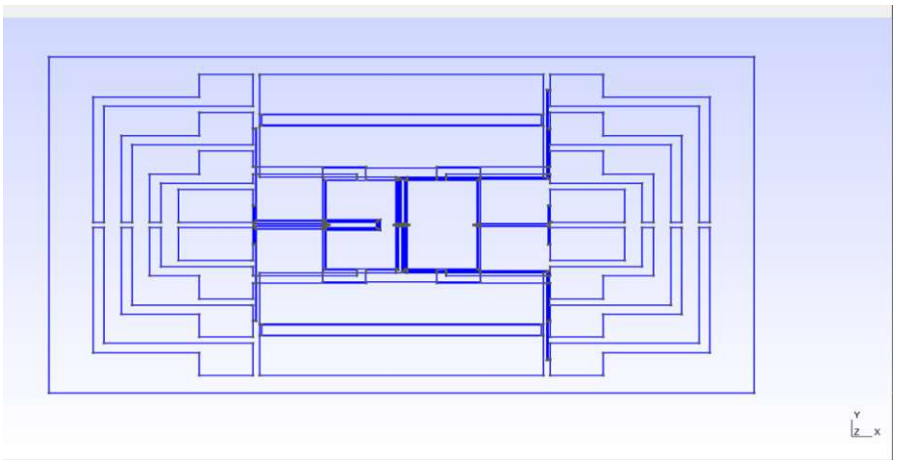


Figure 4.8: Initially added split polygons in Gmsh.

taking care of the holes as discussed above, Figure 4.8. The initial mesh of the added split polygons without performing any extrusion is given in Figure 4.9:

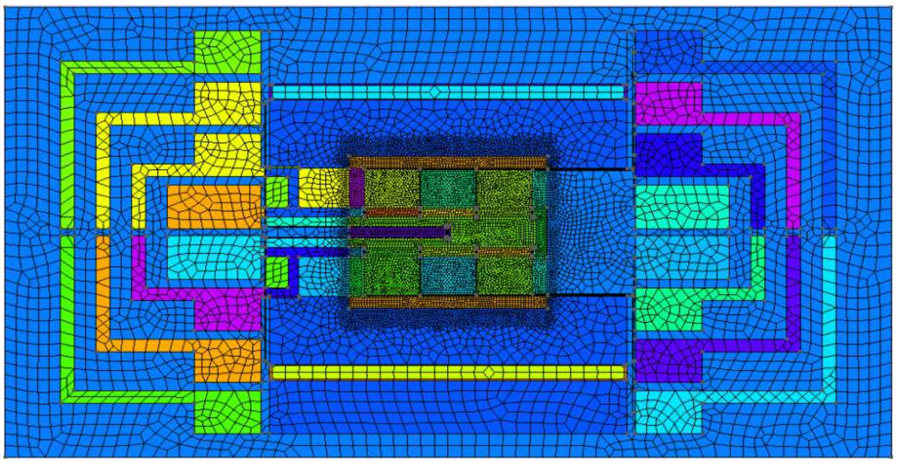


Figure 4.9: 2D mesh of the split polygons which will be extruded along the z-axis.

The extrusion is performed by checking the `origin_layers` of each split polygon against the layer stacking procedure. This produces the extruded volumes which are placed on top of each other, forming a material sequence for each split polygon. The number of subvolumes for each extrusion is chosed based on the layer thickness. The mesh shown on Figure 4.9 corresponding to each split polygon is extruded from $z = 0$ up to its z_{max}. The refinement in the center is obtained through Gmsh's mesh size fields. We can define an adapted mesh size field for all polyheater models by choosing a rectangular mesh size field whose coordinates are 50µm longer than the coordinates of the poly plate in absolute values. The labelling of the materials is performed through Gmsh's Physical Groups. Volumes of the same material are added to a corresponding physical group named after the material.

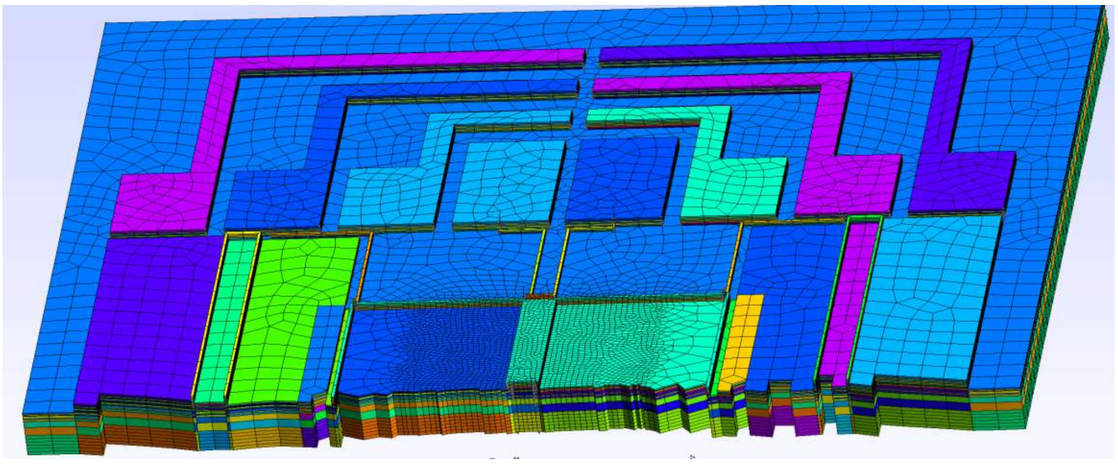


Figure 4.10: Cross-sectional view of an extruded model using Gmsh's clipping planes.

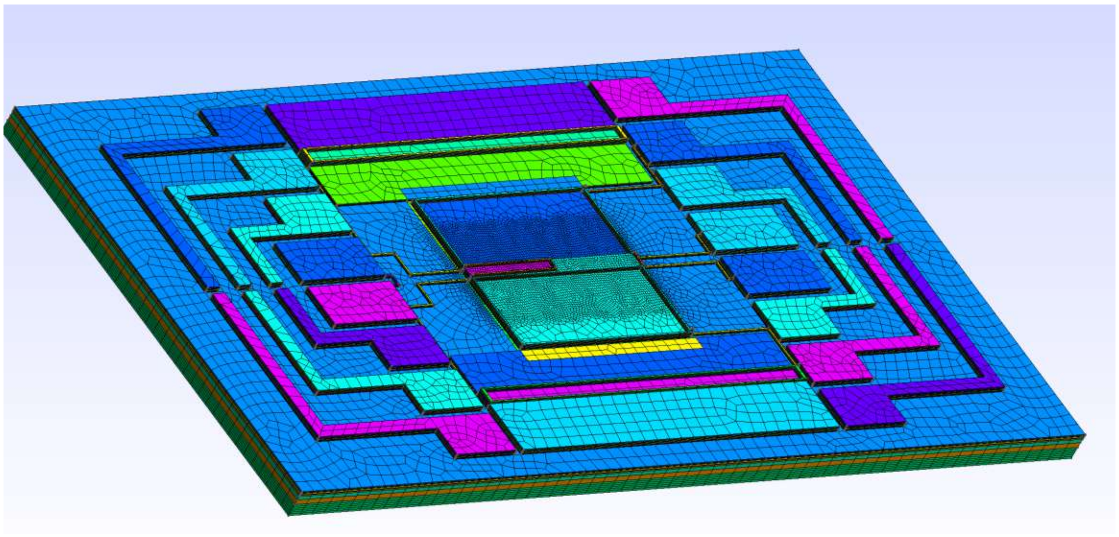


Figure 4.11: Mesh statistics: 225102 nodes, 192638 hexahedra, 24250 prisms.

4.3.3 Limitations of gds2Gmsh

Neglecting sidewall depositions

As shown on the Figure 4.13 when extruding the model we fail to represent the two small tungsten sidewall volumes indicated in Figure 4.12 due to the fact that these geometrical

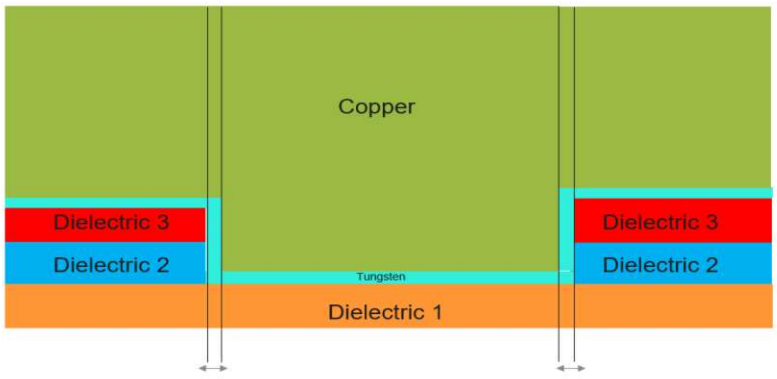


Figure 4.12: Real as-fabricated model.

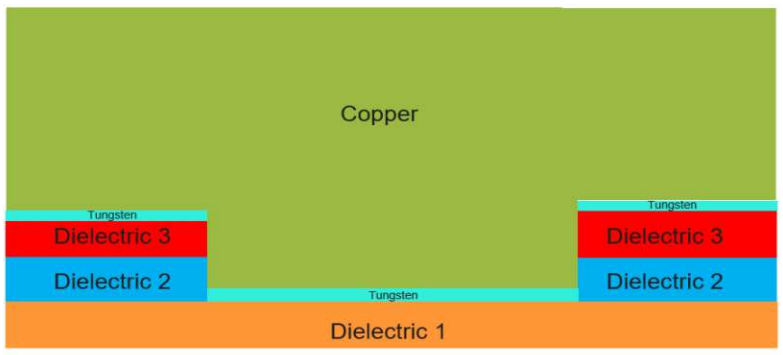


Figure 4.13: Model built using extrusion.

features are not present in the .gds layouts; they are a byproduct of a deposition process. The amount of tungsten volume which is lost and substituted with copper is fairly small. The thickness of the deposited tungsten is in the sub-micron scale, both thicknesses of dielectric 1 and dielectric 2 are in nanometer scales, and this phenomena occurs only in the regions where vias are etched. Therefore, we are not making a considerable error by neglecting the tungsten sidewall depositions.

Performance issues

When adding a new volume in the Gmsh model it is important to synchronize the geometry through the `gmsh.model.occ.synchronize()` API call in order to ensure that all the geometrical entities of the model are properly updated. This process creates the relevant Gmsh data structures, which are required for further operations. Although synchronizations can be called at any time, it is important to note that they require a considerable amount of calculations. Therefore Geuzaine in [37] mentions that it is better to minimize the number of synchronization points instead of synchronizing after every CAD command.

In our scenario, on average, a simplified layout can have between 40 to 60 polygons, its equivalent layout collapsed to 1 layer can have typically 130-150 split polygons. The material sequence described in the beginning of this chapter has 8 materials, which means 10-15 volume extrusions for each split polygon are required, due to the fact that different split polygons have different local stacks defined by the layer stacking procedure as shown schematically in Figure 4.14:

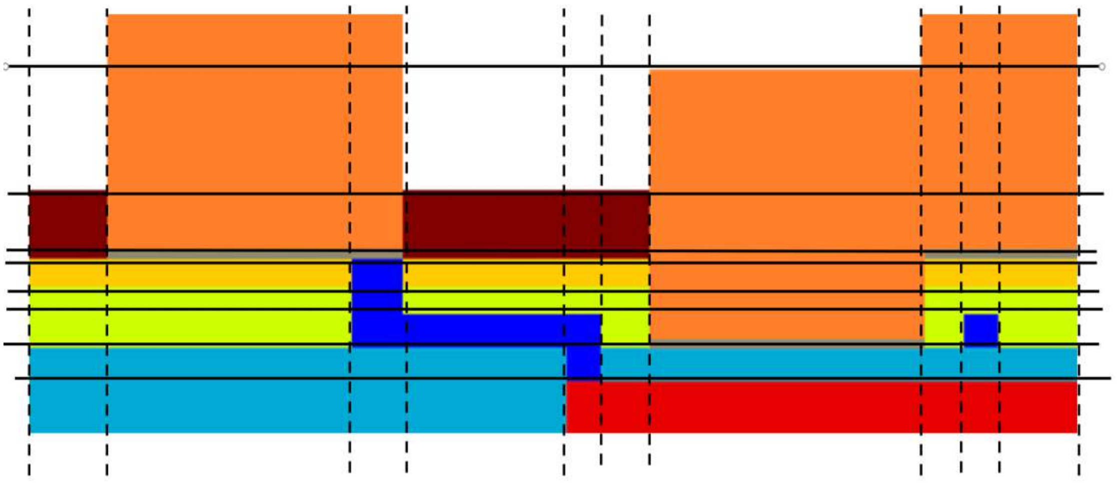


Figure 4.14: Schematic view: The dashed lines confine the local stack of each split polygon. Each local stack of a split polygon is separated by the smallest height levels of the other stacks.

In the worst case we could have 150 split polygons × 15 volume extrusions = 2250 volumes added, which means 2250 API calls of `gmsh.model.occ.synchronize()`. The more volumes are added during the process, the more time it takes to synchronize the entire model, which means that the fastest extrusion is the first split polygon and the slowest is the last split polygon. Despite these constraints, in order to fully extrude a 3D swept-meshed simplified polyheater model on average takes 15-25 minutes which is fairly acceptable taking into consideration the alternative being repetitive manual work over 1-2 days to achieve the same result.

CHAPTER 5

Results

To ensure the validity of the gds2Gmsh-generated 3D swept-meshed polyheater models, their thermal performance will be studied using the finite element method. Subsequently, in the post-processing of the simulation data, temperature values will be extracted from the regions defined by the aluminium temperature sensors in the M1 layer. This will be then compared to the experimental values of the temperature sensors obtained through the physical tests. There will be quantitative evaluations of: the impact of the simplification procedure on the results and the impact of the mesh size. Based on current limitations, further improvements will be proposed.

5.1 Simulation setup

The FEM solver which is used for this study is Ansys Mechanical APDL [22] running on a HPE C7000 cluster with the following specifications: 24-core, 48-thread Intel(R) Xeon(R) CPU E5-2690 v4 @2.60GHz processor, 512GB RAM, and 900GB local SSD storage.

Initial and boundary conditions

For the initial condition, the ambient temperature is set to 20°C. In order to define the boundary conditions, the following assumptions can be made. Due to very small time scales (heat pulses of 200µs-300µs), there is insufficient time for heat convection(requires several milliseconds) between the polyheater and the surrounding air. Regarding the heat exchange through radiation, the investigated temperatures are not high enough for radiation to be significant. Adding this to the fact that the heated metal plates on top have low emissivity, we can conclude that there is almost no heat exchange through radiation. For the conduction part, the copper pads on the top are contacted using metal needles which are used for the electric actuation of the chip, but the needles have very small contacting tips (few micrometers), so the heat conduction through them is negligibly small. Therefore, we assume adiabatic boundary conditions (no heat flux accross surface boundaries). Note that, if the heat pulses are loaded in milliseconds, the convection and conduction start to become significant, and the adiabatic boundary condition is no longer valid.

Material properties

The modelling of material properties has been performed locally by considering all the different material volumes comprising the device as isotropic materials. For each material, the related properties have been provided: Thermal conductivity in $W/(m \cdot K)$, the specific heat for the material in $J/(kg \cdot K)$ and the material density in g/cm^3. As the fabrication materials are very pure, comparable to in-literature established materials, the material properties are based on: [38], [39], [40], [41], [42].

Heat loading

In the experiment, Joule heating is initiated by applying a rectangular voltage pulse to the device, Figure 5.1. The power is measured, and is used to define the heat generation loading in the polysilicon plate. In the heat equation this is represented by the forcing term $g(x, y, z, t)$, mentioned in section 2.1, which in this case is a heat source.

The thermal load is such, that the temperature continually rises as voltage is applied. Following the end of the pulse, a sharp drop in temperature is observed, which is mainly governed by heat flow into the copper metallization and the silicon substrate. Since the main goal of the thermal simulation is to determine the temperature rise in the metallization, the thermal problem is typically only solved for a duration of 260µs in time steps of 4µs.

5.2 Verification of models with different types of heating

The following validation will be performed on two models which have homogenous and inhomogenous types of heating. This means that their polysilicon plate, or their `poly` layer, are as shown in Figure 5.2a, and Figure 5.2b. These polyheater types can be used for example to study copper degradation under high temperature load conditions [15]. For the generation of Gmsh models, in both cases, a rectangular mesh size field which is

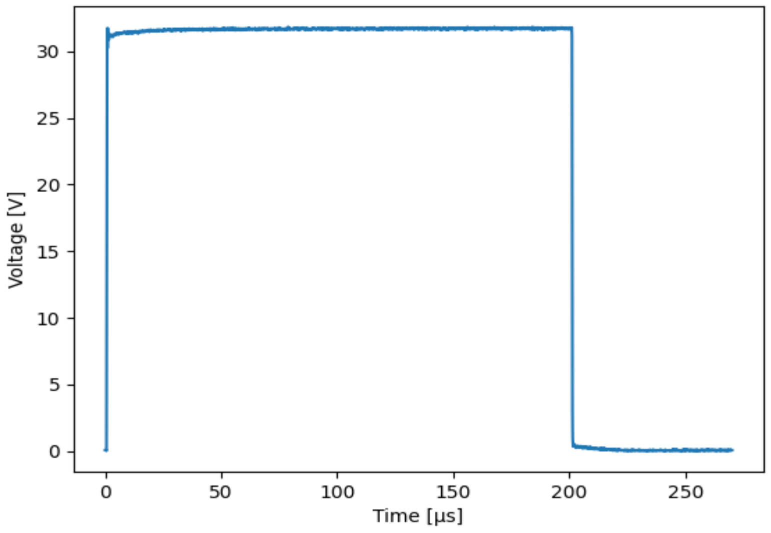
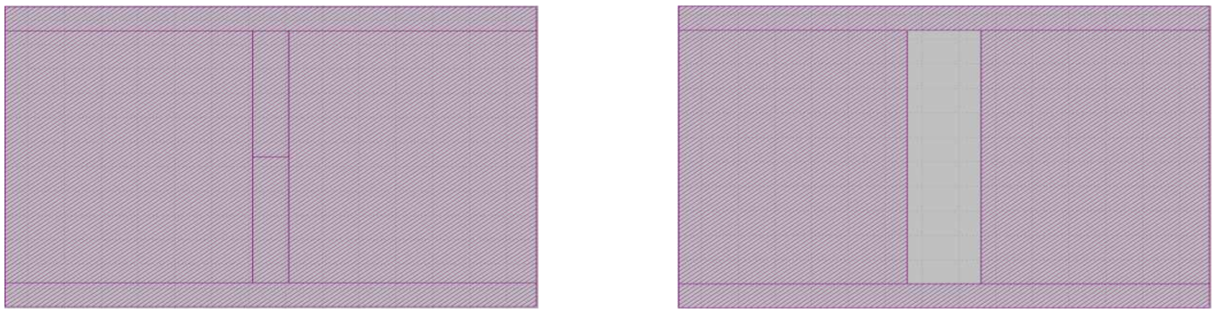


Figure 5.1: Rectangular voltage pulse that initiates the Joule heating in the device.

adapted to the polysilicon plate is chosen, with element size (diagonal length of element's bounding box) being 5µm inside the rectangle, and 50µm outside of it.



(a) Homogenous polysilicon plate.

(b) Inhomogenous polysilicon plate.

Figure 5.2: Design of polysilicon plates for different types of heating.

Due to Ansys Mechanical APDL having post-processing and data visualisation capabilities which are not ideal for data-centric studies, the following Python modules were used: PyVista [43], and PyAnsys [44].

Figure 5.3 and, Figure 5.4 show the intended effect of the polysilicon layer designs, which is heating the copper plate in different ways. The underlying homogenous polysilicon plates heat the main copper plate on top in different ways, depending on their geometry.

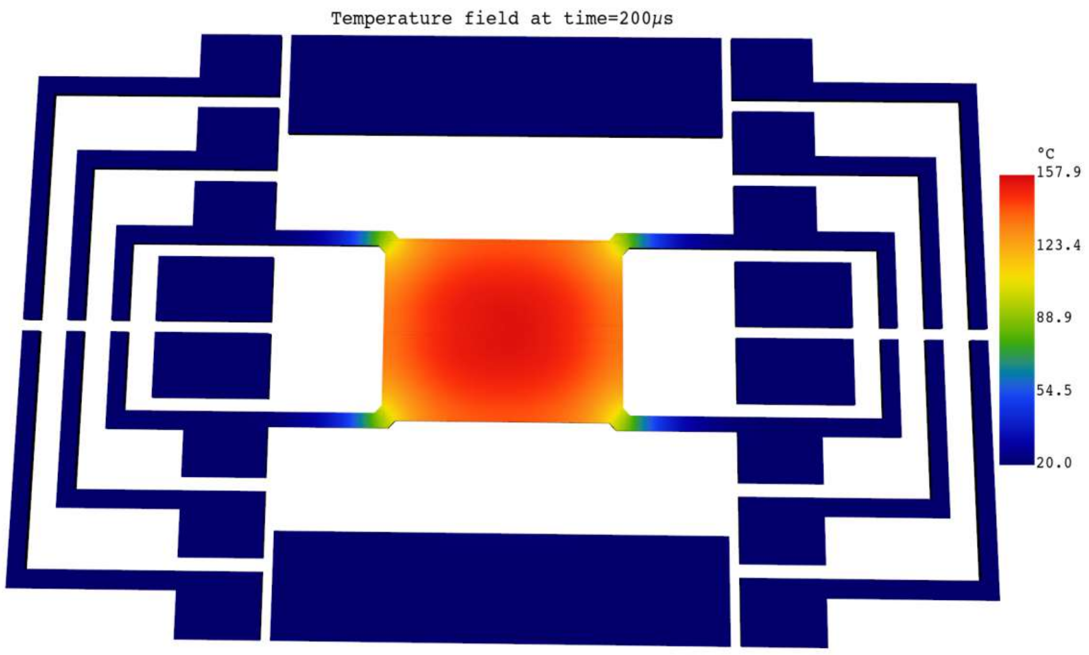


Figure 5.3: Homogenous heating of the central Cu plate.

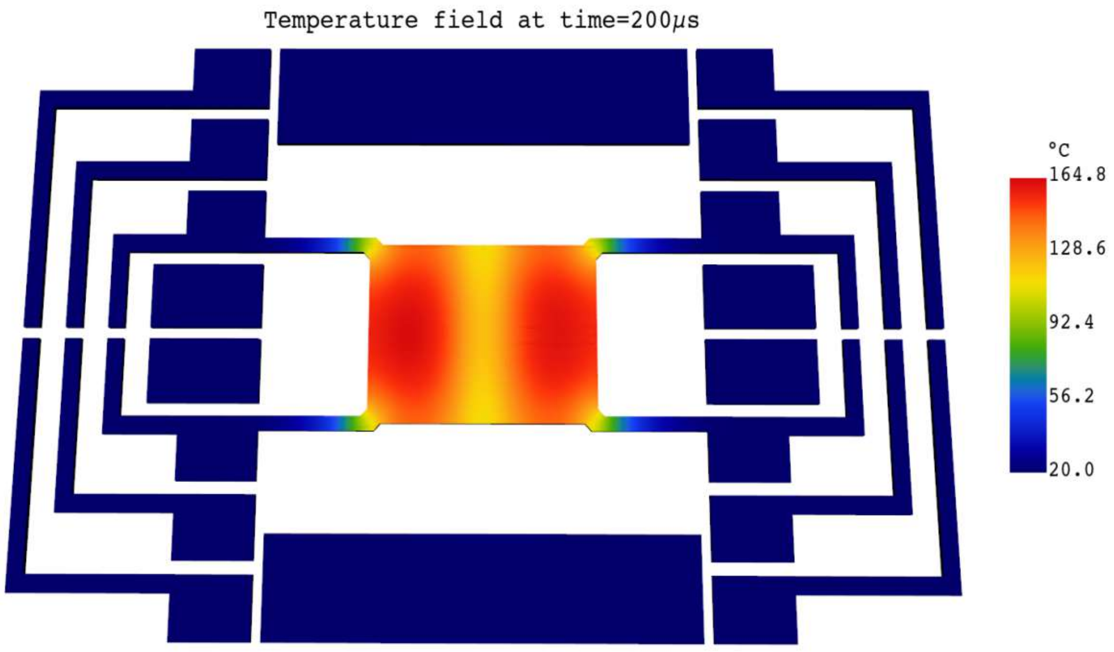


Figure 5.4: Inhomogenous heating of the central Cu plate.

In Figure 5.4 the homogenous polysilicon plate performs homogenous heating to the main copper plate, while in Figure 5.4 the polysilicon plate with a central cavity in its geometry performs inhomogenous heating to the main copper plate, defined by the position of the cavity. In Figure 5.5 this is illustrated by plotting the temperature, and its gradient along a horizontal line that goes through the middle of the central copper plate in its y and z coordinates. This provides an initial validation, that our gds2Gmsh models exhibit the expected behaviour, when subjected to such a simple comparison.

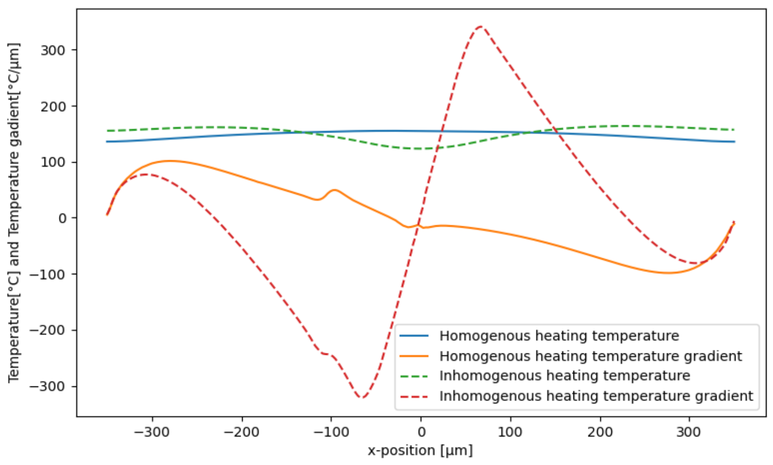


Figure 5.5: Temperature and thermal gradient along a 700µm line, which stretches through the entire width of the centrally heated copper plate.

Concerning the numerical results, the following procedure is undertaken in all presented simulations. The comparison is performed against experimental data, taken from the aluminium temperature sensors with a measurement device which provides 3000 temperature values along the time interval specified in Section 5.1. For visualisation, this data has been smoothed and reduced in size using a moving median filter. The corresponding temperature values in the simulations have been taken as the average element temperature in the sensor region.

The numerical results corresponding to the different types of heating discussed here, are presented in Figure 5.6 and Figure 5.7. Overall, the simulation results are in excellent agreement with the experimental values; this is partially due to choosing a small element size for the region overlapping with the polysilicon plate.

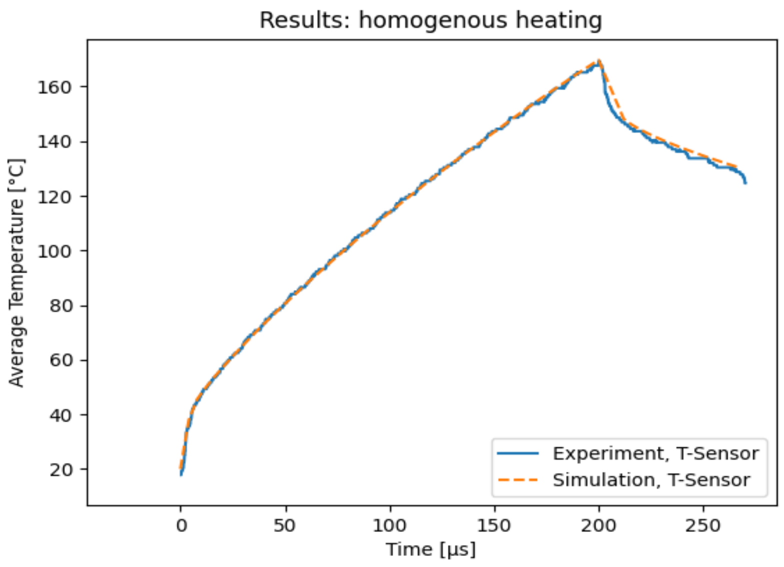


Figure 5.6: Excellent agreement on the average temperature between simulation and experiment for the homogenous heating case

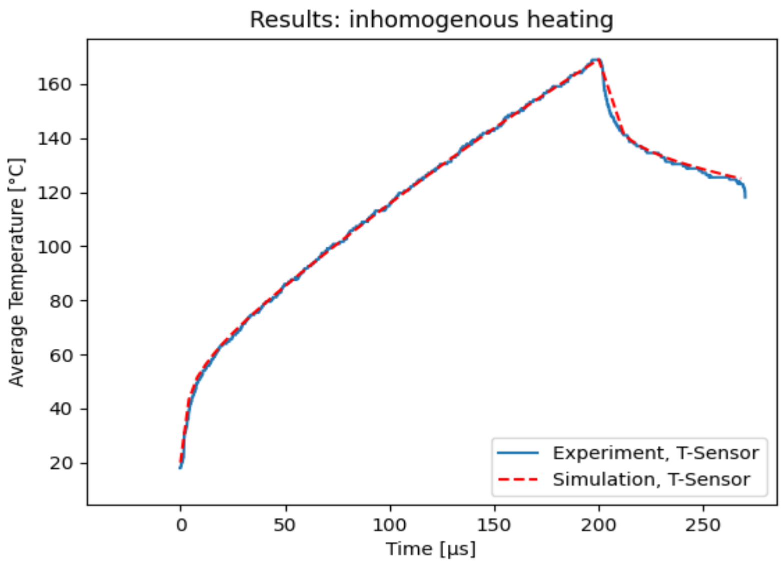


Figure 5.7: Excellent agreement on the average temperature between simulation and experiment for the inhomogenous heating case

5.3 Mesh size study

In the following case-study, we consider half-volume gds2Gmsh models of a polyheater model which exhibits double horizontal symmetry for computational efficiency. The simplification routine applied on this particular model required 0.37 seconds for execution. In Gmsh, a rectangular adaptive mesh size was applied, covering entirely the area which is 20µm and 10µm longer in the x and y directions, respectively, than the polysilicon plate's lengths. Inside this mesh size field, three different element values are chosen: For the coarse mesh, the element size is 10µm, for the medium mesh the element size is 5µm, and for the fine mesh the element size is 2.5µm. For all three different meshes, the element size outside of the defined rectangular mesh size field is chosen to be 50µm due to this area not being affected by central heating. In Gmsh, for the 2D meshing, the Frontal-Delaunay algorithm [45] is chosen, and for the 2D recombination, the Blossom algorithm [46] is used. The different meshes are shown in Figure 5.8, Figure 5.9, and Figure 5.10.

5.3.1 Mesh quality parameters in Gmsh

To assess the mesh quality, Gmsh offers the functionality to plot for each element: γ, signed inverse condition number (SICN), and signed inverse gradient error (SIGE) parameters.

The parameter γ measures a type of volume-to-edge length and area aspect ratio given by the formula:

$$\gamma = \frac{V}{|e_{max}|(\sum_{i=1}^{n} A_i)} \tag{5.1}$$

Here V is the volume, e_{max} is the length of the longest edge, and each A_i corresponds to the area of each face of the element. Due to the fact that we expect uniform heating with no sudden changes, this parameter has no significant effect on the mesh quality.

SICN is inversely proportional to the condition number of the linear transformation matrix S between an element of the mesh and a unit-length element, given by $\kappa(S)$. Smaller values of the SICN parameter indicate well conditioned transformation matrices S, which produce non-heavily distorted elements and high quality meshes.

SIGE measures how well a discrete element is able to estimate the value of the gradient of a given field. High values of this parameter indicate low errors in the field gradient calculations.

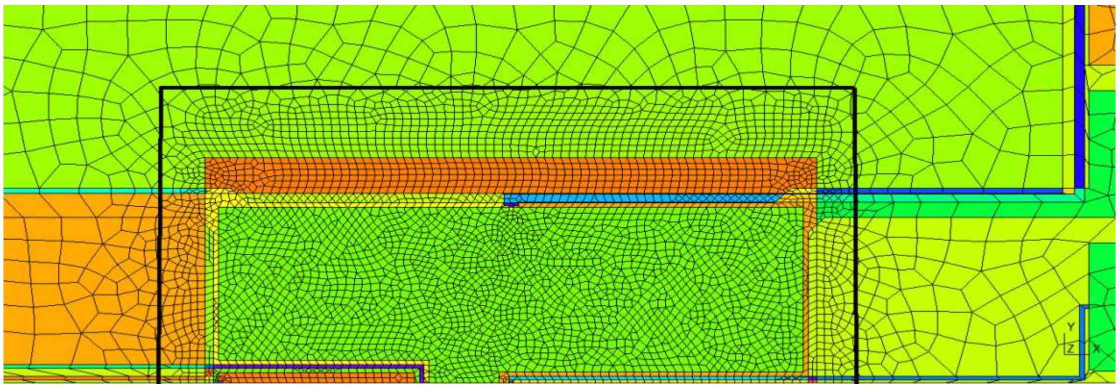


Figure 5.8: Fraction of the coarse mesh zoomed-in close to the mesh size field. Statistics: 145242 nodes, 122411 hexahedra, 19456 prisms.

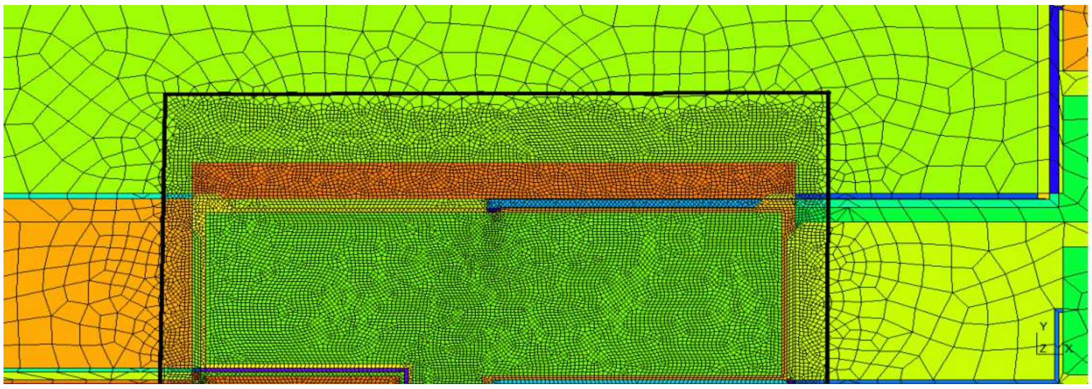


Figure 5.9: Fraction of the medium mesh zoomed-in close to the mesh size field. Statistics: 396122 nodes, 360362 hexahedra, 17333 prisms.

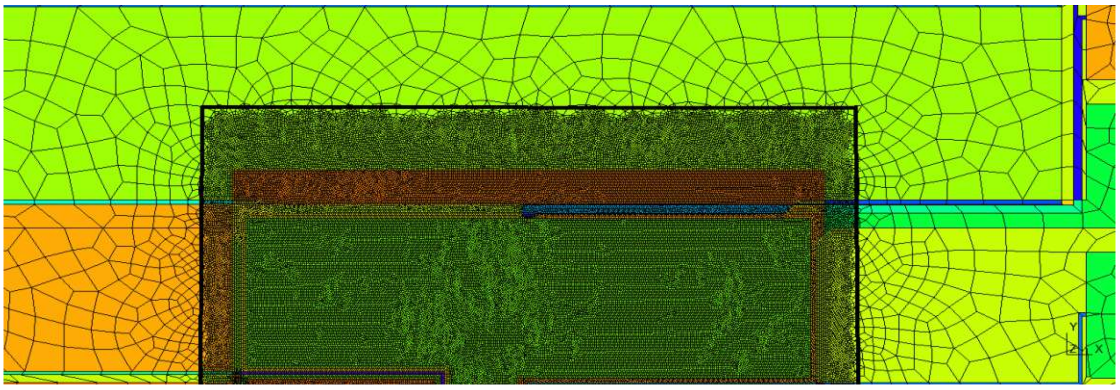


Figure 5.10: Fraction of the fine mesh zoomed-in close to the mesh size field. Statistics: 1317310 nodes, 1229480 hexahedra, 28232 prisms.

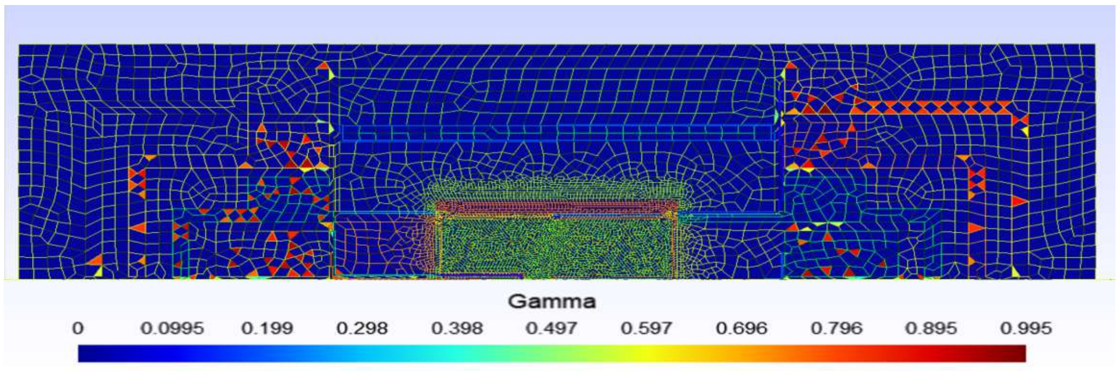


Figure 5.11: Plot of γ mesh-quality parameter.

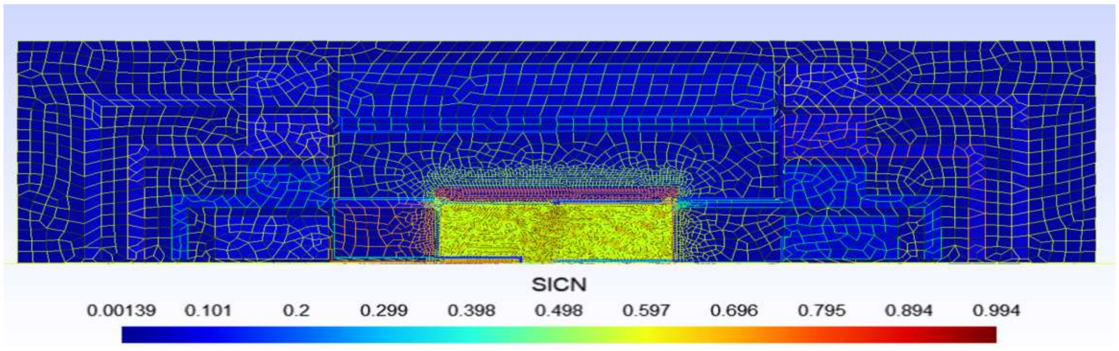


Figure 5.12: Plot of SICN mesh-quality parameter.

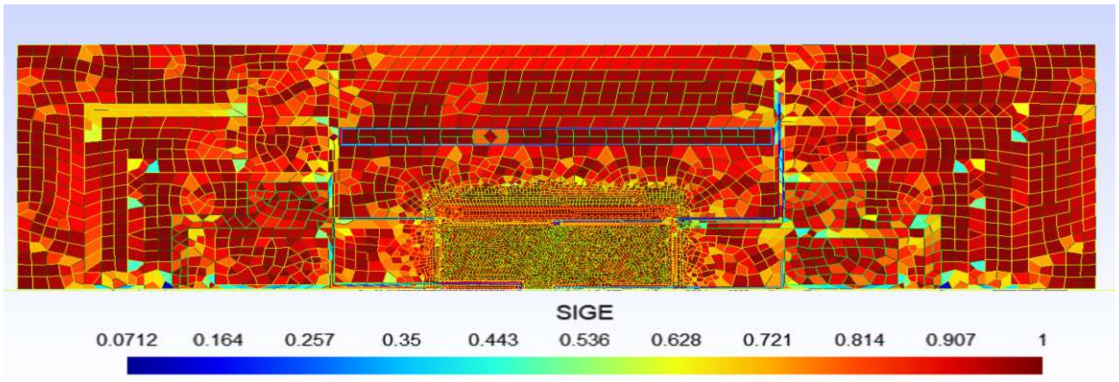


Figure 5.13: Plot of SIGE mesh-quality parameter.

In Figure 5.11, Figure 5.12, and Figure 5.13 these Gmsh-provided mesh quality parameters are visualised for the coarse mesh. From the discussion above, it can be noted that even for a coarse mesh, the near-ideal values for SICN and SIGE indicate a good quality mesh. This fact can be attributed to the use of Frontal-Delaunay algorithm, which produces the higher quality meshes, compared to the other Gmsh meshing algorithms [47].

5.3.2 Numerical results

The simulation results of the thermal response in the three different meshes discussed in Section 5.3 are plotted against the experimental values of the same single temperature sensor below.

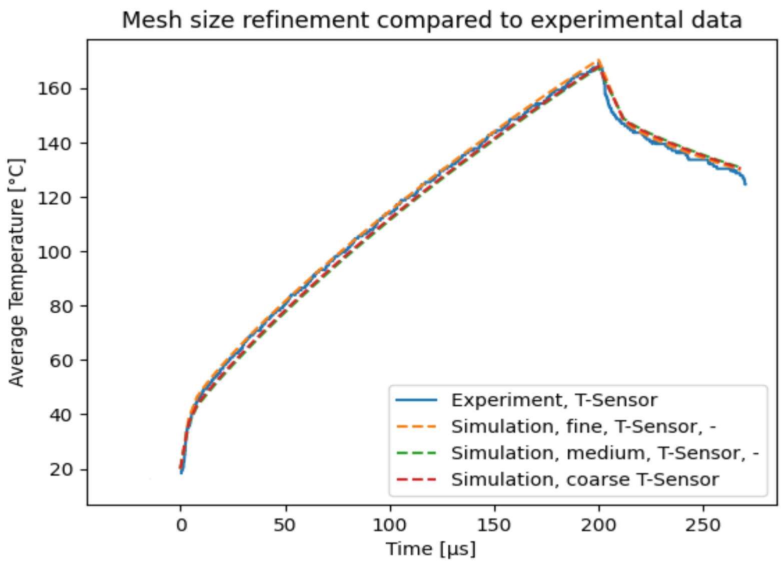


Figure 5.14: Heating from room temperature up to 160°C and cooling down.

The half-model considered in Section 5.3, contains only one temperature sensor, and its experimental and simulated thermal response are plotted in Figure 5.14 and Figure 5.15. The model generation through gds2Gmsh was conducted in a personal computer with the following specifications: 11th Gen Intel(R) Core(TM) i5-1145G7 @2.60GHz, 16GB RAM. The solving was performed through Ansys Mechanical APDL in the cluster mentioned in Section 5.1, using 12 cores. Table 5.1 gives the time performance of these processes with respect to mesh size:

From Figure 5.14 and Figure 5.15, it can be concluded that there is good agreement between the simulation results and the experimental data, even for the coarse mesh. The

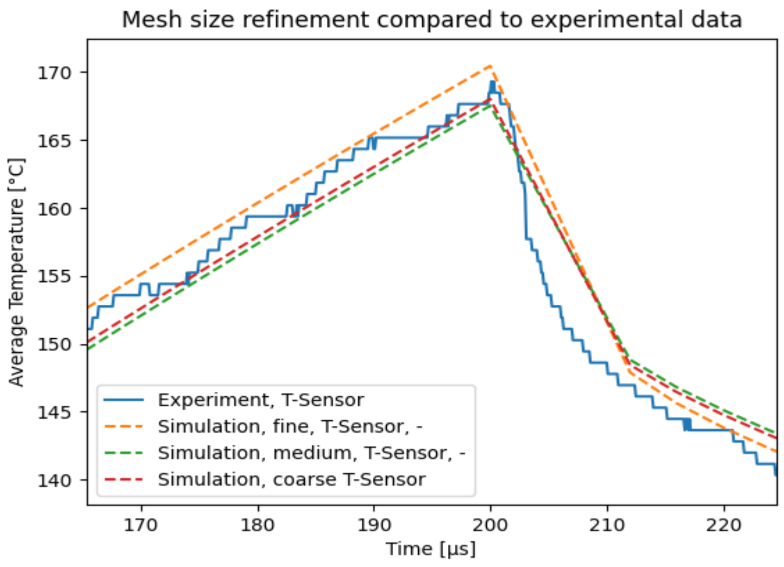


Figure 5.15: Peak temperature values.

Mesh size	Model generation	Solving
coarse	00:16:13	00:11:03
medium	00:16:57	00:35:22
fine	00:18:03	01:44:17

Table 5.1: Runtime performance of gds2Gmsh model generation on a personal computer and solving on the cluster.

size of the mesh does not have an impact on the model generation, since the performance depends on the number of geometry synchronizations per each added volume (as discussed in Section 4.3.3), and there is the same number of solid volumes being extruded per each mesh. The small difference during the model generation can be attributed to the different element sizes in the intial 2D meshing at $z = 0$. Note that, in real use cases, the model generation time is not of particular concern, because it is a one-time effort to create a mesh for a given model.Due to the significant improvement in the solving runtime, the coarse mesh should be usually chosen, since it reproduces the experimental results fairly well, having just 2°C difference at their peak temperature values of 170°C, meaning an error of just above 1%, as shown in Figure 5.15.

5.4 Simplification study

Recalling the simplification procedure from Chapter 3, we show that the 2D simplifications performed over the filtered .gds layouts generate insignificant errors for our thermal simulations. We consider again a model with double horizontal symmetry, which has three different temperature sensors, as previously shown in Figure 3.4. For the unsimplified variant, we consider a gds2Gmsh model, extruded from an unsimplified, filtered .gds layout, with adaptive rectangular mesh size fields, covering the polysilicon plate. Two of the temperature sensors that overlap with the polysilicon plate are refined by the rectangular mesh size field (Figure 5.16a). For the third temperature sensor that does not overlap with the polysilicon plate, a second rectangular mesh size field is chosen with the same refinement size. In the simplified variant, the simplifications allow for a coarser mesh to be used with a reasonable element quality, as illustrated in Figure 5.16b.

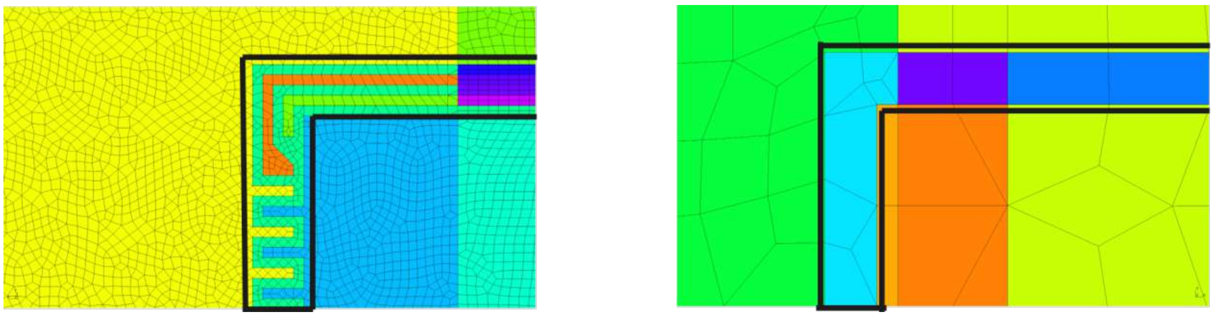

(a) Unsimplified temperature sensor. **(b)** Simplified temperature sensor.

Figure 5.16: Mesh detail close to one of the temperature sensors, highlighted with the thicker black lines.

Numerical results for the 3 temperature sensors

By using 24 cores in the cluster mentioned in Section 5.1, the runtime required to solve the unsimplified mesh was 3:47:53, while the simplified version, was executed in 00:06:27. Despite choosing the same mesh adaptivity fields with the same element sizes, the Frontal-Delaunay algorithm had to perform some self-refinement in the geometrical details, which are present in layers other than M1, in order to produce high quality elements.

For the studied polyheater model, the numerical results can be interpreted by recalling the position of the temperature sensors related to the polysilicon plate, as was shown in Figure 3.4. Temperature sensors 1 and 2 experience the most significant increase in temperature, since they overlap with the heating polysilicon plate. For temperature sensor 1, the results of the simplified and unsimplified variants both replicate the experimental

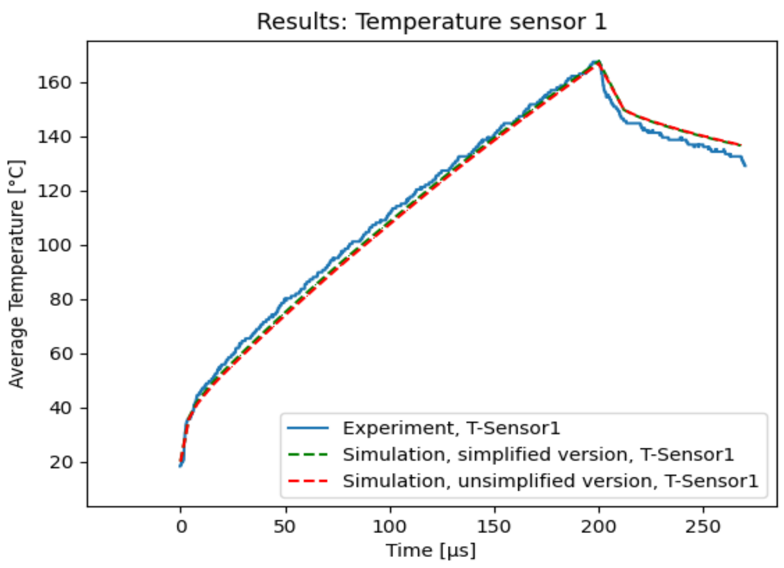


Figure 5.17: Due to its central position, temperature sensor 1 experiences the highest temperatures.

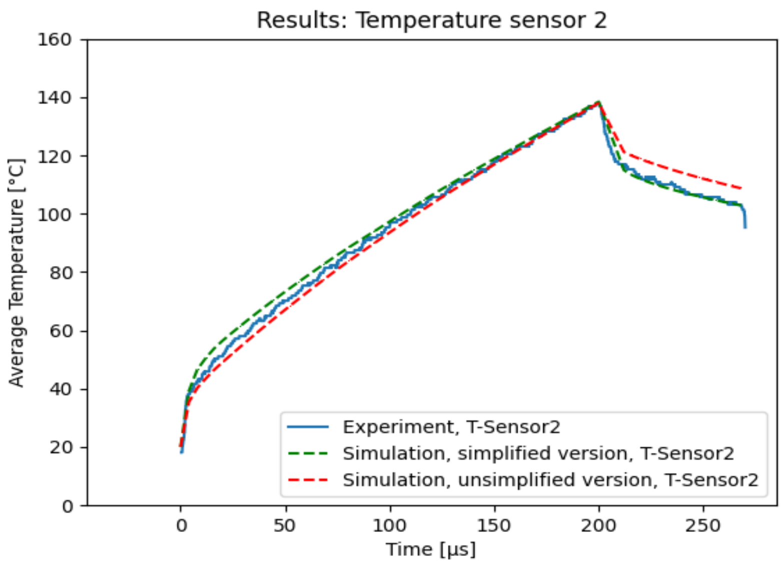


Figure 5.18: Since temperature sensor 2 is on the edge of the heating polysilicon plate, it shows lower temperature values compared to the highest value.

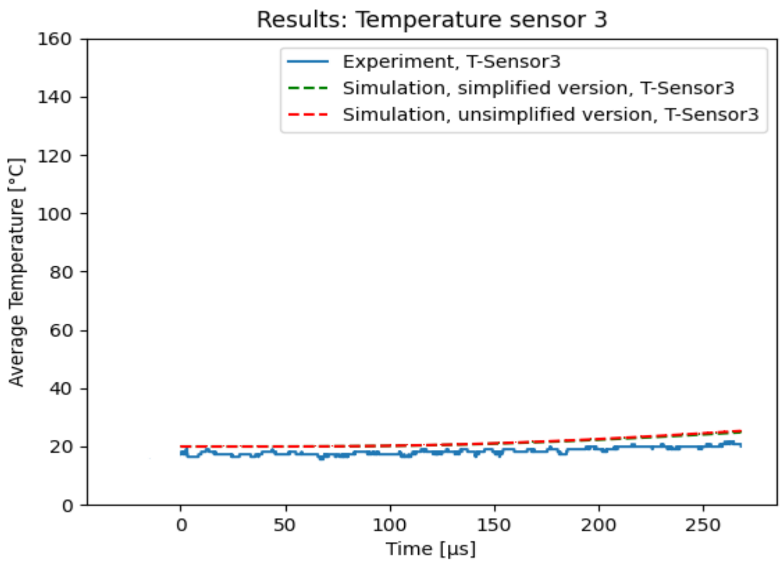


Figure 5.19: Temperature sensor 3 is outside of the polysilicon plate, therefore the temperature is quite low and it does not display major changes in temperature.

values very well. For temperature sensor 2, which is located near the edge of the heated region, we observe a small difference between the two variants in the initial heating phase, as well as the cooling phase. These differences may be explained by the fact that the true topography near the copper edge is simplified in the extrusion model, which impacts a detailed and a smeared sensor region differently. For temperature sensor 3, there is no significant change in the temperature values, since it is not in direct contact with the polysilicon plate. Owing to the fact that the time scales are very small, there is not enough time for heat to propagate and reach the level of temperature sensor 3. All the small errors present in the results of the 3 temperature sensors, are cumulatively due to the 2D geometry simplifications made in Chapter 3, and due to the sidewall neglections mentioned in Chapter 4.

CHAPTER 6

Summary and outlook

In this work, the main goal was to provide a fully automated framework for the pre-processing routine in thermal finite element analysis of polyheater devices. This was achieved in multiple stages and the developed approach requires minimal user input. The incentive was to ensure that users do not have to draw the model geometries from scratch in order to perform a simulation analysis, but to use available microfabrication design data. This design data was extensive in details which have little or no impact on the thermal finite element analysis. A geometry defeaturing/simplification routine, motivated by recent methodologies in TCAD, was devised in an automated manner. The reasoning behind these simplifications was based on the device design and the physics behind the problem. The algorithm was scaled to work on a family of models, with unique representatives sharing some small class similarities.

The second major task was to create 3D meshed CAD models, from these simplified layouts, which are ready to be applied to thermal finite element analysis. At first, surface topographies of polyheater fabricated models were considered by employing an open-source process simulator (ViennaPS) to create realistic 3D CAD models. Due to its limitations in efficiently meshing thin regions, this tool was not suitable for the task. Therefore, simple extrusion was considered, by using the open-source Gmsh meshing software, with very powerful geometry pre-processing capabilities. A fully automated procedure starting from a simplified .gds layout, and concluding with a 3D swept-meshed CAD model with the correct material metadata, ready for thermal FEA, was achieved through gds2Gmsh.

The 3D CAD models, generated through our automated simplification, extrusion, and meshing were validated in thermal FEA against available experimental data. A good agreement was demonstrated between the two. This was shown through simulations of models with different types of heating, a mesh size study and a simplification study. When studying the mesh size, it was evident that, even a coarse mesh, like the one described in Chapter 5, produced approximately the same temperature values as the ones obtained experimentally. In the simplification study, we show the quantitative difference between the results produced using an unsimplified layout and those from a simplified layout. The differences were about 1%, confirming that the simplification is suitable, and both variants agree well with the experimental data. It should be noted that the simplified was much more computationally efficient to solve; up to 35 times faster than the unsimplified model.

With the developed automated framework, one can generate a 3D mesh of any polyheater model to perform further investigations in thermal stresses, thermo-mechanical fatigue of power metallizations, delamination phenomena, chip bowing, etc. It is also important to note that, gds2Gmsh can be used to generate a 3D model by supplying the extrusion technology for any .gds file. This framework can produce high quality meshes for thin geometries, and it can be used for MEMS geometries as well.

A further improvement to gds2Gmsh could be to consider the irregularities coming from real fabrication processes. This would create even more realistic 3D models than by applying simple extrusions. Nonetheless, this is not a trivial task. While there are several libraries available to perform process simulation, they have their own meshing engines, which are specialised for their purpose. Combining Gmsh's powerful pre-processing and meshing capabilities with process simulation libraries could lead to better CAD models for various finite element analyses in the future.

List of Figures

List of Tables

List of Algorithms

Acronyms

2D	2-Dimensional. 17
3D	3-Dimensional. 18
APDL	Ansys Parametric Design Language. 20
API	Application Programming Interface. 36
CAD	Computer Aided Design. 15
CPU	Central Processing Unit. 46
CTE	Coefficient of Thermal Expansion. 2
EDA	Electronic Design Automation. 15
FEA	Finite Element Analysis. 21
FEM	Finite Element Method. 6
GDS	Graphic Design System. v
GUI	Graphical User Interface. 17
IC	Integrated Circuit. 15
MEMS	Micro-electromechanical systems. 17
PCM	Process Control Monitoring. 24
RAM	Random Access Memory. 46

SICN	Signed Inverse Condition Number. 52
SIGE	Signed Inverse Gradient Error. 52
SSD	Solid State Drive. 46
TCAD	Technology Computer-Aided Design. 19

Nomenclature

R:	Electrical resistance
$\alpha_{TCR,T_{ref}}$:	Temperature coefficient of resistance
T:	Calibration temperature
q:	Local heat flux
ρ:	Material density
k:	Scalar thermal conductivity
$\mathbf{k}$:	Tensorial thermal conductivity
ϵ:	Surface emissivity
σ:	Stefan-Boltzmann constant
$a(\cdot,\cdot)$:	Stefan-Boltzmann constant
$(\cdot,\cdot)_{L^2(\Omega)}$:	Inner product in $L^2(\Omega)$
$\overline{\Omega}$:	Closure of the set Ω
$\partial\Omega$:	Boundary of the set Ω
c:	Specific heat capacity
C:	Heat capacity

Bibliography

[1] Sebastian Moser. *Thermo-mechanical fatigue of metallizations in microelectronic applications.* PhD thesis, Montan Universität Leoben, September 2022.

[2] Paul Ullmann. Method for real-time observation of thermo-mechanical induced fatigue in copper thin films. Master's thesis, TU Wien, January 2017.

[3] Harry A. Schafft and John S. Suehle. The measurement, use and interpretation of the temperature coefficient of resistance of metallizations. *Solid-State Electronics*, 35(3):403–410, 1992.

[4] Xue-Shuo Shang, Qing-Wen Li, Qun Cao, Zi-Rui Li, Wei Shao, and Zheng Cui. Mathematical modeling and multi-objective optimization on the rectangular micro-channel heat sink. *International Journal of Thermal Sciences*, 184:107926, 2023.

[5] Dawei Zhuang, Yifei Yang, Guoliang Ding, Xinyuan Du, and Zuntao Hu. Optimization of microchannel heat sink with rhombus fractal-like units for electronic chip cooling. *International Journal of Refrigeration*, 116:108–118, 2020.

[6] Altayyeb A. Alfaryjat, Hussein A. Mohammed, Nor Mariah Adam, Mohd Khairol Anuar Ariffin, and Mohammad Izadi Najafabadi. Influence of geometrical parameters of hexagonal, circular, and rhombus microchannel heat sinks on the thermohydraulic characteristics. *International Communications in Heat and Mass Transfer*, 52:121–131, 2014.

[7] Dale W. Thoe Eleftherios C. Zachmanoglou. *Introduction to Partial Differential Equations with Applications.* Dover Books on Mathematics, 1987.

[8] Raffaele Coppeta, Ayoub Lahlalia, Darjan Kozic, René Hammer, Johann Dr. Riedler, Gregor Toschkoff, Anderson Pires Singulani, Zeeshan Ali, Martin Sagmeister, Sara Carniello, Siegfried Selberherr, and Lado Filipovic. Electro-thermal-mechanical modeling of gas sensor hotplates. *Sensor Systems Simulations*, 2:17–72, 2019.

[9] John Conrad Jaeger Horatio Scott Carslaw. *Conduction of heat in solids.* Oxford University Press, 1959.

[10] M. Necati Özışık. *Heat Transfer: A Basic Approach.* McGraw-Hill, 1985.

[11] Philippe G. Ciarlet. *The Finite Element Method for Elliptic Problems.* SIAM, 1978.

[12] Steve Di Bartolomeo. All about Calma's GDSII stream file format. https://www.artwork.com/gdsii/gdsii/, 2011. [Online; accessed 19.04.2023].

[13] Rainer Minixhofer. *Integrating technology simulation into the semiconductor manufacturing environment.* PhD thesis, TU Wien, 2006.

[14] Matthias Koefferlein. Klayout. https://www.klayout.de/, 2019-2020. [Online; accessed 20.04.2023].

[15] Paul Hoffmann, Sebastian Moser, Corinna Kofler, Michael Nelhiebel, Daniel Tscharnuter, Balamurugan Karunamurthy, Heinz E. Pettermann, and Melanie Todt. Thermomechanical fatigue damage modeling and material parameter calibration for thin film metallizations. *International Journal of Fatigue*, 155:106627, 2022.

[16] Integrated Circuit Design Group University of Twente. Gds3d - an application used for rendering ic (chip) layouts in 3d. https://github.com/trilomix/GDS3D. [Online; accessed 27.04.2023].

[17] Roger Light. gds2pov 20080229. https://www.atchoo.org/gds2pov/. [Online; accessed 27.04.2023].

[18] Thomas Kramer. gds3xtrude. https://codeberg.org/tok/gds3xtrude. [Online; accessed 27.04.2023].

[19] OpenSCAD Project. Openscad. https://openscad.org/. [Online; accessed 27.04.2023].

[20] Cogenda. Gds2mesh. https://www.cogenda.com/article/Gds2Mesh. [Online; accessed 27.04.2023].

[21] COMSOL Multiphysics®. ECAD import module. https://www.comsol.com/ecad-import-module. [Online; accessed 21.04.2023].

[22] Ansys Mechanical. Finite element analysis (FEA) software for structural engineering. https://www.ansys.com/products/structures/ansys-mechanical. [Online; accessed 21.04.2023].

[23] Coventor. SEMulator3D. https://www.coventor.com/products/semulator3d/. [Online; accessed 03.08.2023].

[24] Lado Filipovic et al. ViennaPS. https://github.com/ViennaTools/ViennaPS. [Online; accessed 15.05.2023].

[25] VTK Project. VTK, visualization toolkit. https://vtk.org/. [Online; accessed 15.05.2023].

[26] Atul Thakur, Ashis Gopal Banerjee, and Satyandra K. Gupta. A survey of CAD model simplification techniques for physics-based simulation applications. *Computer-Aided Design*, 41(2):65–80, 2009.

[27] Yong-Gu Lee and Kunwoo Lee. Geometric detail suppression by the fourier transform. *Computer-Aided Design*, 30(9):677–693, 1998.

[28] Shawn Martin Steve Owen, Timothy M. Shead. CAD defeaturing using machine learning. *Sandia National Laboratories*, 2019.

[29] Lucas H. Gabrielli. gdspy. https://gdspy.readthedocs.io/en/stable/. [Online; accessed 10.04.2023].

[30] Sean Gillies. Shapely. https://shapely.readthedocs.io/en/stable/manual.html. [Online; accessed 10.04.2023].

[31] GEOS Project. GEOS. https://libgeos.org/. [Online; accessed 10.04.2023].

[32] Jean Serra. Image analysis and mathematical morphology. *Cytometry*, 4:184–185, 09 1983.

[33] Gmsh Project. Gmsh. https://gmsh.info/. [Online; accessed 15.05.2023].

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

[35] COMSOL Multiphysics. Fundamentals of swept meshing. https://www.comsol.de/support/learning-center/article/Introduction-to-Swept-Meshing-51861/152#good-mesh. [Online; accessed 27.07.2023].

[36] OpenCASCADE Project. OpenCASCADE. https://dev.opencascade.org/. [Online; accessed 01.08.2023].

[37] Christophe Geuzaine. Gmsh tutorial 1. https://gitlab.onelab.info/gmsh/gmsh/blob/master/tutorials/python/t1.py#L88. [Online; accessed 03.08.2023].

[38] Robert Hull. *Properties of Crystalline Silicon.* INSPEC, The Institution of Electrical Engineers, 1999.

[39] Lado Filipovic. *Topography simulation of novel processing techniques.* PhD thesis, TU Wien, 2012.

[40] John Harb Christian Lott, Tim McLain and Larry Howell. Modeling the thermal behavior of a surface-micromachined linear-displacement thermomechanical microactuator. *Sensors and Actuators A: Physical*, 101(1):239–250, 2002.

[41] R.Byron Bird, Warren E. Stewart, and Edwin N. Lightfoot. *Transport Phenomena.* Number Bd. 1 in Transport Phenomena. Wiley, 2006.

[42] Jerome G. Hust and Alan B. Lankford. Thermal conductivity of aluminum, copper, iron, and tungsten for temperatures from 1 K to the melting point. *National Bureau of Standards*, 1984.

[43] PyVista. PyVista. https://docs.pyvista.org/version/stable/. [Online; accessed 01.10.2023].

[44] Ansys. PyAnsys. https://docs.pyansys.com/version/stable/#. [Online; accessed 01.10.2023].

[45] Jean-François Remacle, François Henrotte, T. Baudouin, Christophe Geuzaine, E. Béchet, Thibaud Mouton, and Emilie Marchandise. A frontal Delaunay quad mesh generator using the l-infinity norm. *International Journal for Numerical Methods in Engineering*, 94:455 – 472, 2013.

[46] Jean-François Remacle, Jonathan Lambrechts, Bruno Seny, Emilie Marchandise, Amaury Johnen, and Christophe Geuzaine. Blossom-quad: A non-uniform quadrilateral mesh generator using a minimum-cost perfect-matching algorithm. *International Journal for Numerical Methods in Engineering*, 89:1102 – 1119, 02 2012.

[47] Christopher Geuzaine. Tutorial 10. https://gitlab.onelab.info/gmsh/gmsh/blob/gmsh_4_11_1/tutorials/python/t10.py#L136. [Online; accessed 30.07.2023].