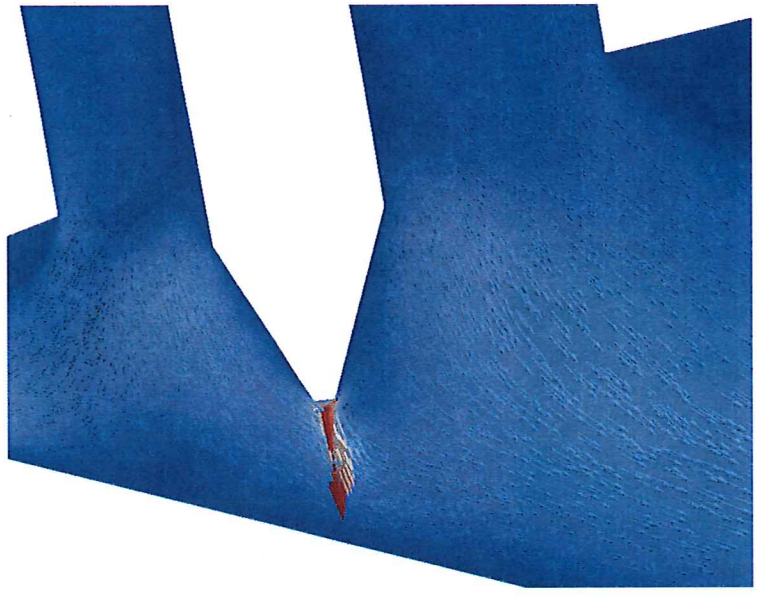
UNIVERSITY WEST

June 24, 2013

Study of the influence of the electrode tilt angle in GTAW doing CFD simulation of the heat source

Johanna Matsfelt



BACHELOR'S THESIS
Mechanical Engineering, Product development and design
Department of Engineering Science, University West

BACHELOR'S THESIS

Study of the influence of the electrode tilt angle in GTAW doing CFD simulation of the heat source

Summary

Gas Tungsten Arc Welding (GTAW) is often used in production tilting the electrode by a few degrees. However, when GTAW is studied using Computational Fluid Dynamics (CFD) the electrode uses to be assumed perpendicular to the base metal (i.e. zero tilt). This bachelor thesis aimed at investigating if a perpendicular electrode is a valid approximation for the CFD simulation of GTAW process with 12° electrode tilt. No earlier simulation study with a similar aim is known.

The project started from a simplified problem with electromagnetism (and no fluid flow) in an infinite electrically conducting rod. This problem has a known analytical solution that was used for validating the simulation results. A 3D mesh was developed for this test case so as to reproduce the analytic solution with good accuracy. This mesh was then used as base for building the 3D mesh of the GTAW problem.

The model describing the physics of GTAW combines the electromagnetic model with a thermal fluid model. The 3D mesh was reduced to half space because of symmetry along the welding path. After memory related problems the number of cells of the mesh had to be further reduced. Two test cases that only differ by the electrode tip angle (0° and 12° angle) were simulated. The simulation results show that on the top surface of the base metal the shape of the heat affected zone is narrower behind the electrode and wider in front of the electrode for a 12° tilt compared to a 0° tilt. The heat distribution on the base metal is thus influenced if the tilt angle is 12°. This shows that a perpendicular electrode is not a valid approximation for the CFD simulation of GTAW process with a 12° electrode tilt.

Date:	June 24, 2013
Author:	Johanna Matsfelt
Examiner:	Lars-Erik Svensson, University West
Advisor:	Isabelle Choquet, University West
Programme:	Product development and design
Main field of study:	Mechanical Engineering **Education level:** first cycle
Credits:	15 HE credits
Keywords	GTAW, magnetohydrodynamic, mesh development, OpenFOAM CFD simulation, electrode tilt angle, heat transfer
Publisher:	University West, Department of Engineering Science, S-461 86 Trollhättan, SWEDEN Phone: + 46 520 22 30 00 Fax: + 46 520 22 32 99 Web: www.hv.se

Preface

This bachelor thesis was done in the welding research group at Production Technology West in Trollhättan. The thesis corresponds to 15 HE credits and is carried out at C-level. The work started in the early April and ended in the middle of June.

The welding research group does experimental studies supplemented with simulations to gain knowledge about welding processes. This knowledge can be used to control the welding process at a higher accuracy than what is available today.

I would like to thank my advisor Isabelle Choquet for letting me be part of her research and for all the interesting discussions. I would also like to thank my examiner Lars-Erik Svensson and the staff at Production Technology West for their support.

If nothing else is noted the work, figures and tables are made by the author.

Trollhättan, June 2013

Johanna Matsfelt

Contents

Appendices

Symbols

$\vec{A}$	Magnetic potential vector $[^{Vs}/_{m}]$
$\overrightarrow{A_0}$	Magnetic potential vector reference value $[^{Vs}/_{m}]$
α	Thermal diffusivity $[^{m}/_{s^2}]$
α^*	Angle [rad]
$\vec{B}$	Magnetic flux density called magnetic field $[^{Ns}/_{Cm}]$
β	Angle [rad]
c_p	Specific heat capacity $[^{J}/_{kgK}]$
$\vec{E}$	Electric field $[^{V}/_{m}]$
e	Electron charge [C]
ϵ_N	Net emission coefficient [-]
h	Specific enthalpy $[^{J}/_{kg}]$
I	Current intensity [A]
$\bar{\bar{I}}$	Unit tensor [-]
$\vec{J}$	Current density $[^{A}/_{m^2}]$
κ	Thermal conductivity $[^{W}/_{mK}]$
k_b	Boltzmann constant $[^{J}/_{K}]$
L	Length [m]
μ	Dynamic viscosity $[^{kg}/_{ms}]$
μ_0	Permeability of vacuum $[^{Vs}/_{Am}]$
p	Pressure [Pa]
q	Heat conduction per unit area $[^{J}/_{m^2}]$
r	Radius [m]
r_0	Radius of the rod [m]
ρ	Density $[^{kg}/_{m^3}]$
σ	Electric conductivity $[^{A}/_{Vm}]$
T	Temperature [K]
t	Time [s]

θ	Angle [rad]
$\vec{U}$	Velocity [$^{m}/_{s}$]
V	Electric potential [V]
x_e	Length of the last cell [m]
x_{e1}	Length of the last cell in the first block [m]
x_s	Length of the first cell [m]
x_{s2}	Length of the first cell in the second block [m]
$\nabla \cdot$	Divergence operator
∇	Gradient operator
Δ	Laplacian operator

Glossary

CFD	Computational Fluid Dynamics
CO_2	Carbon dioxide
CPU	Central Processing Unit
ER	Expansion Ratio
GMAW	Gas Metal Arc Welding
GTAW	Gas Tungsten Arc Welding
Heavy species	Atoms and ions, not electrons
IPC	Increase Per Cell
LTE	Local Thermodynamic Equilibrium
MAG	Metal Active Gas
MIG	Metal Inert Gas
NOC	Number of Cells
PDE	Partial Differential Equations
TIG	Tungsten Inert Gas

1 Introduction

During the early years in the history of welding, welding was mostly used by blacksmiths but after years of development it is now part of modern production technology. There exist many types of welding e.g. arc welding, gas welding and resistance welding. The main types of welding methods discussed in this thesis are the Gas Tungsten Arc Welding (GTAW) and the Gas Metal Arc Welding (GMAW); they are part of arc welding since the energy source is an electric arc. The GTAW process is also known as Tungsten Inert Gas (TIG). By the name it is shown that the electrode is made of tungsten, but other types of electrode materials also exist. All the materials used for the electrode in GTAW have in common that they are not consumed since they remain stable at high temperature. The gas has an inert effect and is thereby not active i.e. does not react with the electrode. Pure argon or argon together with different fractions of helium, hydrogen or nitrogen for instance can be used as shielding gas in GTAW [1]. The other type of welding, GMAW, includes Metal Inert Gas (MIG) welding and Metal Active Gas (MAG) welding. In GMAW the electrode is consumed and is often made of a material with a chemical composition similar to the base material [1]. In MIG welding the shielding gas is inert and pure argon or argon with portions of helium can be used [1]. In MAG welding the gas is active and the shielding gas can be pure Carbon dioxide (CO_2) or CO_2 with portions of helium or argon [1].

During welding heat enters the base metal. The amount of heat that enters the base metal and the temperature distribution obtained due to the heat are of interest for the material properties of the weld e.g. the size of the heat affected zone. They can also be of importance if heat sensitive components are located close to the weld. The heat distribution can change due to different parameters. In this thesis it will be investigated if the electrode tilt angle is a parameter that also influences the heat distribution. This investigation is done for GTAW with argon as shielding gas.

1.1 Background

In Computational Fluid Dynamics (CFD) simulations of the GTAW process the models often assume 0° tilt angle [2]. But in practice, the electrode used in GTAW is most of the time tilted by a few degrees [1]. This thesis will try to clarify if the assumption of 0° tilt angle in the CFD simulations is a valid assumption or not.

1.2 Overview of previous works

Research in simulation of GTAW and other welding methods concerns many different aspects such as the effect of the electromagnetic model or investigating if the assumption of Local Thermodynamic Equilibrium (LTE) is a valid assumption or not. But no articles concerning simulation of the heat source in GTAW with a tilted

electrode was found. This section is thus an overview of some of the recent articles concerning the CFD modeling of arc welding. CFD is used in arc welding to simulate the weld pool, the heat source, and the metal transfer.

Different models of the energy transport in the weld pool are investigated in the review article "Recent developments in modeling of heat transfer during TIG welding – a review" by Varghese *et al.* [2]. In this work the arc heat source is not simulated but set in the form of a boundary condition imposing a heat distribution source term. This distribution, namely a Gaussian distribution, involves adjustable parameters. A major drawback of this approach is that to be adjusted these parameters require an accurate determination of the heat source. Commercial software often use conduction as the only form of energy transport in the weld pool models [2] and no particular consideration for energy transported by e.g. convection or radiation is taken. When using a conduction model for simulating and average (rather than temperature dependent) values for the material properties such as the density or the specific heat large differences discrepancies between calculated and measured temperature are obtained [2]. It is also shown in [2] that the convection taking place in the pool significantly affects the weld pool shape.

The liquid pool simulation is also investigated in the article "GTAW liquid pool convections and the weld shape variations under helium gas shielding" by Dong *et al.* [3]. It is observed that at low oxygen content (20 ppm) the fluid flow induced by the Marangoni convection in the pool is outward, which produces wide and shallow weld shapes [3]. When the electrode gap is increased the width of the weld is increased while the ratio between the weld depth and the weld width is decreased. At high oxygen content ($\geq$80 ppm) it is found that the current density in the pool decreases when the electrode gap is increased, which causes a decrease in the electromagnetic force [3]. This results in a weakened inward Marangoni convection and a decrease in the weld depth [3].

The shielding gas and the metal vapor do influence the geometry of the weld pool as discussed in the article "Modelling of thermal plasmas for arc welding: the role of the shielding gas properties and of metal vapour" by Murphy *et al.* [4]. It is found in the article that when metal vapor is taken into account the heat flux and the current density in the weld pool are decreased. This leads to a shallower weld pool compared to results obtained when neglecting the metal vapor [4]. The shielding gases investigated numerically in [3] were argon, helium, hydrogen and nitrogen. The maximum temperature in the vicinity of the cathode occurred when pure argon was used as shielding gas [4]. The maximum velocity and temperature in the arc occurred in hydrogen shielding gas [4]. In this article a number of imaginary gases were also constructed based on argon gas replacing one of its properties such as the thermal conductivity with the property of another gas such as helium for instance (see Figure 4 in [4]). This particular example results in a larger heat conduction which leads to a

lower arc temperature (see Figure 6 in [4]). This may be used to design shielding gas mixtures with gas properties that could give better weld, although real mixture do not behave in a so simplified way. The effect of metal vapor on the arc is also investigated in this article. At low temperature the metal vapor increases the electrical conductivity within the arc [4]. The same observation is done in the article "The effects of metal vapour in arc welding" by Murphy [5]. This author also concludes that the metal vapor has major effects on properties such as the net radiative emission and the electrical conductivity. An increase of these properties results in a decrease of the arc temperature [5]. A low presence of metal vapor (as low as 1%) is sufficient to leads to a significant increase of electrical conductivity in the arc. It allows allowing the plasma to become electrically conducting at much lower temperature: 4000K in argon with 1% metal vapor, instead of 7000K for pure argon [5]. This results in a different distribution of the heat flux due to the current density [5].

The metal transfer in GMAW is discussed in the article "Three-dimensional modeling of arc plasma and metal transfer in gas metal arc welding" by Xu *et al.* [6]. Concerning the arc heat source it is concluded that in the realistic situation where the welding tool is moving the calculated fields (such as the temperature field) are not axisymmetric; they are shifted in the direction of motion [6]. As a result, it is also found that in this situation the arc pressure, the heat flux and the current density distributions on the surface of the base metal are not Gaussian distributions, contrary to the usual assumption done to simulate the weld pool [6].

In the above articles the plasma is assumed to be in local thermodynamic equilibrium. Research is currently done concerning the non-equilibrium phenomena. The article "Treatment of non-equilibrium phenomena in thermal plasma flows" by Rat *et al.* focusses on two-temperature plasma where the electrons and heavy species are at different temperatures [7]. The non-equilibrium situation occurs close to the anode and cathode surfaces and in the outer part of the arc where the shielding gas mixes with the surrounding atmosphere. It can also take place if cold gas is injected. In all these cases, as the plasma is rather cold, the electron number density is low. Then the frequency of collisions between light electrons and heavy species (ions and neutrals) is too low to allow efficient energy transfer between the species (about 40 000 collisions are needed to transfer the electron energy to the heavy species) [7]. As a result, electrons and heavy species have distinct temperatures. The assumptions made in this article when calculating the composition of the non-equilibrium plasma cannot be validated experimentally with the experimental methods available at that time (namely in 2008) [7]. In non-equilibrium plasma the ionization reaction rates also need to be considered. These rates are temperature dependent and they vary by many orders of magnitude across the arc. It implies stiff gradients and notable increase in computational time [7]. In two-temperature plasma the chemical non-equilibrium is also investigated in the article "Two-temperature chemically non-equilibrium modeling of transferred arcs" by Baeva *et al.* [8]. These authors observed that near the

electrode the deviation from the LTE is large [8]. This deviation was also observed to occur in the outer regions of the arc where the electron density is low. In the other regions of the plasma (that is in the plasma core) near LTE was observed [8]. Measurements were done and indicated that the LTE model overestimates the temperature in the arc column while it underestimates it in the arc fringes. This shows the importance of accounting for deviation from LTE (and thus chemically non-equilibrium) in the models to reach a better agreement between the temperature predicted by the simulations and the experimental results [8].

The model to be chosen for the electromagnetism is discussed in the article "On the choice of electromagnetic model for short high-intensity arcs, applied to welding" by Choquet *et al.* [9]. The electric potential model for the electromagnetism assumes that the magnetic field is one-dimensional. This model can be suited for long axisymmetric arcs when the tip of the electrode is flat (so infinite tip radius). Then the radial current density may be assumed negligible compared to the axial current density so that the magnetic field can be assumed one dimensional [9]. If the arc is axisymmetric but short and/or the electrode tip has a finite radius this simplification is not justified and a two dimensional magnetic field model is needed [9].

Research is also done concerning the interaction between the arc with the cathode and the anode. This is investigated in the article "Understanding and modeling plasma-electrode interaction in high-pressure arc discharges: a review" by Benilov [10]. The layer near the anode and cathode can be divided in regions characterized by different kinds of disturbances and governed by different kinds of physics. Close to the wall is a charged layer (so negative electron charge do not balance the ions charge). This layer is called the space-charge sheath. For the cathode this layer is about 10^{-8} m to 10^{-7} m thick and is charged positively. It has a major influence on the plasma heat source as it allows the emission of electrons from the cathode surface and provides the energy needed for promoting plasma ionization [10]. In some cases the next layer is a transition layer called Knudsen layer. Further away from the wall is a layer that is not charged (so negative electron charge balances the ions charge) in which ionization takes place and this ionization is not balanced by recombination. It thus results in a net production of ions and electrons. The ionization layer is also very important for the cathode as ion production by ionization takes place there. These ions are attracted to the cathode and give thermal energy to the cathode (this thermal energy is used to emit electrons from the cathode). This layer is less important for the anode. Another layer, even closer to the plasma core is neutral (balanced charges) and at chemical equilibrium (balanced ionization and recombination) but in thermal non-equilibrium (so electrons and heavy particles do not have the same temperature). This layer is called the thermal perturbation sheath and it is the last layer before the plasma core. For the anode, this last layer is at a distance of about 4×10^{-6} to 9×10^{-3} m from the anode wall. This layer and the plasma bulk are the most important sources of energy flux towards the anode [10].

1.3 Question formulation

- Investigate if the tilt angle of the electrode changes the temperature distribution on the surface of the base material in GTAW.
- Is a Gaussian distribution a correct approximation of the temperature distribution on the surface of the material?

1.4 Aim and objectives

- The aim is to investigate if the tilt angle in GTAW influences the heat transfer distribution on the surface of the base metal.
- The heat distribution on the base metal resulting from the CFD simulation with and without tilt angle will be compared.
- Determine from this qualitative study if a quantitative study is worth being done in future work.

1.5 Limitations

- The melting of the base material will not be considered, nor will the influence of metal vapor on the arc. These limitations are reasonable since the experiments allowing measuring the total heat transferred to the base metal is done for water cooled base metal.
- The conditions in the cathode layers located on the electrode surface and in the anode layers located on the base metal will be set and not calculated. This is the reason why this study is qualitative rather than quantitative.

2 Methodology

To investigate if the tilt angle in GTAW influences the heat transfer distribution on the surface of the base metal, we could either consider experimental or modeling investigation. Experiment investigations can allow measuring the global heat transfer to the base metal, through the measurement of the temperature difference between inlet and outlet water used for cooling the base metal for instance. They can also allow measuring a local temperature distribution on the metal surface (using an infra-red camera for instance). However, the local heat transfer as well as the local temperature in the plasma just above the surface cannot be measured with nowadays measurement tools. So the heat transfer distribution on the surface of the base metal is not yet accessible experimentally. The remaining option is thus modeling. As the heat source of GTAW is modeled by a large system of coupled and non-linear equations (see section 3.3) the model has no analytic solution and CFD modeling needs to be used.

For this, two gas tungsten arc simulations were planed: one without tilt angle and one with a large tilt angle for welding applications, namely 12°.With no tilt angle the

problem can be assumed axisymmetric. Then a 2D axisymmetric mesh rather than a 3D mesh avoids unnecessary computer memory and Central Processing Unit (CPU) time. With a tilt angle this simplification cannot be done and a 3D mesh is needed. It can however be reduced to half of the space when the tilted electrode remains aligned with the welding direction, as further detailed below.

The calculation of the heat transfer distribution on the surface of the base metal is done solving thermal fluid equations coupled with electromagnetism for modeling the arc heat source. This CFD calculation is done with the CFD software OpenFOAM (version 2.1.x). This software is installed on the computer cluster at University West-PTC. The cluster is operated by a Linux system. A preliminary step in this project was thus to get familiar with the Linux operating system.

When starting this project a mesh for axisymmetric GTAW was available. But there was no 3D mesh. Developing the 3D mesh was an important part of this project. Such a mesh cannot be obtained doing a straightforward extension of the 2D axisymmetric mesh. The following strategy was thus used to understand step by step how to proceed.

Step 1: In the first step a simpler problem (allowing shorter calculation time) and a simpler geometry were considered: electromagnetic conduction in an infinitely long conduction rod. This simpler problem has another advantage of importance: it has an analytic solution that can be used for testing the quality of the simulation results. A 2D axisymmetric mesh suited for an infinite rod was available to start from. This mesh was further developed to a 3D mesh.

It was then realized that CPU time could be saved by removing half of the 3D mesh taking advantage of the symmetry of the problem. The 3D mesh was then reduced to a half 3D mesh with a symmetry plane. The half 3D mesh was done in different ways to enable a good mesh quality, as further detailed later on.

These half 3D meshes were used to run simulations. The simulation results were compared with the analytical solution and with the numerical solution from the 2D axisymmetric mesh. The chosen type of mesh was further used in the next step for developing the 3D mesh for GTAW.

Step 2: For the GTAW case with 0° tilt angle a 2D axisymmetric mesh was also available. The 2D axisymmetric mesh was studied to understand in detail how it had been made. A 3D mesh was then done for the same GTAW problem with 0° tilt angle. This mesh was then reduced to a half 3D mesh with a symmetry plane along the weld path (again to save memory and CPU time).

This half 3D mesh was then further developed to enable a tilt angle of 12°. The simplification to a half space could be maintained as the tilted electrode was aligned with the weld path.

The simulations were done using the 2D axisymmetric mesh for the 0° tilt GTAW case and the half 3D mesh for the 12° tilt GTAW case. The simulation results were compared. The last step did consist in proposing model modifications for future improvements.

3 Theory/models

Two models were used. The first is the simplest: a model for electromagnetism alone. The electromagnetic model was validated against an analytical solution to an infinite electrically conduction rod problem. The second is the more complex magneto hydrodynamic model coupling electromagnetism with a thermal fluid model. The second model has no analytic solution.

3.1 Electromagnetic model

The equations used in this section are from page 3 in the article "On the choice of electromagnetic model for short high-intensity arcs, applied to welding" by Choquet *et al.* [9]. Further details on how the equations are derived can also be found in this article. To calculate the electric potential (V) the following partial differential equation (PDE) named Poisson scalar equation is solved

$$\nabla \cdot (\sigma(T)\, \nabla V) = 0 \qquad (1)$$

In (1) $\nabla\cdot$ is the divergence operator, ∇ the gradient operator, σ the electric conductivity which is temperature depended, and T is the temperature. The electric conductivity used in the arc simulation for argon gas can be seen in Figure 1 bellow.

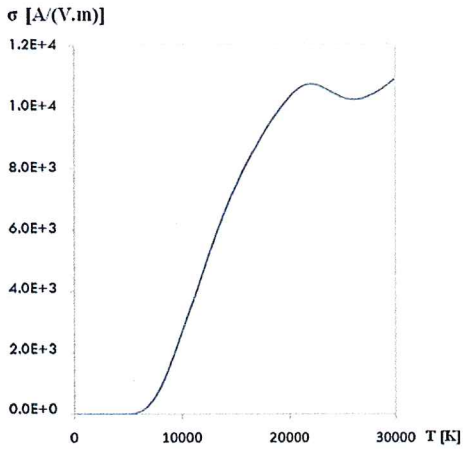


Figure 1: Electric conductivity of argon gas dependency on the temperature, from Choquet *et al.* [9], with permission.

Knowing the electrical potential, the magnetic potential vector $\vec{A}$ can be calculated by solving the following PDE named Poisson vector equation

$$\Delta\vec{A} = \mu_0\sigma(T)\nabla V \qquad (2)$$

In (2) Δ is the Laplacian operator and μ_0 the permeability of vacuum $\left(4\pi \times 10^7\ {}^{N}/_{A^2}\right)$. Knowing V and $\vec{A}$, the current density $\vec{j}$, the electric field $\vec{E}$, and the magnetic field $\vec{B}$ can be calculated from

$$\vec{j} = \sigma(T)\vec{E} \qquad (3)$$

$$\vec{E} = -\nabla V \qquad (4)$$

$$\vec{B} = \nabla \times \vec{A} \qquad (5)$$

3.2 Analytical solution to the infinite rod

The above electromagnetic system (1)-(5) can be solved analytically in the particular case of an infinite long electrically conducting rod surrounded by a poorly conducting media. This analytic solution is used later on as reference to check the quality of the 3D mesh developed in this study. The equations in this section are from page 40 and 41 in the Licentiate thesis "Plasma Arc Welding Simulation with OpenFOAM" by Sass-Tisovskaya [11]. Further details on how the equations are derived can also be found in this document. In this analytical solution to the infinite rod problem it is known that the only non-zero component of the magnetic potential vector is the axial component which is denoted by A_x [11]. The axial component of the magnetic potential changes only along the radial direction r according to

$$A_x = A_0 - \frac{\mu_0 j_x r^2}{4} \qquad (6)$$

In (6) A_0 is the reference value of the magnetic potential vector to enable the numerical and analytical solutions to be compared. The radius at which the vector magnetic potential is calculated is denoted by r. The current density along the axial direction is denoted by j_x in (6) and it is obtained from

$$j_x = \frac{I}{\pi {r_0}^2} \qquad (7)$$

The current intensity is denoted by I in (7) and r_0 is the radius of the conducting rod. The solution of equation (2) outside of the rod is

$$A_x = A_0 - \frac{\mu_0 j_x r^2}{2}\left(0.5 + ln\left(\frac{r}{r_0}\right)\right) \qquad (8)$$

The magnetic field reduces to a single component along the azimuthal direction, function of the radial position r. It is obtained from

$$B = \frac{\mu_0 j_x r}{2} \quad if \quad r \leq r_0 \qquad (9)$$

Outside of the rod the magnetic field is

$$B = \frac{\mu_0 j_x {r_0}^2}{2r} \quad if \quad r \geq r_0 \qquad (10)$$

3.3 Magneto hydrodynamic model

The magneto hydrodynamic model is used for simulating GTAW. Details of how the equations are obtained can be found on pages 19-28 in the Licentiate thesis "Plasma Arc Welding Simulation with OpenFOAM" by Sass-Tisovskaya [11].

The system of equations is used to obtain among other the pressure field, the temperature field and the current density, which are fields of major interest in welding [9]. The first equation is the conservation of mass also known as the continuity equation

$$\frac{\partial}{\partial t}\rho + \nabla \cdot \left(\rho \vec{U}\right) = 0 \qquad (11)$$

where t is the time, ρ the density which depends on the temperature, and $\vec{U}$ is the velocity of the flow. This equation is combined with the conservation of momentum for laminar flow also known as the laminar Navier-Stokes equation

$$\frac{\partial}{\partial t}\left(\rho \vec{U}\right) + \nabla \cdot \left(\rho \vec{U}\vec{U}\right) - \vec{U}\nabla \cdot \left(\rho \vec{U}\right) + \nabla \cdot \left(\mu\left(\nabla \vec{U} + \left(\nabla \vec{U}\right)^{T}\right) - \frac{2}{3}\mu(\nabla \cdot \vec{U})\bar{\bar{I}}\right) =$$

$$-\nabla p + \vec{j} \times \vec{B} \qquad (12)$$

In (12) the dynamic viscosity denoted by μ depends on the temperature, p is the pressure, and $\bar{\bar{I}}$ is the unit tensor. The current density $\vec{j}$ is known from equation (3) and the magnetic field $\vec{B}$ is known from equation (5). The conservation of enthalpy h is also part of the system of equations as we do investigate a thermal fluid

$$\frac{\partial}{\partial t}(\rho h) + \nabla \cdot \left(\rho \vec{U} h\right) - h\nabla \cdot \left(\rho \vec{U}\right) - \nabla \cdot \vec{q} =$$

$$\nabla \cdot \left(\vec{U}p\right) - p\nabla \cdot \vec{U} + \vec{j} \cdot \vec{E} - 4\pi\epsilon_N + \nabla \cdot \left(\frac{5k_b \vec{j}}{2ec_p} h\right) \qquad (13)$$

In (13) the temperature dependent net emission coefficient is denoted ϵ_N, k_b is the Boltzmann constant $\left(1.380658 \times 10^{-23}\ {}^{J}/_{K}\right)$ and e is the electron charge ($1.60217733 \times 10^{-19}\ C$). The third term on the right hand side, $\vec{j} \cdot \vec{E}$, is the Joule heating. The electric field $\vec{E}$ is known from equation (4). The heat conduction per unit area denoted by $\vec{q}$ in (13) is given by Fourier's law expressed with the enthalpy

$$q = \alpha \nabla h \qquad (14)$$

The thermal diffusivity denoted by α in (14) is a function of other quantities according to

$$\alpha = \frac{\kappa(T)}{\rho c_p(T)} \qquad (15)$$

The thermal conductivity κ is temperature dependent. The thermal capacity c_p is also temperature dependent; it is obtained at constant pressure from

$$c_p(T) = \left(\frac{\partial h}{\partial T}\right)_p \qquad (16)$$

4 Design/mesh

To learn how the mesh should be designed and written in OpenFOAM an infinite rod was studied as starting point. This is a simpler case than GTAW since the model is reduced to electromagnetism and the geometry of the problem is simpler. The methodology was then applied to the GTAW cases with 0° and 12° tilt angle.

4.1 Methodology

The meshes were made in several steps. It was first decided to split the domain into blocks based on the geometry of the problem. The introduction of blocks indeed allows a better control of both the geometry and progression in size of the cells forming the mesh. The nodes defining a block need to be oriented and numbered according to OpenFOAM pre-processor (called blockMesh) rule [12]. The nodes delimiting any edge of a block are connected by straight lines or by arcs. The block faces corresponding to boundaries of the computational domain are identified as patches to be able to specify boundary conditions on these patches. The cells of a block are designed to ensure a good mesh (and thus simulation) quality according to the following standard criteria:

- The cell size is not allowed to change by more than 10 % between neighboring cells [13].
- The skewness of the elements is as much as possible avoided to try to keep the cells angles close to 90°. This criterion allows representing in a correct way the influence of pressure in the momentum equation [13].
- The cells are refined in regions with large gradients and they can be larger in regions with smaller gradient [13].
- The whole computational domain is covered by cells and the cells do not overlap each other, implying that they are contiguous [12].
- The cells are convex [12]. This means that any two nodes in any cell can be tied together with a line which does not go outside of the cell, and the geometrical cell center is inside the cell [12].
- All the face area vectors of any cell are pointing out of the cell and their sum is equal to zero up to the computer accuracy (equal to 10^{-12} in the present study) [12]. This criterion is referred to as geometrical closedness.
- All the edges of any cell each connect two and only two faces of the cell, so that the cells fulfill the criterion of topological closedness. [12].
- The orthogonality criterion is met for all the cells [12].

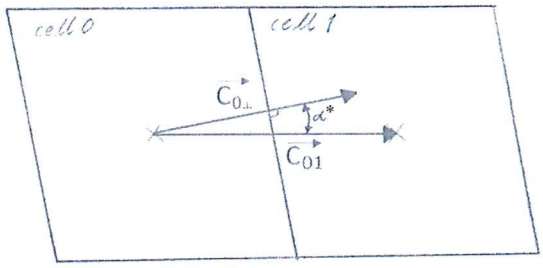


Figure 2: The orthogonality criterion described by an example

The orthogonality criterion is explained with the example of Figure 2. The vector from the geometrical center of cell 0 to the geometrical center of cell 1 is denoted $\overrightarrow{C_{01}}$ in Figure 2. In the same figure the vector $\overrightarrow{C_{0\perp}}$ can be seen which goes from the geometrical center of cell 0 and is perpendicular to the face between cell 0 and cell 1. The angle formed between the vectors $\overrightarrow{C_{0\perp}}$ and $\overrightarrow{C_{01}}$ is denoted α^* in Figure 2. The orthogonality criterion is fulfilled if α^* is less than 90° [12].

4.1.1 Calculation of the increase in size between adjacent cells

When setting the discretization parameters to split an edge (of length L) of a block into cells, the increase in size between neighboring cells is not an input parameter in OpenFOAM. This can be a problem for controlling the increase in size criterion. The parameters to be entered are the number of cells (NOC), the expansion ratio (ER) and the length and radius for the location of the nodes in the mesh. So it important to check if these quantities fulfill the criterion of maximum 10 % increase in size between neighboring cells (IPC). The following calculations were done in that aim.

The equation for the expansion ratio (see Figure 5.5, page 139 in the user guide of OpenFOAM) [12] is

$$ER = \frac{\delta_e}{\delta_s} \qquad (17)$$

where δ_e and δ_s are the lengths of the last and first cells along the chosen edge. For convenience, these lengths will be referred as x_e and x_s in this text. ER is here supposed to be known.

From the definition of the IPC, the length of the j^{th} cell is

$$x_j = x_s * IPC^{j-1} \qquad (18)$$

where x_s denotes the length of the 1st cell.

Knowing that the total length of the block edge is L, and using (18) we have

$$L = \sum_{j=1}^{NOC} x_j = \sum_{j=1}^{NOC} x_s * IPC^{(j-1)} \qquad (19)$$

So that knowing the number of cells NOC, we can calculate the increase in size between neighboring cells inserting equation (17) in (20) and rearranging. It leads to

$$IPC = ER^{\left(\frac{1}{NOC-1}\right)} \qquad (20)$$

Then it can be checked that the increase in size between neighboring cells does not exceed the maximum criterion of 10 %. If this criterion is not satisfied, ER and NOC need to be adjusted. For instance, if the NOC is kept unchanged, the ER parameter can be adjusted using the following data:

- the length of the last cell (obtained from (18))

$$x_e = x_s * IPC^{NOC-1} \qquad (21)$$

- the length of the first cell (obtained from (19))

$$x_s = \frac{L}{\sum_{j=1}^{NOC} IPC^{(j-1)}} \qquad (22)$$

- and the expansion ratio (see (17))

$$ER = \frac{x_e}{x_s} \qquad (23)$$

These equation were implemented in a Matlab code (see Appendix A) to calculate the increase in length per cell, and adjust the expansion ratio when needed.

4.1.2 Calculation of the increase in size between cells of adjacent blocks

The criterion of 10% maximum increase in size between neighboring cells also needs to be fulfilled in the case of two adjacent cells belonging to different blocks. Equations (17) - (23) are used for each of the two cells. The increase in size between the last cell of a first block (x_{e1}) and the adjacent first cell of the next block (x_{s2}) is then calculated from

$$IPC = \frac{x_{s2}}{x_{e1}} \qquad (24)$$

This quantity together with the calculated expansion ratio for each of the cells, are provided as output data by the Matlab code given in Appendix B.

4.2 Infinite electric rod

Physical problem: The rod described for this problem first set of test cases can be seen in Figure 3 bellow.

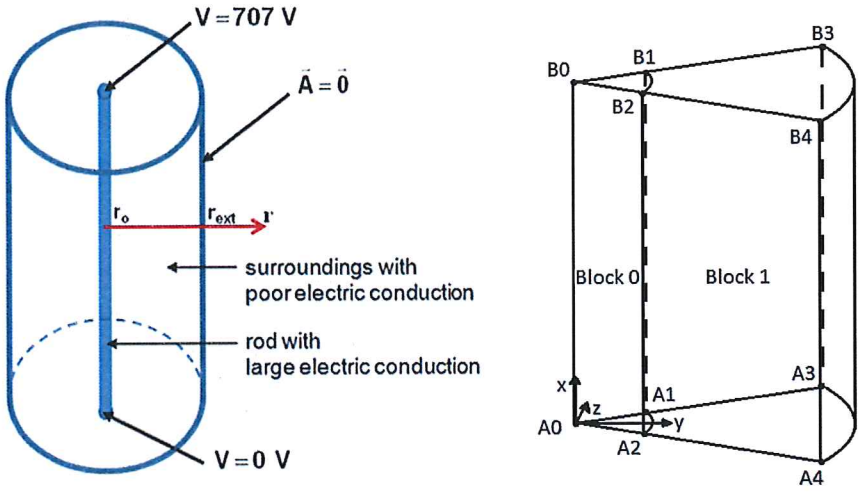


Figure 3: Left: Picture of a section of the infinite rod from Choquet *et al.* [9], with permission; Right: blocks and nodes for the corresponding 2D axisymmetric mesh

The radius of the computational domain is r_{ext} = 16 mm and the radius of the conducting rod is r_o = 1 mm. The height h of the computational domain is 10 mm, which approximates an infinitely long rod as h >> r_o (more precisely h = 100 r_o). Current is flowing in the conducting rod and the current intensity is set to 600 A. The rod (marked in blue in Figure 3, left) has the electric conductivity of 2700 A/Vm which corresponds to argon gas at T=10600 K [9]. The volume surrounding the rod has an electric conductivity 10^{-5} A/Vm which corresponds to argon gas at T=300 K [9].

The first step for preparing the numerical simulation consists in preparing the mesh. Preparing a mesh includes different tasks. First split the computational domain into blocks. Next define each block based on nodes and edges. And then discretize the blocks into cells. A 2D axisymmetric mesh done by Sass-Tisovskaya [11] has been used as basis for doing the 3D mesh needed here. It should be noticed that all the meshes in OpenFOAM are generated as 3D even the 2D axisymmetric problems [12]. This is the reason why the computational domain used for the 2D axisymmetric mesh is a sector of angle of 5° rather than 0°. From now on this 5° sector angle is referred to as the wedge angle.

2D axisymmetric mesh:

The computational domain is shown in Figure 3, right. It is decomposed into blocks, each block being from 8 nodes and edges. The block faces delimiting the computational domain need to be listed in a so-called patch list and named to be able to set the boundary conditions needed for the numerical calculations. The blocks are then discretized in space, controlling the discretization size along each edge. These different steps are now further detailed.

Blocks: In this test case the computational domain is split into two blocks: one block (named block 0) for the central part with high electric conductivity and the second block (named block 1) for the surrounding with poor electric conductivity.

Block nodes: To build these blocks each block-node (simply called node) needs to be specified [12]. Every node is defined by x, y and z coordinates and an identification name (e.g. A3 as can be seen in Figure 3, right). The coordinates can be set as real numbers or as equations. The later method is used in this thesis to allow changing easily the geometrical parameters, as can be seen in the example of Appendix C. The order in which the nodes define a block is important [12] since it affects the sign of the fluxes calculated by OpenFOAM [12]. The convention used is now explained with an example: the block with the shape of a cube shown in Figure 4 bellow.

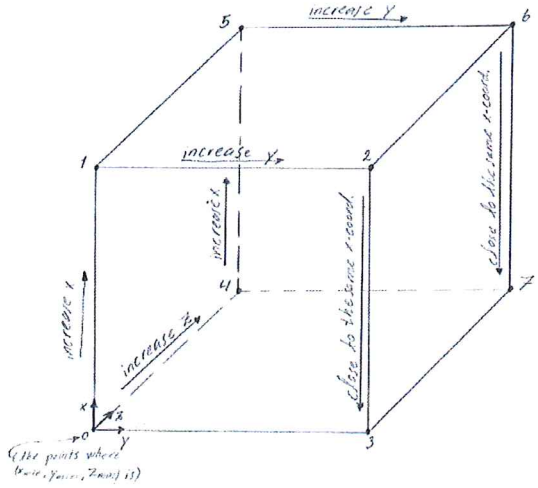


Figure 4: The way of defining the nodes in a block, if it is a cube

The first node delimiting the block has the lowest value in the x, y and z directions. It is named node 0 (Figure 4) and is used as a local origin for the block. The x-coordinate is then increased to move to node 1. This (node 0 – node 1) direction defines the local axis x_1 of the block. Node 2 is reached from node 1 by increasing the y-coordinate. This (node 1 - node 2) direction defines the local axis x_2. The first face of the block is closed by decreasing first the x-coordinate to reach node 3 and then by decreasing the y-coordinate to move from node 3 back to node 0. Starting again from

node 0 the z-coordinate is then increased to obtain the first node on the opposite side (or face) of the block; for the block this is node 4. This (node 0 – node 4) direction defines the local axis x_3 of the block. Then we proceed as for the first face: the x-coordinate is increased to go to node 5, next the y-coordinate is increased to go to node 6. The second side of the block is then closed by decreasing first the x-coordinate to move to node 7 and finally by decreasing the y-coordinate to move from node 7 back to node 4. All the blocks need to be defined in a consistent way for the whole mesh [12].

Block edges: The type of connection between two nodes needs to be specified. In this thesis straight lines and arcs are used. Examples can be seen in Figure 3 (Right) where the nodes A1 and A3 are connected by a line and nodes A3 and A4are connected by an arc. When arcs are defined the center of the curvature also needs to be specified and it is defined in the same way as the nodes by x, y, and z coordinates [12].

Boundaries – or patches: Patches are the faces of blocks that are faces of the computational domain on which boundary conditions need to be set [12]. The nodes of the patches are also ordered in a specific way to make the unit vector to the face point out of the domain [12].

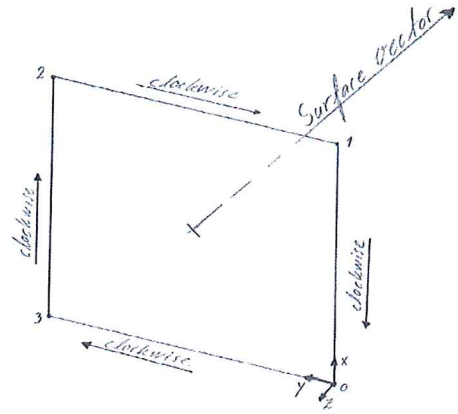


Figure 5: How the patch for the front side of the cube would be defined

The previous example with the cube of Figure 4 is used again. The front side of the cube can be seen in Figure 5 looking from the inside of the cube. When defining the patch for this side the definition of the nodes can start from any of the nodes [12]. Then continue to move around the nodes of the face in the clockwise direction when looking from the inside of the block. Starting from node 3 for instance, this would result in node 3 followed by node 2 continuing to node 1 and ending at node 0. With this ordering the face vectors point out of the domain as shown in Figure 5.

The patches for the test case of the infinite rod can be seen in Figure 6. The boundary conditions are specified on each patch. The patch "symmetry axis" is defined as a symmetry axis for both electric and magnetic potential. The patches "front" and "back" are specified as "wedge" boundary condition holding for axisymmetric

configuration. The boundary conditions for the rest of the patches summarized in Table 1 bellow.

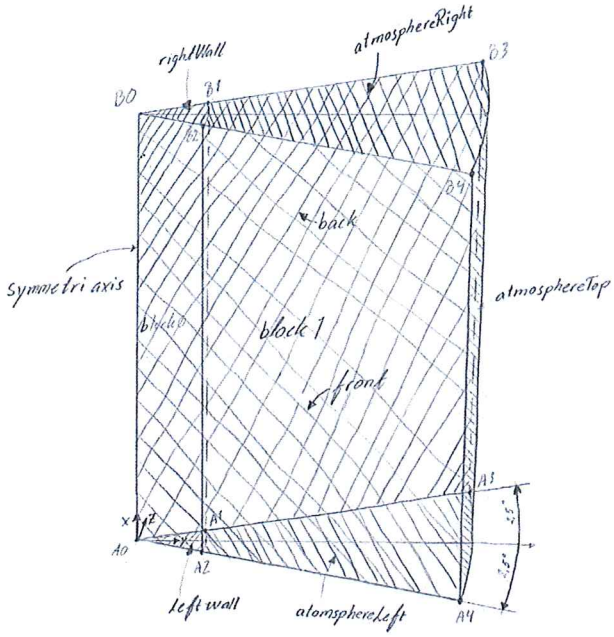


Figure 6: The infinite rod with patches

Table 1: The boundary conditions for patches in the 2D axisymmetric infinite rod mesh

Patch:	A [Vs/m]	V [V]
atmosphereTop	uniform 0	(dV/dn)=0
atmosphereRight	(dA/dn)=0	(dV/dn)=0
atmosphereLeft	(dA/dn)=0	(dV/dn)=0
leftwall	(dA/dn)=0	uniform 707
rightWall	(dA/dn)=0	uniform 0

The boundary conditions summarized in Table 1 were set based on a study done by Sass-Tisovskaya pages 46-48 in [11]. The gradient of the magnetic potential vector (Neumann condition) was set to zero on all the patches except one (otherwise the problem would be ill-posed. A Dirichlet condition is imposed on the remaining patch. This is done setting the magnetic potential vector to zero. It is important that the patch on which this zero magnetic potential vector is imposed is far away enough from the conducting rod to be valid. This is the reason why the radius of the computational domain is so large compared to the rod radius r_0. The gradient of the electric potential was set to zero on patches where there is no conduction of the electric current. This condition is indeed inferred from (3) and (4). It was decided to set the electric potential to 0 V on one end of the conducting rod (on the patch called "rightWall"). The electric potential set on the opposite end of the conducting rod was

set to 707 V. This value was calculated from equations (3), (4) and (7) so that the total current I = 600 A. It is thus expected that all the current is flowing inside the rod and along the axial direction of the rod. .

3D mesh: To develop a 3D mesh based on the 2D axisymmetric mesh the wedge angle was first increased from 5° to 90°

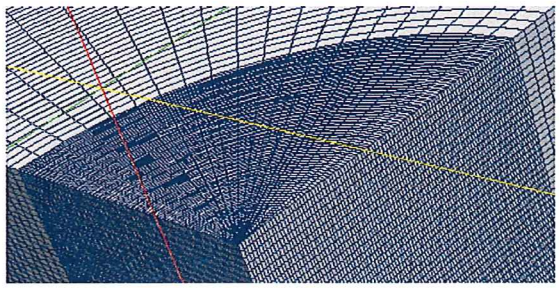


Figure 7: Sharp elements in the middle of the mesh

But this approach produces elements with very large skewness, as can be seen in Figure 7. Element skewness is not desired when solving numerically PDE [13].

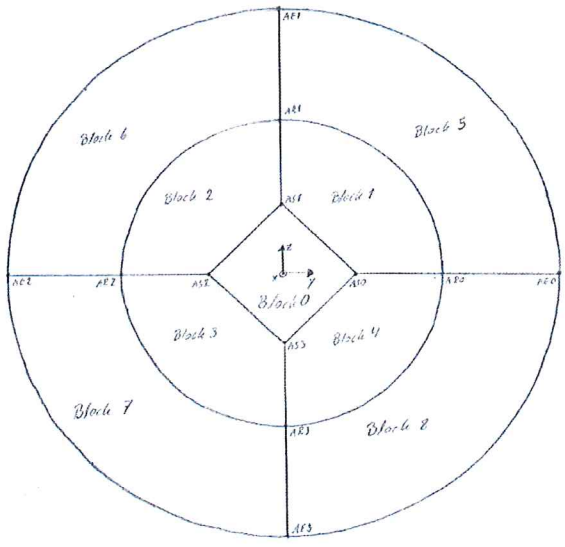


Figure 8: Enlarged middle section of plane A, not in scale

To avoid this problem and fulfill the skewness criteria (see section 4.1) a better control the cells shape was needed. For this, the computational domain was decomposed into a larger number of blocks including a block of square section (Block 0) inserted in the middle of the computational domain. . A sketch of a section of the computational domain in plane A can be seen in Figure 8. The whole computational domain for the 3D mesh can be seen in Figure 9 bellow.

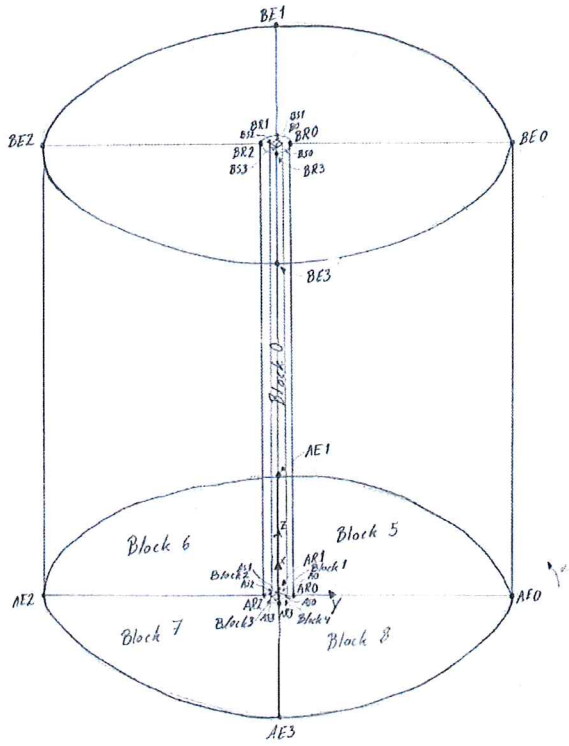


Figure 9: Sketch of the computational domain of the 3D mesh, with coordinate system

The instructions coded to do the whole 3D mesh are available in Appendix C. It was then realized that CPU time could be saved by using only half of the mesh (without changing the physics of the problem).

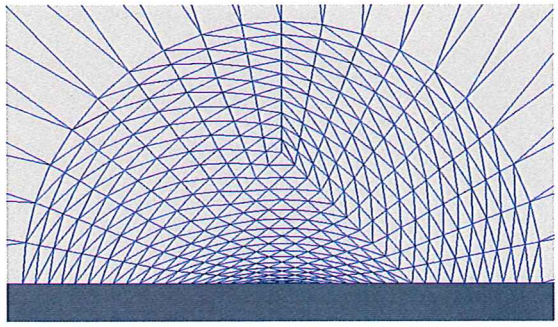


Figure 10: Middle section of the half 3D mesh with square setup from the whole 3D mesh.

The cells in a section of the half 3D mesh can be seen in Figure 10. In the middle of the lower part of Figure 10 the cells have high skewness which is an unwanted behavior. To avoid this problem the square block was then turned by 45°. To do this new nodes were needed to obtain the corners e.g. AS01 as can be seen in Figure 11. The mesh with this changed square setup can be seen in Figure 12. The cells in the middle of the lower part of Figure 12 are not skew and can provide a good base for a high quality simulation.

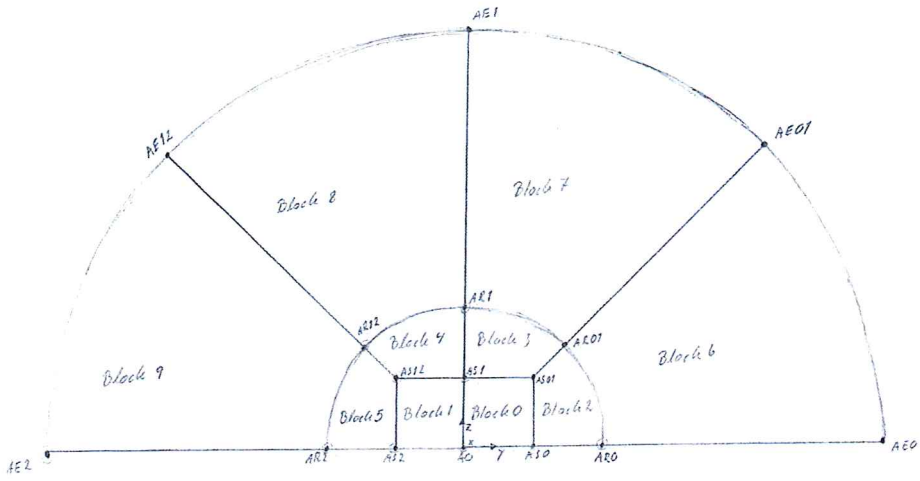


Figure 11: The left part of the infinite rod half mesh with blocks, the sketch is not made to scale.

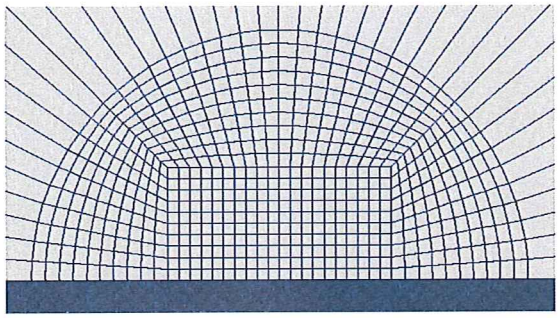


Figure 12: Middle section of the half 3D mesh with changed square setup

As can be seen in Figure 11 the nodes of the 3D mesh were renamed compared to the 2D axisymmetric mesh in Figure 6. The new notation includes two letters and a number. The first letter of a node name refers to a plane normal to the symmetry axis (called here the x-axis). These planes are the same as for the 2D axisymmetric mesh. Plane A is located in x = 0 and plane B is in x = 10 mm, see Figure 11. The second letter of a node name refers to the radial location of the node. The letter S holds for a node located at a corner of the square in the middle. The letter R holds for nodes located in r_0, the rod radii. The letter E means that the node is located on the radial boundary of the computational domain. Finally the number indicates the direction: "0" is along +y (identified by the angle θ=0°), "1" is along +z (identified by the angle θ=90°) and "2" is along –y (identified by the angle θ=180°). The nodes with numbers "01" or "12" are located between the nodes "0" and "1" (and identified by the angle θ=45°)or between nodes "1" and "2" (and identified by the angle θ=135°). This notation was used to for defining the mesh nodes, as can be seen in Appendix D.

The Cartesian coordinates (x, y, z) of the nodes in Appendix B are thus easily related to the node names:

- the x-coordinate is the x-coordinate of the plane identified by the first letter of the name.
- the y-coordinate and z-coordinate are easily calculated from the radial distance r to the x-axis (which is identified through the second letter in the node name) and the angular position θ (indicated by the number at the end of the node name): $y = r \cos(\theta)$ and $z = r \sin(\theta)$.

The number of blocks was increased from 2 blocks for the 2D axisymmetric mesh to 10 blocks for the 3D mesh. The Block 0 and Block 1 are new and form the square in the center of the conducting rod. Block 2 to Block 5 form the outer region of the conducting rod, and their name number increases with the angular position θ. Block 6 to Block 9 form the region with poor electric conductivity which extends outside the conducting rod up to the radial boundary of the computational domain.

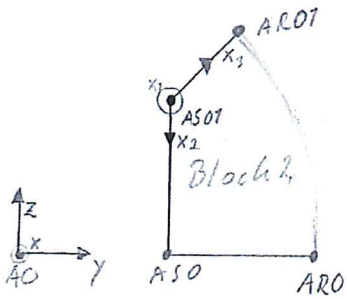


Figure 13: Plane A part of Block 2 with global and local coordinate system

When defining the blocks of the half 3D mesh the nodes of the blocks are ordered as described in section "Block nodes" page 16. For Block 2 for instance starts from the node AS01 (see Figure 13). The local x_1 direction of this block is defined by the nodes AS01 and BS01. The local x_2 direction is defined by the nodes AS01 and AS0. The local x_3 direction is defined by the nodes AS01 and AR01. Finally the ordered list of nodes defining block 2 is `(AS01 BS01 BS0 AS0 AR01 BR01 BR0 AR0)`, as can be seen in appendix D. the other blocks are defined in a similar way in Appendix D.

The number of cells along the directions x_1, x_2 and x_3 are specified for each block as parameters to enable easy change. The number of cells along the x_1-direction is specified as "xABnumberOfCells" in Appendix D. The number of cells along the x_2 and x_3 direction are both specified as "rRodSNumberofCells" for blocks inside the square, "rRodNumberOfCells" for blocks inside the rod. Outside of the rod the number of cells in the x_2 direction is specified in "rNumberOfCells" and "rNumberOfCellsExterior" along the x_3 direction.

Four tests given in Table 2 were run for the half 3D Mesh with different number of cells along the directions of the blocks. The main goal of these tests was to investigate the influence of the number of cells along the radial direction on the numerical solution, to determine the number of cells needed to reach convergence in mesh.

Table 2: Specification of test case 1, 2, 3 and 4 for the half 3D mesh and the infinite rod.

Parameter:	Test 1	Test 2	Test 3	Test 4
xABnumberOfCells	100	100	50	50
rRodSNumberOfCells	2	20	40	50
rNumberOfCells	2	20	40	50
rRodNumberOfCells	2	20	40	50
rNumberOfCellsExterior	20	200	400	500

In the half 3D mesh the cells of the blocks inside the rod were assumed to be uniform setting "simpleGrading" to one in all the directions (see Appendix D where "rGrading" is equal to 1) [12]. Concerning the blocks outside of the rod, their cells do increase in size along the radial direction. The increase in size ("simpleGrading") is indicated by the parameter "rGradingExterior" (see Appendix D). This parameter is the expansion ratio ER (as can be seen in Figure 5.5 in the User Guide version 2.1.1. [12)]. It was set to 5. Using the Matlab code of Appendix A the increase per cell IPC was calculated along the radial direction inside and outside the rod. The output results are given in Table 3. In this table, the length L represents the radial length.

Table 3: Increase per cell calculation in the infinite rod mesh.

Input:	L [10^{-4} m]	NOC, test 1	NOC, test 2	NOC, test 3	NOC, test 4	ER [-]
Inside the rod	5.5	2	20	40	50	1
Outside the rod	150	20	200	400	500	5

Output:	IPC, test 1	ERc, test 1	IPC, test 2	ERc, test 2	IPC, test 3	ERc, test 3	IPC, test 4	ERc, test 4
Inside the rod	1	1	1	1	1	1	1	1
Outside the rod	1.0884	5	1.0081	5	1.004	5	1.0032	5

As expected for a uniform mesh, the expansion ratio calculated inside the rod is equal to one in all the test cases.

The increase per cell calculated with the Matlab code (see Appendix A) is given in Table 3 for the different test cases. It can be seen in this table that for each of the test cases the calculated increase per cell (IPC) is less than 1.1, meaning that the criteria of maximum 10% increase in cell is satisfied. The increase between the last cell in the rod and the first cell outside of the rod were also calculated based on the input data of Table 3. The calculation was also done using the Matlab code of Appendix B. The results are reported in Table 4.

Table 4: Calculated increase in size of adjacent cells at the interface between the rod and the exterior

Output:	IPC, test 1	IPC, test 2	IPC, test 3	IPC, test 4
Limit between	1.0855	1.0962	1.0968	1.0969

It can be seen in Table 4 that for each of the test cases the calculated increase per cell (IPC) from block to block is less than 1.1, meaning that the criteria of maximum 10% increase in cell is satisfied everywhere in the computational domain.

The edges of the computational domain are also defined as "line" and "arc" for the half 3D mesh, as can be seen in Appendix D. All the edges of the computational domain (see Figure 9) are specified in the section "edges" in Appendix C.

Boundaries (or patches) and boundary conditions: Most of the boundary conditions are set as for the 2D axisymmetric case. They can be found in Table 1 and are thus not recalled here. Only the new boundaries introduced for the half 3D case are listed below. The patches "back", "front" and "symmetry axis" (see Figure 6) are specific for the 2D axisymmetric configuration defined on a 5° sector and are not present in the 3D mesh. On the contrary, a symmetry plane (identified in the mesh file with the name "halfModelPlane") is added. All the other patches have the same boundary conditions as in Table 1 but the number of block faces each patch contains is larger than for the 2D axisymmetric mesh.

4.3 GTAW

For the GTAW test cases the electromagnetic model is coupled with thermal fluid dynamics, resulting in the magneto hydrodynamic model of section 3.3. The meshes were made for a tilt angle of zero and twelve degrees. The geometry of the GTAW tool used for preparing the meshes is shown in Figure 14 bellow.

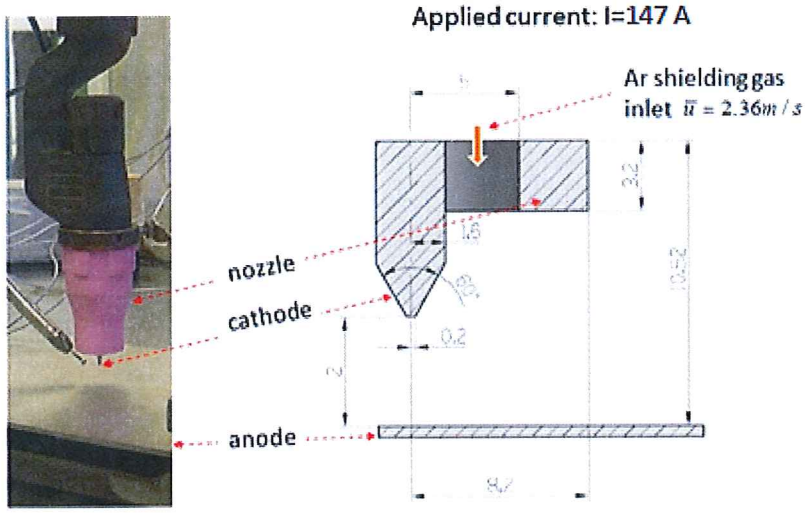


Figure 14: Left: Picture of the GTAW setup (left). Right: sketch to the scale from Choquet *et al.* [9], with permission.

The height of the computational domain is 10.52 mm and its radius is 8.2 mm. The arc length, or distance between the electrode (the cathode) and the base metal (the anode) is 2 mm. The tip of the electrode has an angle of 60°. The minimum tip radii are 0.2 mm and the maximum is 1.6 mm. The internal and external radii of the nozzle are 5 mm and 8.2 mm, respectively. This setup is used when doing the GTAW mesh.

The computational domain is the volume occupied by the gas. The solid parts e.g. the anode, cathode and nozzle are modeled through boundary conditions. In Figure 14 some of the quantities useful for setting the boundary conditions can be seen e.g. the shielding gas is argon and has an average velocity of 2.36 m/s at the inlet. The current is set to 147 A based on experimental data.

4.3.1 Zero degree tilt angle

This first GTAW mesh used was made by Choquet *et al.* [9]. This mesh had a 5° wedge angle and can be seen in Figure 15 bellow.

The 2D axisymmetric mesh is made of seven blocks. Block 0 is the volume located bellow the electrode tip. The other blocks can easily be seen in Figure 15.

The names of the patches on which boundary conditions are set are indicated in Figure 15. The patch "symmetry axis" is defined as a symmetry axis for all the variables of the problem. The patches "front" and "back" are treated as for the

infinite rod in the 2D axisymmetric case. The boundary conditions specified on the other patches are summarized in Table 5.

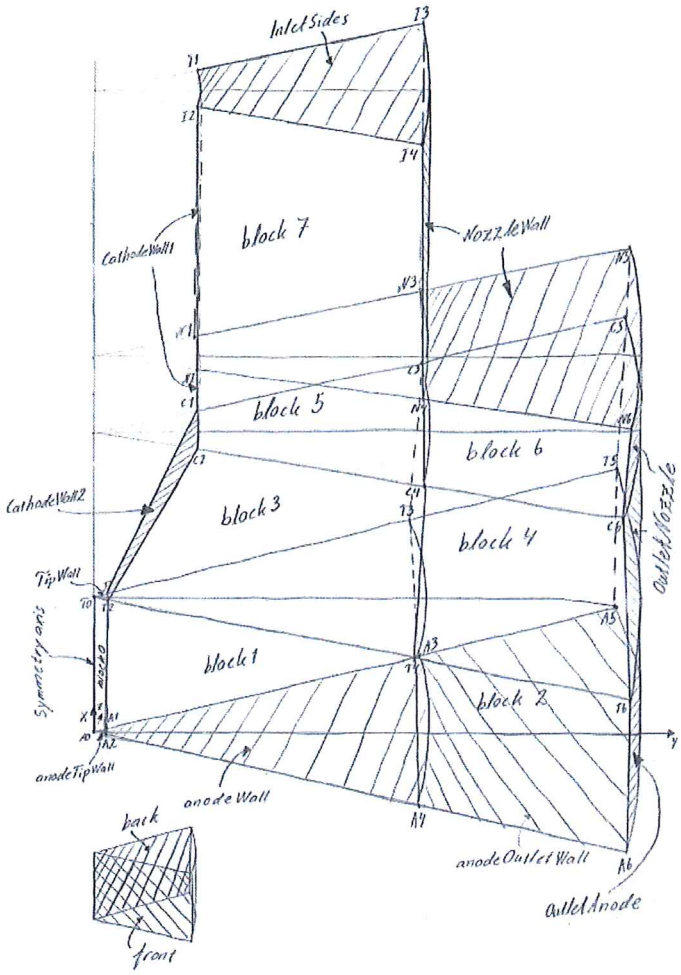


Figure 15: GTAW zero degree tilt angle 2D axisymmetric mesh, total height not in scale

The criteria used for setting the boundary conditions for the magnetic potential vector A and the electric potential V are similar to the criteria used for the former test case of an infinite conducting rod. The velocity was set to zero on the surfaces to satisfy the no-slip condition [14]. The inlet velocity was imposed so as to reproduce the desired shielding gas volume flow rate. The inlet velocity profile was simplified as it was assumed to be uniform, as if the flow was inviscid rather than viscous. The reason for this simplification was to be able to set the same condition for the 0° (with 2D axisymmetric mesh) and 12° tilt angle (with 3D mesh). The parabolic inlet boundary condition (that can be used for a viscous fluid) is programmed in C++ in OpenFoam for a 2D axisymmetric configuration but not for a 3D configuration. It was thus necessary to re-write this boundary condition in C++ for the 3D case (to be able to use a viscous inlet velocity condition for both the 0° and the 12° tilt case). It was decided that C++ programing was not part of this project. Thus the inlet velocity

boundary condition simplified assuming an inviscid fluid at the inlet. At the outlet (patches "OutletAnode" and "OutletNozzle") a zero-gradient velocity condition is assumed. This condition, which is justified if the outlet is far away enough, was assumed to be verified. The inlet gas is assumed to be at room temperature. The outlet is assumed to be far enough to set a zero temperature gradient. The conditions on the anode and cathode are very simplified since the anode and cathode layer are not included into the model. As the temperature boundary conditions are approximations the simulation model can provide quantitative rather than qualitative results. The temperature set on the anode and cathode tip (or base metal and electrode tip) is in reality the temperature inside the plasma at a very short distance from the surface. Because of lack of experimental data there temperatures were assumed to be uniform, to the values of Table 5. As all the current was assumed to flow through the cathode tip, a high temperature (allowing electric conduction) was imposed on the cathode tip, and a simple condition of no-temperature gradient was set on the rest of the electrode. The boundary condition for the pressure is the standard zero-gradient condition on all the surfaces. The shielding gas is assumed to enter the domain at atmospheric pressure (101325 Pa), and the outlet is assumed to be far away enough to be also at atmospheric pressure. The validity of the outlet conditions was checked in former studies [9], [11].

Table 5: The boundary conditions for patches in the 2D axisymmetric GTAW mesh

Patch:	A [Vs/m]	V [V]	U [m/s]	T [K]	p [Pa]
NozzleWall	(dA/dn)=0	(dV/dn)=0	0	(dT/dn)=0	(dp/dn)=0
InletSides	(dA/dn)=0	(dV/dn)=0	2.36	300	101325
CathodeWall1	(dA/dn)=0	(dV/dn)=0	0	(dT/dn)=0	(dp/dn)=0
CathodeWall2	(dA/dn)=0	(dV/dn)=0	0	(dT/dn)=0	(dp/dn)=0
TipWall	(dA/dn)=0	-18135	0	20000	(dp/dn)=0
anodeTipWall	(dA/dn)=0	0	0	7000	(dp/dn)=0
anodeWall	(dA/dn)=0	0	0	7000	(dp/dn)=0
anodeOutletWall	(dA/dn)=0	0	0	7000	(dp/dn)=0
OutletAnode	0	(dV/dn)=0	(dU/dn)=0	(dT/dn)=0	101325
OutletNozzle	0	(dV/dn)=0	(dU/dn)=0	(dT/dn)=0	101325

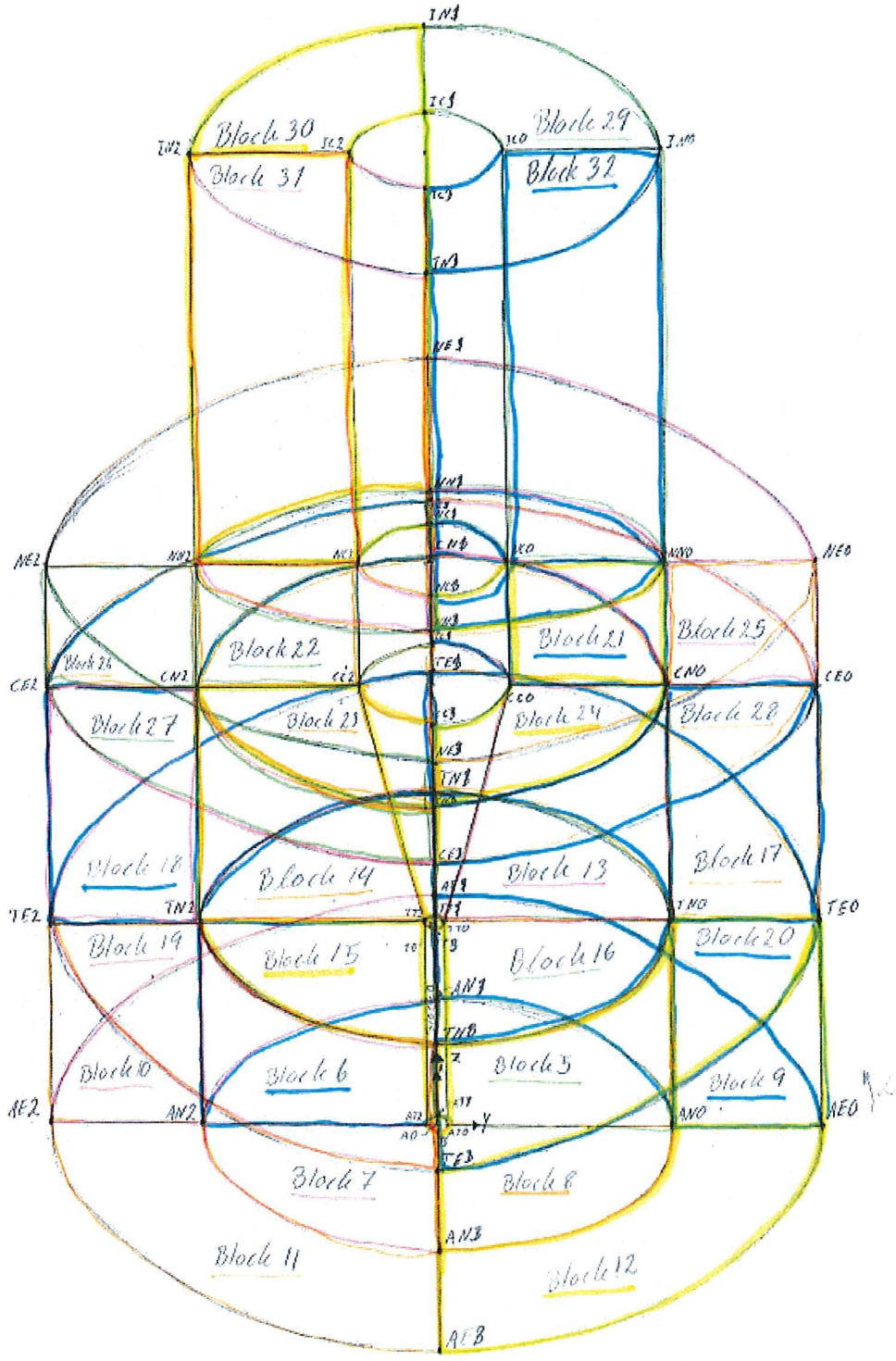


Figure 16: GTAW zero degree tilt; blocks of the whole 3D mesh

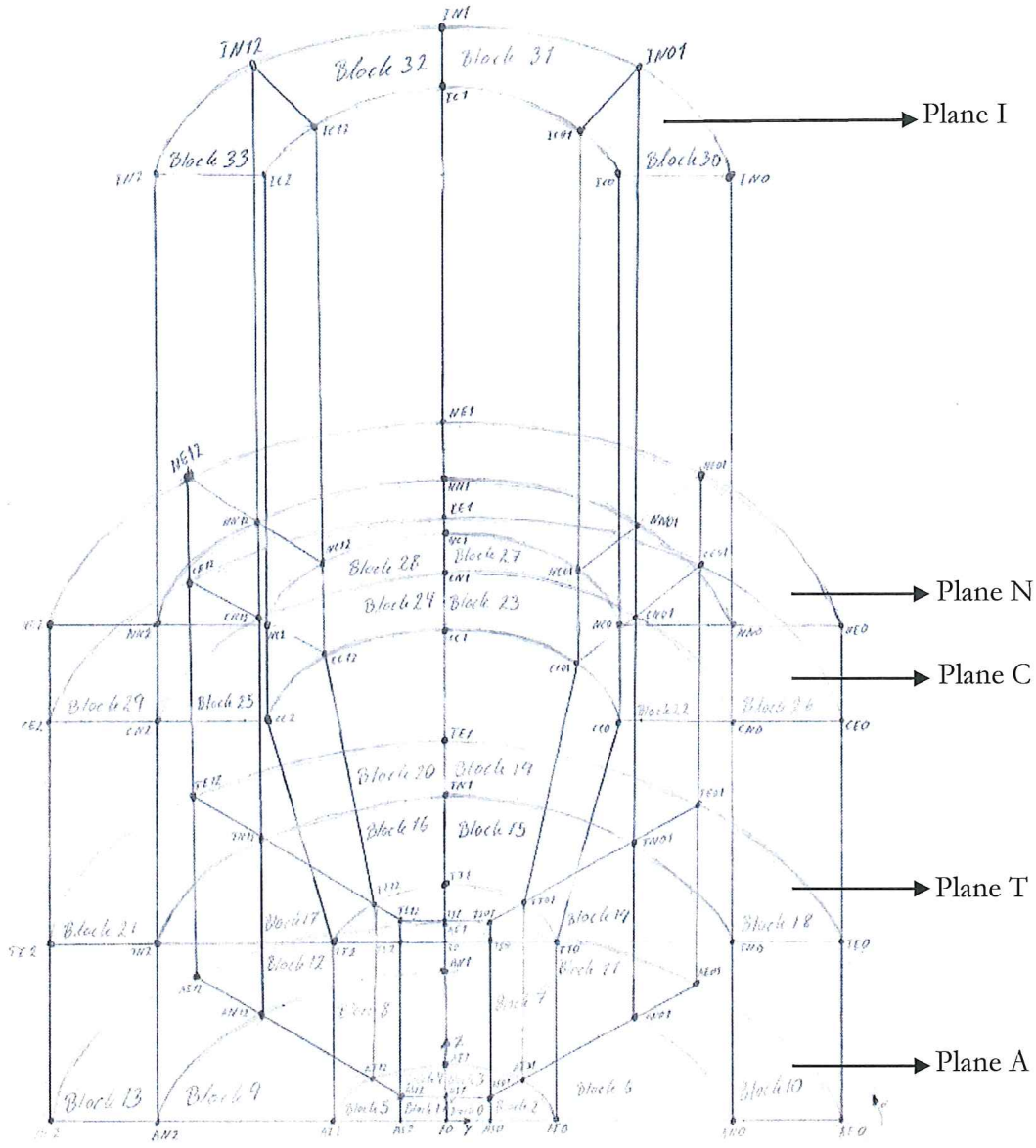


Figure 17: GTAW zero degree tilt half 3D mesh with blocks, not in scale

When further developing a 3D mesh based on the 2D axisymmetric mesh the wedge angle was first increased from 5° to 90°. As for the infinite conducting rod, the region between electrode tip and base metal was decomposed into several blocks, including a block of square section. The total number of blocks was thus increased from 8 for the 2D axisymmetric mesh to 33 for the 3D mesh.

The blocks of the 3D mesh are sketched in Figure 16. Here too, it was realized that CPU time could be saved by introducing a symmetry plane along the welding path and reducing the 3D mesh to a half space. This is possible since the electrode is tilted in the weld path plane.

A sketch (not in scale) of the half 3D mesh including nodes and blocks names can be seen in Figure 17 above. The number of blocks was increased by one block compared to the whole 3D mesh in Figure 16, in order to obtain the same type of square block in the middle of the domain as for the half 3D mesh of the infinite rod problem.

The nodes of the 3D meshes were renamed compared to the 2D axisymmetric mesh. The new notation includes two letters and a number as for the 3D meshes of the infinite rod problem. The first letter in the name of a node refers to a plane normal to the symmetry axis (called here the x-axis). The different planes can be seen in Figure 20: the anode surface plane (Plane A in x=0), the electrode tip plane (Plane T in x=2 mm,) the electrode shoulder (Plane C in x=4.42), the nozzle wall plane (Plane N in x = 5.52 mm) and the inlet plane (Plane I in x=10.52 mm). The second letter in the name of the node refers to the radial location of the node: S holds for nodes at the corners of the square block (r_s = 0.1 mm), T is used for nodes at the cathode tip radii (r_{tip} = 0.2 mm), C for node at the cathode radii (r_{cath} = 1.6 mm), N for nodes at the internal radii of the nozzle (x = 5 mm) and E for nodes at the external radii of the nozzle that is also the radii of the computational domain (x = 8.2 mm). Finally, the number in the name of a node refers to the angular position, exactly as for the infinite rod test case. These notations were used for writing the code for the 3D meshes (see section "vertices" in Appendix E). The coordinates x, y and z for the nodes were defined in the same way as for the 3D mesh of the infinite rod.

The increase in size between neighboring cells was also calculated for the mesh of the GTAW test case with zero degree electrode tilt. The data used in the calculations were the same as in Appendix F except for "tiltAngle".

Table 6: Increase per cell along the x-direction of the GTAW mesh calculated for each block.

Input:	L [10^{-3} m]	NOC [-]	ER [-]
Block 6	2	100	1
Block 14	2.42	121	1
Block 22	1.1	55	1
Block 30	5	150	2

Output:	IPC [-]	ERc [-]
Block 6	1	1
Block 14	1	1
Block 22	1	1
Block 30	1.0047	2

All the blocks delimited by the same two planes (such as blocks 30, 21, 32, 33 delimited by the planes N and A) have the same distribution in cell length along the x-direction. It is thus enough to consider the blocks 6, 14, 22 and 30 to check the increase in size along the x direction between neighboring cells (IPC) in all the blocks. The data given in Table 6 were calculated using the Matlab code of Appendix A. The largest gradients are expected in Block 6, 14 and 22 that are close to the cathode tip. The cells of these blocks are thus the smallest in size and have all have the same size per cell along the x-direction. So the IPC is equal to one in these blocks. Block 30 is

away from the large gradient region, allowing using larger cells. The input expansion ratio was set to 5. As desired, this resulted in a calculated increase in size (see Table 6) lower than the maximum criteria of 1.1.

Table 7: Calculated increase in size of adjacent cells at the interface between the blocks in the x-direction.

Output: Limit between	IPC [-]
Block 6 and Block 14	1
Block 14 and Block 22	1
Block 22 and Block 30	1.1549

The blocks 6, 14 and 22 have cells of same size along x, so the transition between these block is automatically correct. But the Block 22 and Block 30 do have cells of distinct size along x. The increase in size at the transition between these two blocks is calculated to be 1.1549, as can be seen in Table 7. This is slightly larger than the recommended value of 1.1 but still lower than 1.2 (a number also used by CFD users as recommended increase in size between neighboring cells). This result was accepted since the gradients are known to be low in this region of the computational domain. This can be seen (Figure 8 and 9 of [9]) in a published study done with a similar setup. This increase in size slightly larger than 1.1 but still reasonable saves a little bit of CPU time compared to a mesh with smaller cells to obtain an IPC within the preferred tolerances. It was thus decided to maintain the value if ER and IPC.

The increase in size per cells between neighboring cells was also calculated along the radial direction of the mesh.

Table 8: Increase per cell along the radial direction of the GTAW mesh calculated for each block.

Input:	L [10^{-3} m]	NOC [-]	ER [-]
Block 2	0.1	100	1
Block 6	4.8	270	2.3
Block 10	3.2	95	2.25
Block 22	3.4	270	2.3
Block 26	3.2	95	2.25

Output:	IPC [-]	ERc [-]
Block 2	1	1
Block 6	1.0031	2.3
Block 10	1.0074	2.25
Block 22	1.0031	2.3
Block 26	1.0074	2.25

As for the check done along the x-direction, the mesh is discretize in such a way that only few blocks need to be check along the radial direction, such as the blocks 2, 6, 10, 22 and 26. The results obtained using the Matlab code of Appendix A can be seen in Table 8. The largest gradients are expected to be in the vicinity of the electrode tip.

The cell size along the radial direction is thus the smallest in the central part of the domain ($r \leq r_{tip}$). It is also uniform in this region (e.g. Block 0, Block 2). For radial distances $r \geq r_{tip}$ the cell size along the radial direction is slightly increased with r. This applies to Block 6, 10, 22, and 26. The IPC thus needs to be check for these last blocks. It can be seen in Table 8 that the IPC value calculated for all these blocks is lower than the maximum criteria of 1.1. The only remaining check is thus the transition between blocks. The increase per cell between neighboring cells in the limits of the blocks was also calculated along the radial direction.

Table 9: Increase per cell calculation in the limit between the blocks in the radial direction.

Output: Limit between	IPC [-]
Block 2 - Block 6	1.1388
Block 6 - Block 10	0.8338
Block 22 - Block 26	1.1772

The maximum increase in size per cell in adjacent cells located in different blocks was located between Block 2 and Block 6 and equal to 1.1388 as can be seen in Table 9. This is a little bit higher than the recommended value 1.1, but compared to the cost of refining the mesh and the increase in CPU time this value was accepted. The blocks 6 and 10 must have the same inputs as block 22 and 26 except the length of Block 6 and Block 22, to obtain a consistent mesh geometrically. This caused a limitation for the increase in size within the recommended tolerance. The main focus for these two limits was to keep the increase in size between the neighboring cells in the different blocks lower than 1.2. This resulted in an increase in size between the neighboring cells in Block 6 and Block 10 of 0.8338 and in the limit between Block 22 and Block 26 of 1.1772.

The edges of the mesh were defined in the same way as in the infinite rod 3D mesh.
Most of the boundary conditions were set as for the 2D axisymmetric case. The patches "back", "front" and "symmetry axis" (see Figure 15) are specific for the 2D axisymmetric configuration defined on a 5° sector and are not present in the 3D mesh. On the contrary, a symmetry plane (identified in the mesh file with the name "halfModelPlane") is added.
They can be found in Table 1 and are thus not recalled here. Only the new boundaries introduced for the half 3D case are listed below. All the other patches have the same boundary conditions as in Table 5 but the number of block faces each patch contains is larger than for the 2D axisymmetric mesh.

4.3.2 Twelve degree tilt angle

The mesh of the 12° tilt test case is based on the mesh developed previously for a 0° tilt angle. The welding tool, including electrode and nozzle is rotated in the xy-plane (see Figure 18) around the center of the electrode tip denoted T* and located in the plane T in (x_T, 0, 0), see Figure 18. It implies that the nodes of plane A are kept unchanged while all the block nodes belonging planes T and located above plane T are concerned by the rotation.

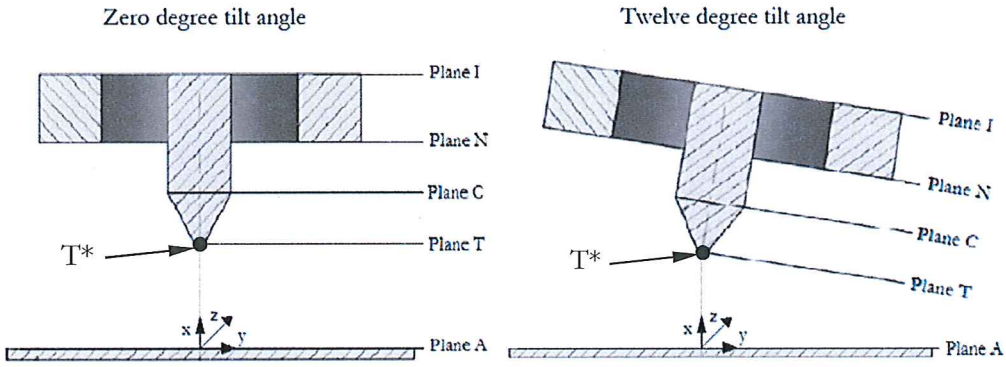


Figure 18: Sketch of the planes affected when tilting the electrode. Left: 0° tilt angle. Right: 12° tilt angle.

Consider a node NN located in or above Plane T and with the Cartesian coordinates (x_{NN}, 0, z_{NN}) in the mesh for a 0° tilt angle. This concerns for instance the nodes containing "0" or "1" in their name, e.g. NE1. When the electrode is tilted with a tilt angle β, the node NN rotates to the new position NN^* = (x^*_{NN}, y^*_{NN}, z^*_{NN}). As illustrated in Figure 19, the new coordinates (x^*_{NN}, y^*_{NN}, z^*_{NN}) are

$$x^*_{NN} = (x_{NN} - x_T) * \cos(\beta) + x_T \quad (25)$$

$$y^*_{NN} = (x_{NN} - x_T) * \sin(\beta) \quad (26)$$

$$z^*_{NN} = z_{NN} \quad (27)$$

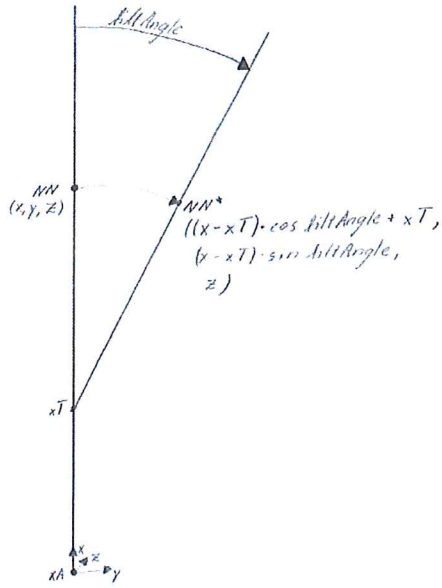


Figure 19: Sketch for a node NN = (x_{NN}, 0, z_{NN}) moved to NN* = (x^*_{NN}, y^*_{NN}, z^*_{NN}) after rotation by a tilt angle β.

Consider now the more general case of a node NN_2 located in or above Plane T and with the Cartesian coordinates (x_{NN2}, y_{NN2}, z_{NN2}) in the mesh for a 0° tilt angle, where y_{NN2} can now differ from zero. This concerns the nodes with a number in their name that is "0" or "1". When the electrode is tilted with a tilt angle β, the node NN_2 rotates to the new position NN_2^*=(x^*_{NN2}, y^*_{NN2}, z^*_{NN2}). As illustrated in Figure 20, the new (x^*_{NN2}, y^*_{NN2}, z^*_{NN2}) coordinates are

$$x^*_{NN2} = (x_{NN2} - x_T) * \cos(\beta) + x_T - y_{NN2} * \sin(\beta) \quad (28)$$

$$y^*_{NN2} = (x_{NN2} - x_T) * \sin(\beta) + y_{NN2} * \cos(\beta) \quad (29)$$

$$z^*_{NN2} = z_{NN2} \quad (30)$$

The new term on the right and side of (28) and (29) is due to the vertical motion along one side of the right-angled triangle (see red arrow in Figure 20). This term is negative when y_{NN2} is positive, e.g. for node NN2, and positive when y_{NN2} is negative, as can be seen in Figure 20.

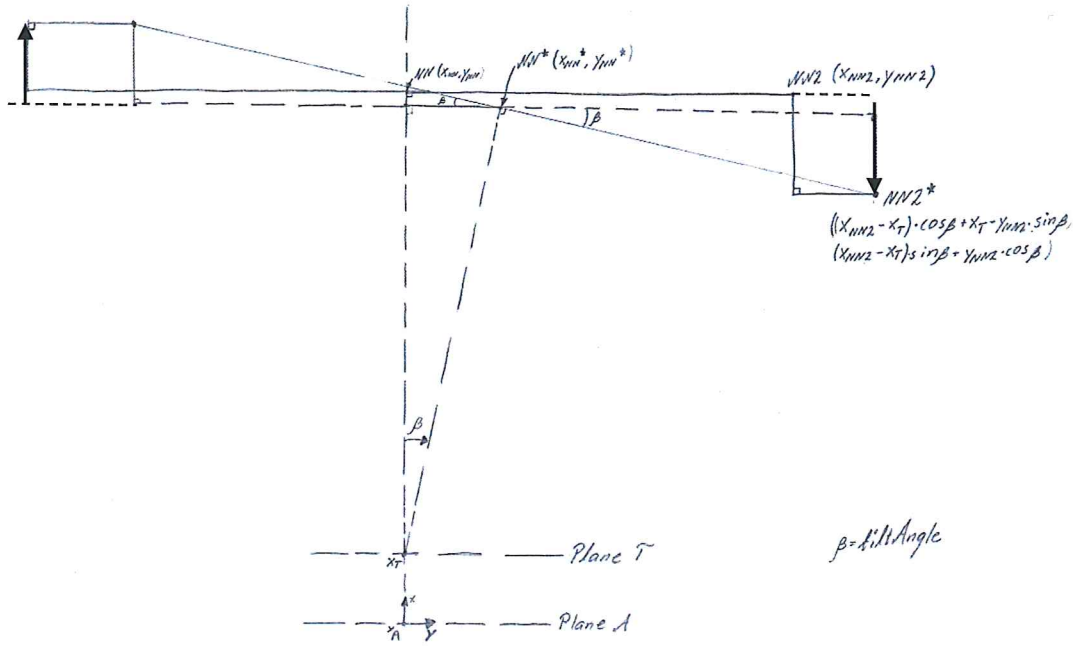


Figure 20: Sketch for a node $NN_2 = (x_{NN2}, y_{NN2}, z_{NN2})$ with $y_{NN2} \neq 0$, moved to $NN^*_2 = (x^*_{NN2}, y^*_{NN2}, z^*_{NN2})$ after rotation by a tilt angle β

The nodes and blocks of the 12° tilt mesh were thus obtained rotating the plane T, C, N and I of the original 0° tilt mesh about the rotation centre T*. The resultant blocks are sketched in Figure 21. But this did result in increased cell skewness in some blocks, such as bocks 10 and 26 (see Figure 17). To avoid this problem, the coordinates of the boundary nodes of plane T and C were adjusted to be at the same radial distance as the boundary nodes of plane A. This can be seen comparing the coding of e.g. node NE1 in Appendix E and in Appendix F. This rewriting resulted in the mesh blocks sketched in Figure 22. The parameters NOC and ER used to discretize the blocks of Figure 22 are the same as for the 0° tilt mesh. These parameters can be found in section 4.3.1 and are not reported again here.

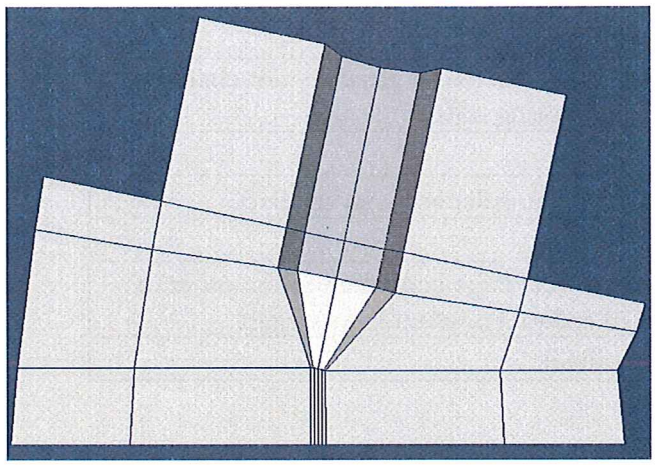


Figure 21: Sketch of the blocks obtained when rotating the of plane T, C, N and I of the original 0° tilt mesh according to equations (26)-(28).to do the 12° tilt mesh.

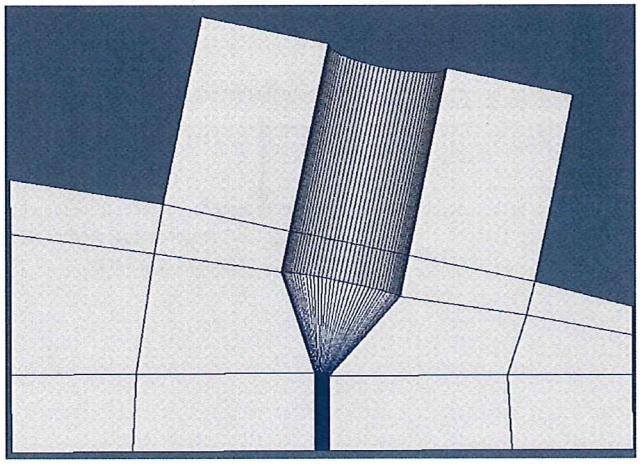


Figure 22: Sketch of the blocks for the 12° tilt case with after adjustment of the boundary nodes of plane C and N.

4.4 Calculation strategy

No issues occurred when running the calculation of the infinite rod. But the calculation of the GTAW test cases was done by Choquet. The reason is that this test case presents very large source terms and very stiff gradients. Starting the calculation is thus difficult. Test cases with stiff gradients or complex physics are often difficult to run or to start running. Moreover, this uses to be problem dependent. So understanding how to proceed when starting running a simulation for a new problem can take some time. Choquet shared her experience on how to make the calculation run for a steady state problem. The risks of facing difficulties when starting a calculation are greater for problems with e.g. turbulence, or a very high velocity such as a supersonic flow, or large source terms as may happen in the presence of chemical reactions. This is due to the fact that a too large gradient or a too large source term may cause division by zero when the computer inverts the Jacobian matrix.

Table 10: Some clues for starting a difficult calculation

1.	Check that the boundary conditions are correct
2.	Check that the initial data is not too rough, i.e. that it has does not changes very much in magnitude between neighboring cells
3.	Decrease the relaxation factors
4.	Change from second order accuracy to first order accuracy in space
5.	Check that the mesh is okay.
6.	Find out if any of the source terms are very large compared to the other of the terms the equation, and if needed reduce temporarily its weight
7.	Simplify temporarily the physics involved

Some clues for starting a difficult calculation can be seen in Table 10. Concerning the GTAW test cases two difficulties were faced. First a too rough initial field when the

temperature in the computational domain is initialized at room temperature while the electrode tip temperature is 20000K. In that case the calculation does not even run a single iteration. And second the energy source term due to Joule heating is so large when starting the calculation that the temperature becomes larger than 30000K, which is the maximum temperature in the data tables for thermodynamics and transport properties. The strategy developed for starting the calculation did consist in working with the points 2, 3, and 6. Concerning point 2, a hot column of gas was initialized in the volume bellow the electrode tip up to the cathode, as sketched in Figure 23 . This has no impact on the converged solution as the problem is steady. Concerning point 3, small relaxation factors were used at start for the fluid system of equations (11)-(13). These factors were then progressively increased as indicated in Table 10. Finally, concerning point 6, the Joule heating was artificially reduced by a factor 100 and then gradually increased to its normal value, as summarized in Table 10.

Table 11: Strategy used by Choquet and Sass-Tisovskaya to start the GTAW calculations.

Iterations from:		0	50	150	200	300
Relaxation factors of	Enthalpy	0.3	0.3	0.3	0.5	0.7
	Velocity	0.3	0.3	0.3	0.5	0.7
	Pressure	0.1	0.1	0.1	0.3	0.5
Weighting factor for	Joule heating source term	0.01	0.1	1	1	1

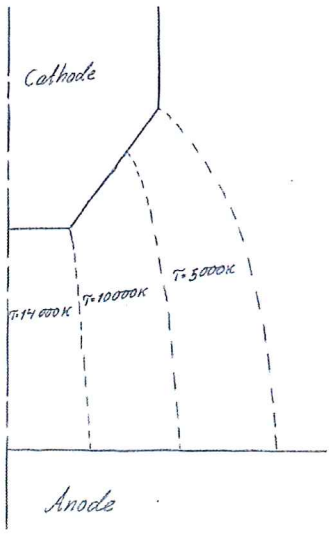


Figure 23: A sketch of hot column used as initial condition to help the calculation start.

The calculation for the GTAW test case with 0° electrode tilt and 2D axisymmetric mesh was run following this strategy. The 2D axisymmetric mesh contains 151120 cells. The calculation for the GTAW test case with 12° electrode tilt and a half 3D mesh could not be run, not even with this strategy. The reason was not related to the

strategy for starting the calculation but to the memory available on the computer. The configuration available for running OpenFOAM on the cluster allowed 4 cores, and this was not enough. An alternative would have been to run the calculation on another computer where this limitation did not exist, but during this thesis this was not an available alternative. The other alternative was to decrease the load imposed on the memory. The load is related to the number of cells the mesh contains and to the number of equations of the model. As can be seen in section 3.1 and 3.3 of this thesis the number of equations is nine. The half 3D mesh described in section 4.3.2 of this thesis contains more than 5 million cells (more precisely 5709600 cells) and could be started. . The number of cells specified for each parameter in Appendix F was then reduced dividing the initial values by two, as indicated in Table 12. The total number of cells for the half 3D mesh was then 718740. The calculation was then possible to start.

Table 12: Number of cells for each parameter changed to try to make the calculation run.

Parameter:	Number of cells initial	Number of cells reduced
xTipNumberOfCells	100	50
xCathode2NumberOfCells	121	61
xCathode1NumberOfCells	55	28
xCathodeInletNumberOfCells	150	75
rTipNumberOfCells	10	5
rCathodeNumberOfCells	270	135
rOutletNumberOfCells	95	48

5 Simulation results and discussion

Simulations were done for the test cases of the infinite electric rod and the GTAW with and without tilt angle. The infinite rod simulations results were compared to the analytical solution of section 3.2. The simulation results of the GTAW test case with a 12° electrode tilt angle (half 3D mesh)was compared to the simulation results of the GTAW test case with 0° electrode tilt angle (2D axisymmetric mesh).

5.1 Infinite electric rod

For the infinite electric rod, the simulations were done comparing the test cases 1, 2, 3 and 4 with the analytical solution of section 3.2. The test cases 1 to 4 differ by the number of discretization cells, see Table 2. Test case 1 has the largest cells and test case 4 the most refined. Convergence could be reached in 2 iterations with final residuals lower than 10^{-8}. The results obtained with the test case 4 are not reproduced here since they cannot be distinguished from the results of test case 3.

The current density along the radial and axial direction is plotted in Figure 24 from the second test case. It should be noticed that the current density calculated for the test cases 1 and 3 are the same. They are thus not reproduced here. As expected for this problem, the calculated current density component along the axial direction is only function of the radial position. The current density is piecewise constant, as shown by the line with plus symbols in Figure 24. It is equal to 1.9e+08 A/m^2 in the rod that is from r=0 m to r=1.0e-03 m, which is due to a uniform electric conductivity of 2700 A/Vm in the rod. For radii larger than 1.0e-03 m the current density is zero due to the very low electric conductivity of 10^{-5} A/Vm in this region of the computational domain. The calculated current density component along the radial direction is zero everywhere in the computational domain, as shown by the line with filled squares in Figure 24. This means that, as expected for this problem, all the calculated current is flowing inside the rod and only along the axial direction of the rod.

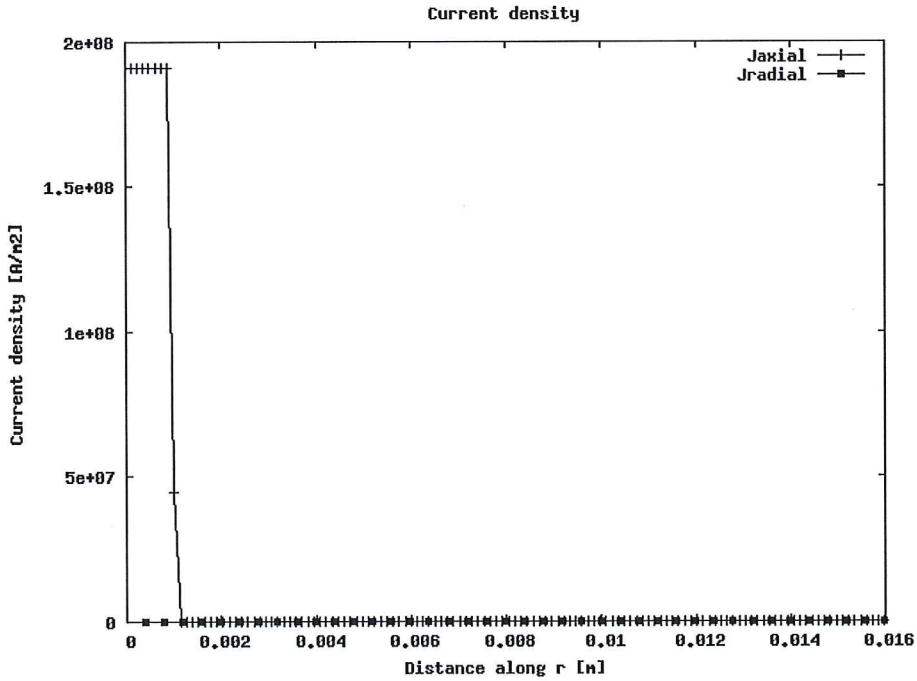


Figure 24: Current density axial and radial from test case 2.

The magnetic potential vector A is plotted in Figure 25 for the test cases 1 (unfilled square), the test case 2 (black filled square), and the analytic solution (plus symbol). It can be seen that the results calculated for the second test cases are in good agreement with the analytical solution. The results calculated for the first test case, with fewer cells along the radial direction, differ slightly from the other two. This indicates that the mesh of test case 1 may be too rough.

The magnetic potential vector A is plotted in Figure 26 for the test cases 2 (black filled square), the test case 3 (unfilled square), and the analytic solution (plus symbol).

The third test case has the most refined cells along the radial direction. It can be seen that both of the calculated results are in very good agreement with the analytical solution. This indicates that the mesh of test case 3 may be uselessly refined. This last point needs to be further investigated looking now at the calculation results obtained for the magnetic field.

The magnetic field B is plotted in Figure 27 for the test cases 1 (unfilled square), the test case 2 (black filled square), and the analytic solution (plus symbol). It can be seen that the first test case fails to reproduce the maximum value of the magnetic field at the radial distance equal to the rod radius r = 0.001 m. This confirms that the mesh of test case 1 is too rough. The second test case underestimates very slightly the maximum value of the magnetic field compared to the analytic solution. However, the difference is almost negligible.

The magnetic field B is plotted in Figure 28 for the test cases 2 (black filled square), the test case 3 (unfilled square), and the analytic solution (plus symbol). It can be seen that calculated results of test case 3 are in very good agreement with the analytical solution, and is very close to the maximum value of B in r = 0.001 m. Also the results of test case 2 and 3 are almost the same, except a small underestimation in case 2 compared to case 3 when r = 0.001 m. The amplitude of this estimation is now investigated in more detail.

The increase in maximum value of the magnetic field when refining the mesh can be seen in Table 13. The increase from the first to the second test case is very significant: 17.29 %. From the second test case to the third test case the increase is only 0.89 %, this is lower than 5 % so the second test case is considered to be enough accurate for the purpose of this work.

To conclude, these results mean that the mesh of test case 1 has too large cells to reach a good convergence in mesh. The meshes of test case 3 and 4, which are the most refined, do not provide any significant improvement of the solution compared to test case 2 while they require more computer memory and some more calculation time. They are thus uselessly refined. Finally the mesh of test case 2 is a good compromise for this problem; it has cells small enough to reach a satisfying convergence in mesh.

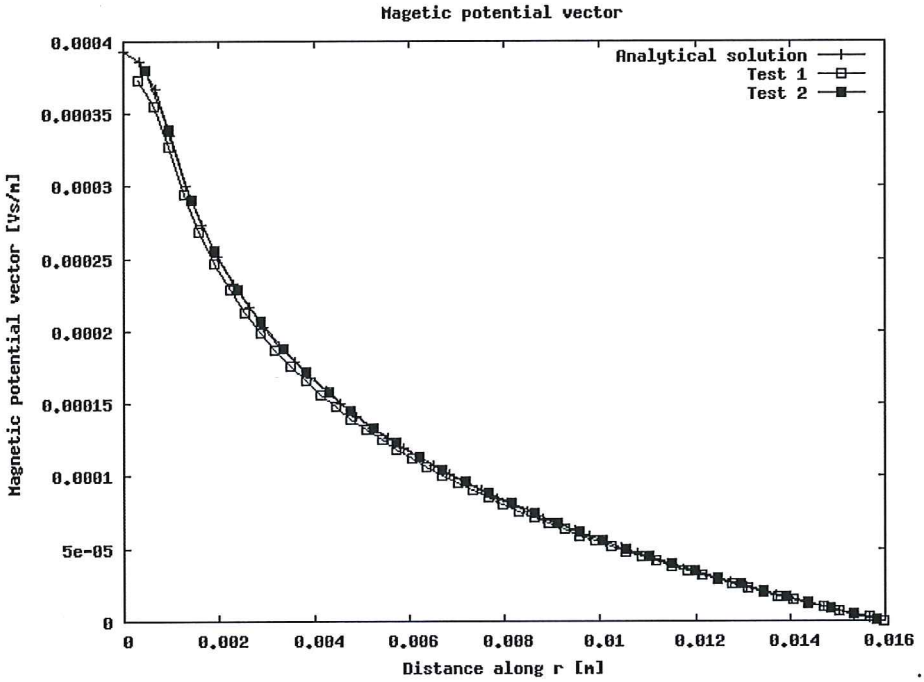


Figure 25: The magnetic potential vector from simulation of test cases 1 and 2 compared to the analytical solution.

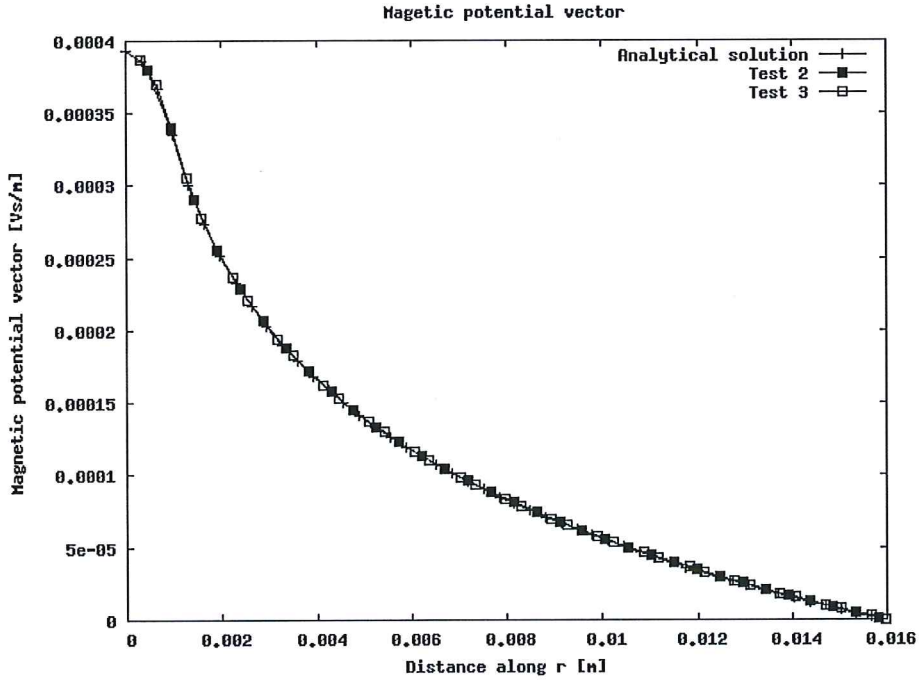


Figure 26: The magnetic potential vector from simulation of test cases 2 and 3 compared to the analytical solution.

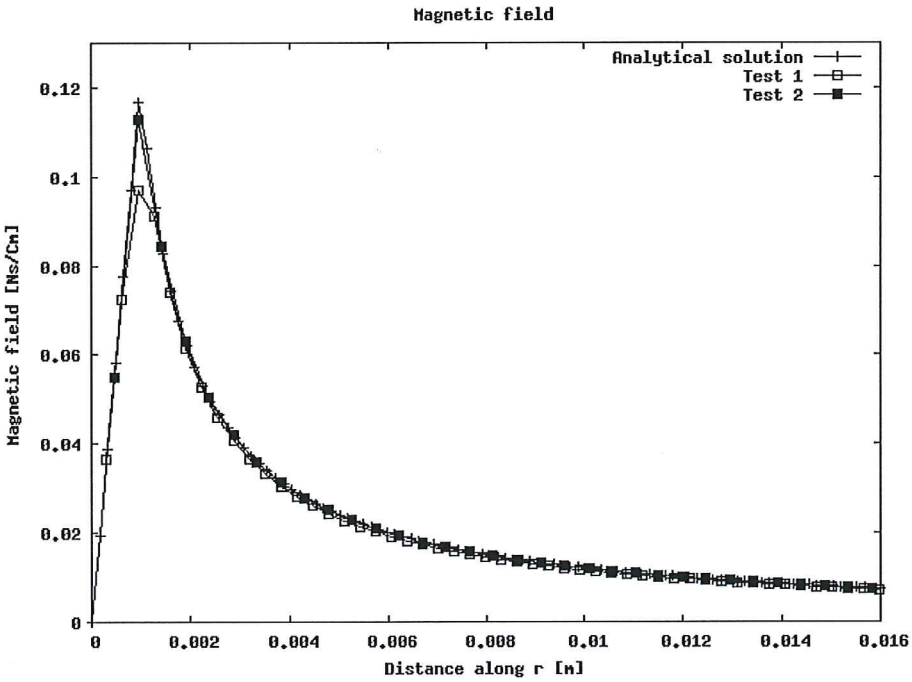


Figure 27: The magnetic field from simulation of test cases 1 and 2 compared to the analytical solution.

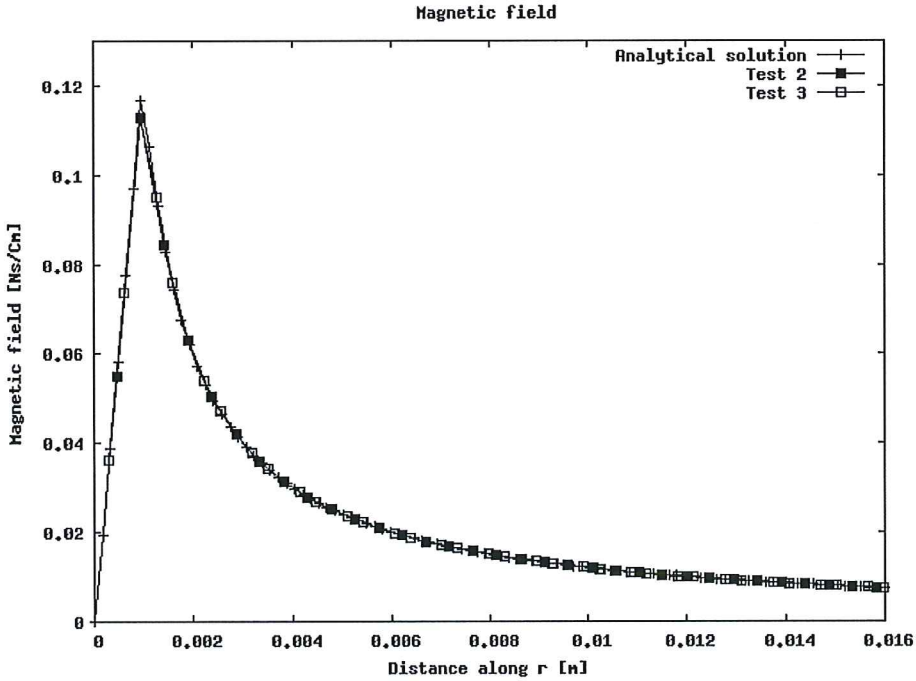


Figure 28: The magnetic field from simulation of test cases 2 and 3 compared to the analytical solution.

Table 13: Maximum magnitude of the magnetic field B_{max} , and increase in percent of B_{max} to the next test case for all the test cases and for the analytic solution.

Test case:	B_{max} [Ns/Cm]	Increase in percent to the next test case [%]
1	0.100037	17.29
2	0.117629	0.89
3	0.118502	0.15
4	0.118678	1.30
Analytical	0.120218	-

5.2 GTAW

As explained in section 4.4, the GTAW test cases were more difficult to run than the previous infinite conducting rod test cases. The residuals were monitored while the calculations were running.

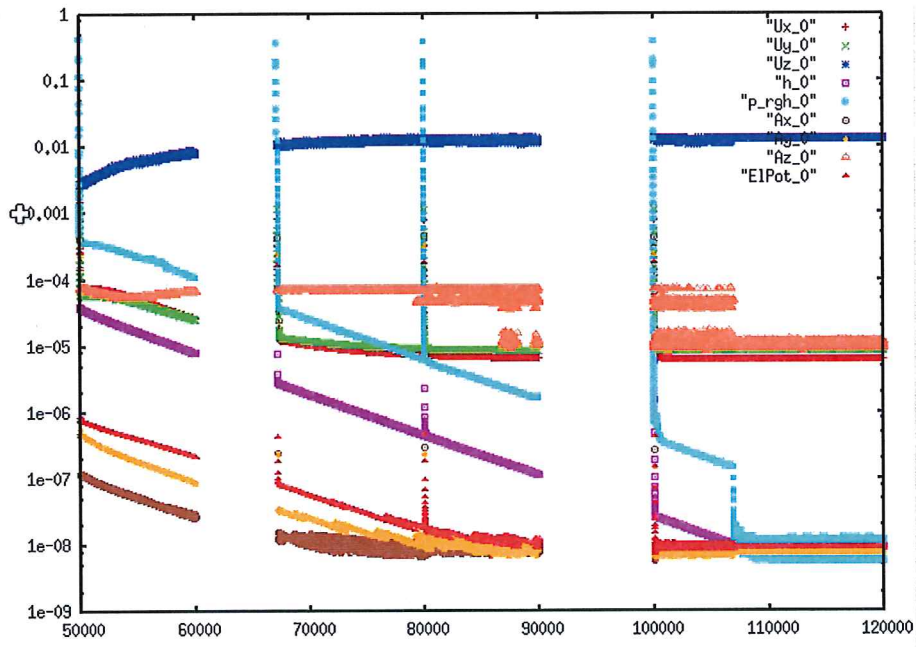


Figure 29: Initial residuals for the 2D axisymmetric simulation case with 0° tilt, from iteration 50000 to 120000.

The initial residuals of the 0° tilt case (with 2D axisymmetric mesh) are plotted in Figure 29 from iteration 50000 to iteration 120000. They are plotted for each of the variables involved in the magneto hydrodynamic system of equations: the velocity components Ux, Uy, Uz, the pressure p_rgh, the enthalpy h, the electric potential ElPot, and the components of the magnetic potential Ax, Ay and Az. It should be noticed that the calculation started at iteration 50000 since it was initialized using as input data the solution of another calculation. The calculation was started using the procedure described in Table 11 (from iteration 50000 to 50300 rather than 0 to 300). Then the numerical parameters were kept unchanged to run further, first from

iteration 50300 to 60000, then run 10000 iterations 4 times, and finally 20000 iterations. It can be seen in Figure 29 that the residuals are not plotted between iteration 60000 to 70000 and 90000 to 120000. This is simply because it was forgotten to save the corresponding data files for plotting. For the last iterations it can be seen that the curves are horizontal which indicates convergence.

By plotting now at the final residuals during the last 20000 iterations, the magnitude of the residuals can be obtained. They are plotted in Figure 30. It can be seen that at the last iteration (120000) all the quantities except the velocity residuals Uz (along the z-direction) have a magnitude of 10^{-8} or less. 10^{-8} was the accuracy imposed in the calculation setup. The reason why the velocity along the z-direction has final residual larger than 10^{-8} is that the magnitude of the velocity along the z-direction, which should be zero in this test case, is not exactly zero in the computer. It is lower than the computer accuracy and oscillates between 10^{-15} and 10^{-19}. It results in a magnitude of the oscillations from iteration to iteration equal to up to 10^{-4}, and results in residuals larger than 10^{-8}.

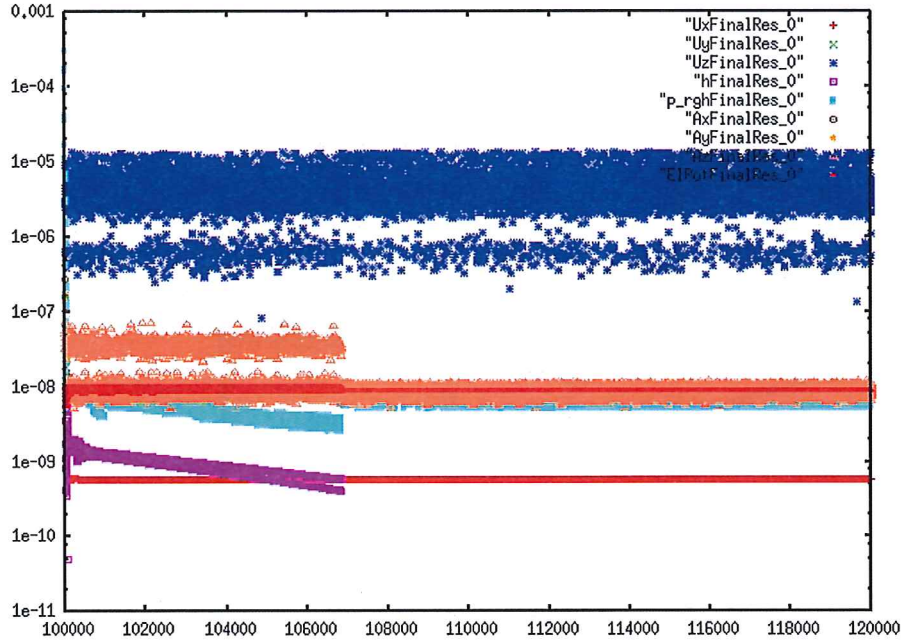


Figure 30: Final residuals for the 2D axisymmetric simulation case with 0° tilt, from iteration 100000 to 120000.

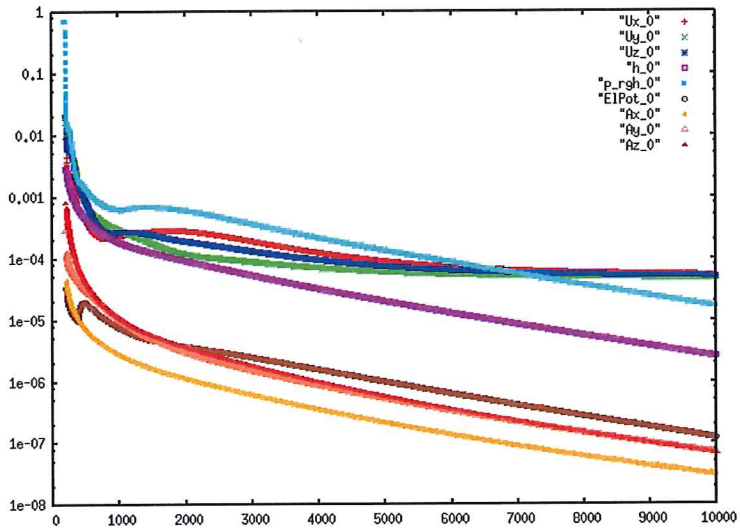


Figure 31: Initial residuals for the GTAW simulation case with 12° tilt, from iteration 1 to 10000.

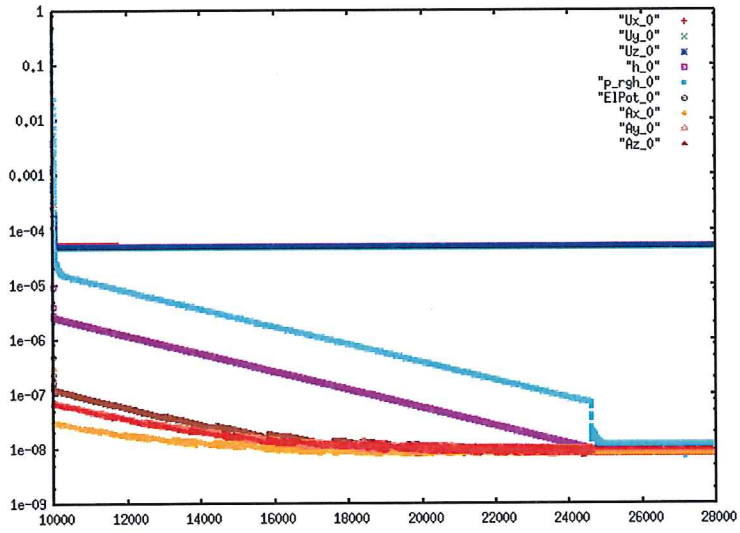


Figure 32: Initial residuals for the GTAW simulation case with 12° tilt from iteration 10000 to 28000.

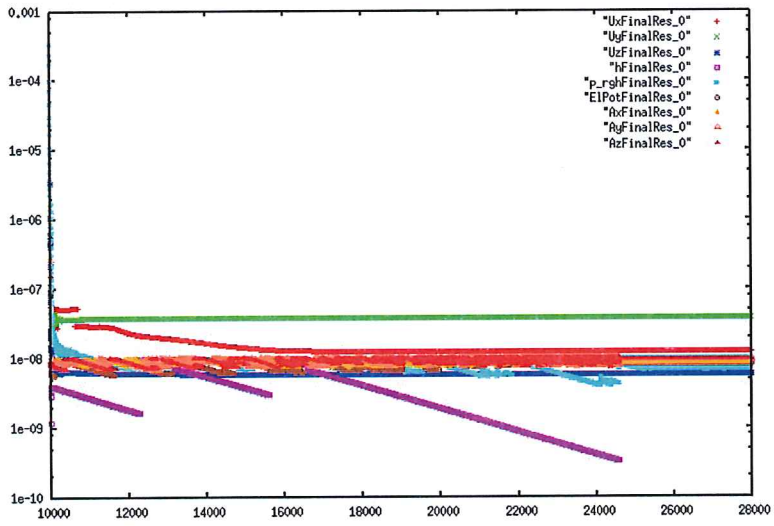


Figure 33: Final residuals for the GTAW simulation case with 12° tilt from iteration 10000 to 28000.

The initial residuals calculated for the half 12° tilt test case (with half 3D mesh) are plotted for each of the variables involved in the magneto hydrodynamic system of equations in Figure 31 for iteration 1 to 10000, and Figure 32 for the iteration 10000 to 28000. The calculation was started using the procedure described in Table 11 (from iteration 0 to 300). Then the numerical parameters were kept unchanged to run further, first from iteration 300 to 10000, and then run 18000 more iterations. For the last iterations it can be seen that the curves are horizontal which indicates convergence. The magnitude of these converged residuals can be obtained by studying the final residuals.

The final residuals from iteration 10000 to 28000 are plotted in Figure 33. At the last iteration it can be seen that the final residuals except the velocity along the y-direction have a magnitude about 10^{-8} (and 7 10^{-8} for the velocity along the y direction) which is accepted.

The temperature distribution obtained for the 0° tilt case (2D axisymmetric mesh) can be seen in Figure 34. The maximum temperature of 20597 K is observed at a short distance below the tip of the electrode. It can be checked that the temperature at the argon shielding gas inlet is about 300 K, which is consistent with the boundary condition in Table 5.

The isothermal lines are highlighted in Figure 35 where the region close to the tip of the electrode is zoomed. The plotted isothermal lines correspond to 10000 K, 12000 K, 14000 K, 16000 K, 18000 K and 20000 K. The outmost isothermal line has a magnitude of 10000 K and the innermost (closes to the tip) corresponds 20000 K.

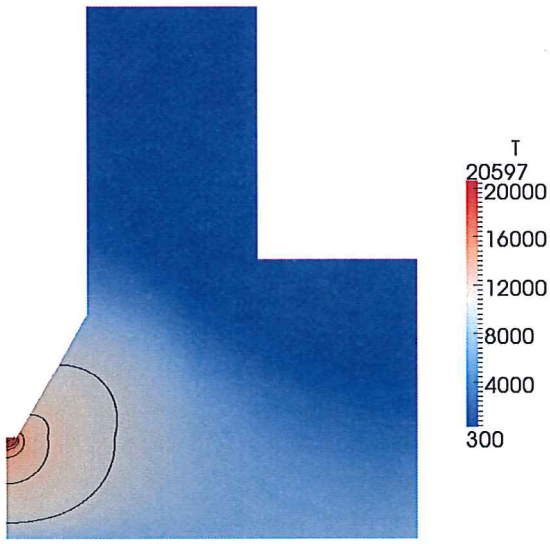


Figure 34: Temperature distribution obtained for the 0° tilt test case.

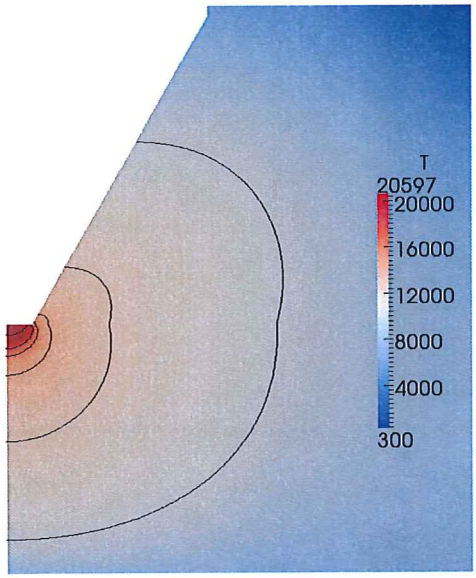


Figure 35: Temperature distribution obtained for the 0° tilt test case zoomed close to the electrode tip.

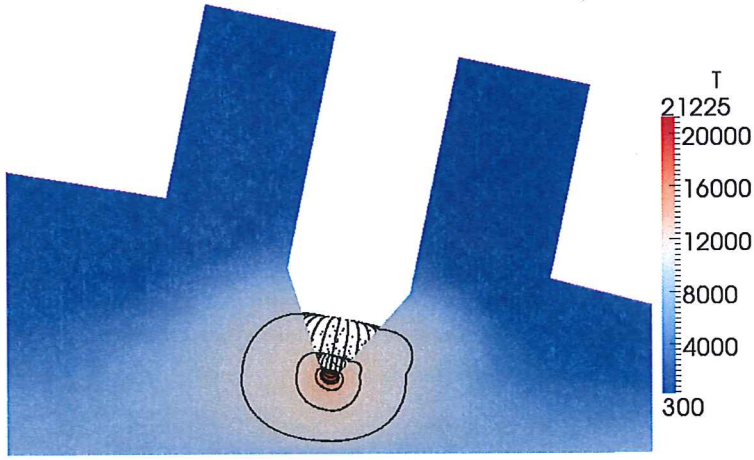


Figure 36: Temperature distribution obtained for the 12° tilt test case.

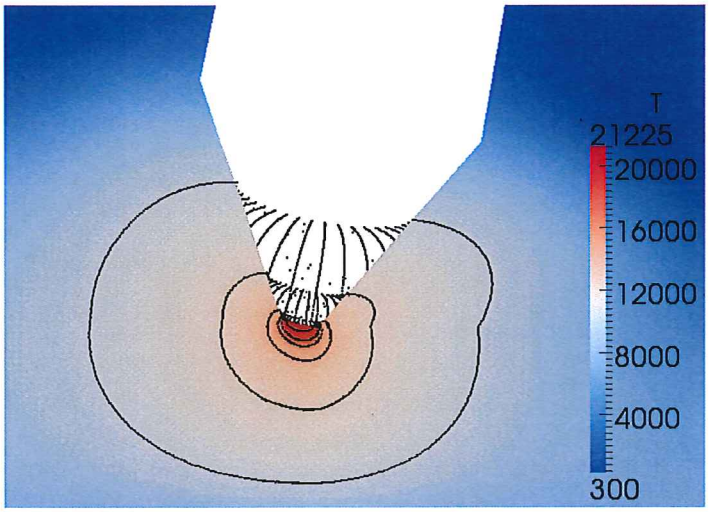


Figure 37: Temperature distribution obtained for the 12° tilt test case zoomed close to the electrode tip; visualized from the symmetry plane.

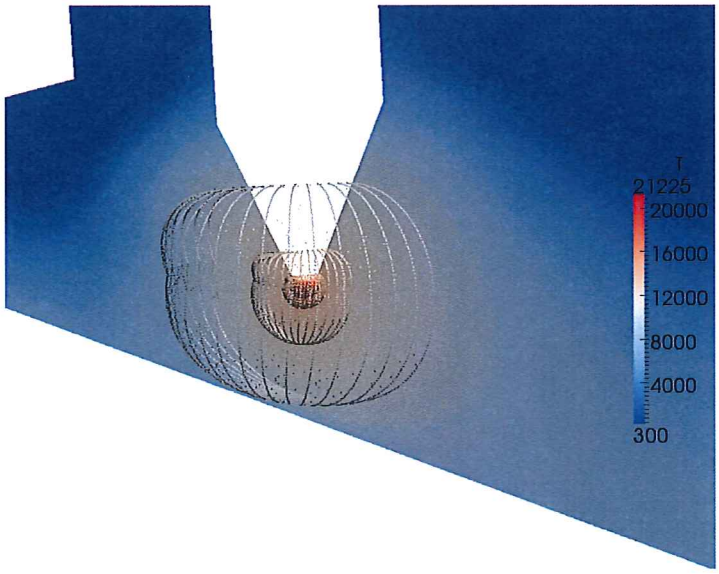


Figure 38: Temperature distribution obtained for the 12° tilt test case zoomed close to the electrode tip; visualized from the opposite side of the arc compared to Figure 39.

The velocity of the simulations was also investigated. For the 0° tilt case, the maximum velocity is observed between the tip of the electrode and the base metal. It is perpendicular to the base metal, points towards the base metal, and has a magnitude of 32 m/s (as see the red arrow in Figure 40).

For the 12° tilt case the maximum velocity is also observed between the tip of the electrode and the base metal. But its direction is perpendicular to the tip surface rather than perpendicular to the base metal surface. It points towards the base metal and has a magnitude of 38 m/s, as shown by the red arrow in Figure 41. It can be seen that the flow field is asymmetric and thereby different from the velocity calculated for the 0° tilt test case.

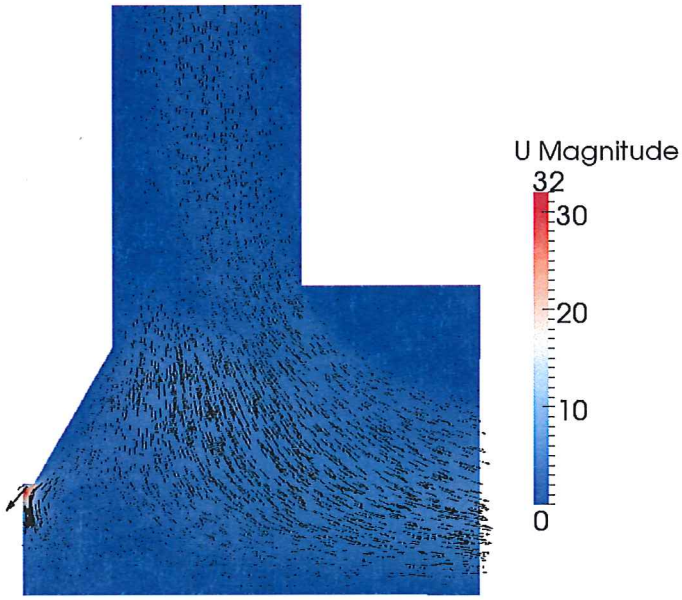


Figure 40: Velocity obtained by the 2D axisymmetric simulation zoomed in the region around the electrode tip.

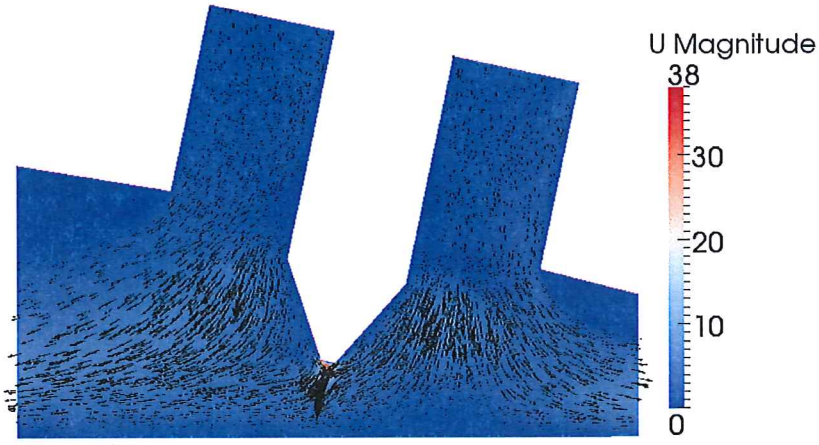


Figure 41: Velocity obtained by the 3D simulation zoomed in the region around the electrode tip.

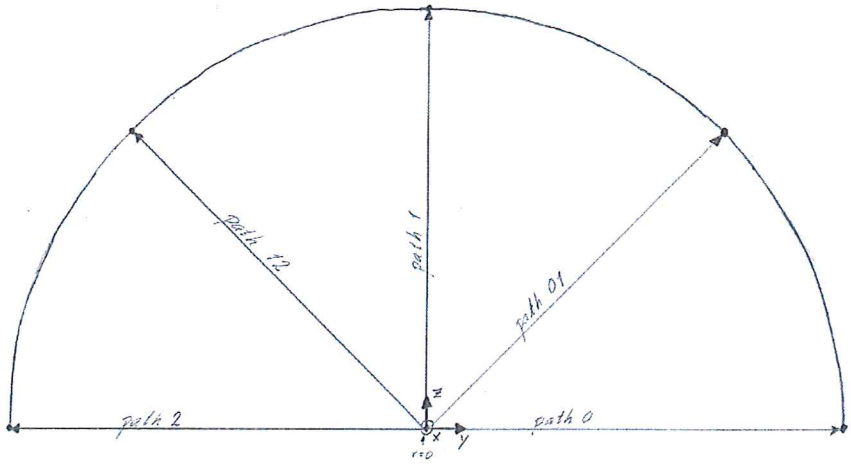


Figure 42: Location of the paths used when plotting heat flux of the 3D simulation.

The heat flux to the base metal was investigated 0.2 mm from the base metal into the computational domain. It was plotted along paths in the yz-plane located in x = 0.2 mm. The paths can be seen in Figure 42. The heat flux plots are in Figure 43 to Figure 47; more plots can be found in Appendix G.

The heat flux towards the base metal (so along the x direction) for the 0° tilt case (plus symbols) and for the 12° tilt case along the different paths (defined in Figure 42) are plotted in Figure 43, zoomed from r = 0 mm to r = 0.8 in Figure 44 and zoomed for r = 2 mm to r = 6 mm in Figure 45. It can be seen that for the 12° tilt case the heat affected area is the widest in front of the electrode (along path 2) and the shortest behind the electrode (along path 0). Along all the other paths intermediate values are obtained. The width of the heat affected zone indeed decreases when moving from the front of the electrode (i.e. path 2) towards the back of the electrode (i.e. path 0). For the 0° tilt case the extent of the heat affected zone is between the extent along path 0 and along path 2; this may be more easily observed in Figure 46 and Figure 47. It is closest to the results of the 12° tilt case along path 1 which is reasonable as path 1 is perpendicular to the weld path and perpendicular to the plane in which the electrode is tilted. (see Figure 43, as well as the Figures of Appendix G).

A zoom of the region from the center of the domain up to the radial distance of 0.8 mm is plotted in Figure 44 and Figure 47. A larger heat flux is observed in the region behind the electrode compared to the regions in front of the electrode. This is reasonable since the region behind the electrode is more closed while the region in front of the electrode is more open with a 12° electrode tilt than a 0° tilt angle. The 0° tilt solution is between the 12° tilt solution along paths 0 and path 2. A comparison between the 0° tilt angle calculation result and the 12° tilt case along path 1 can be seen in Figure 48 and in Appendix G. The results are very close, which again is reasonable as path 1 is perpendicular to the weld path and perpendicular to the plane

in which the electrode is tilted. The 0° tilt case reaches a slightly larger heat flux than the 12° tilt case along path 1.

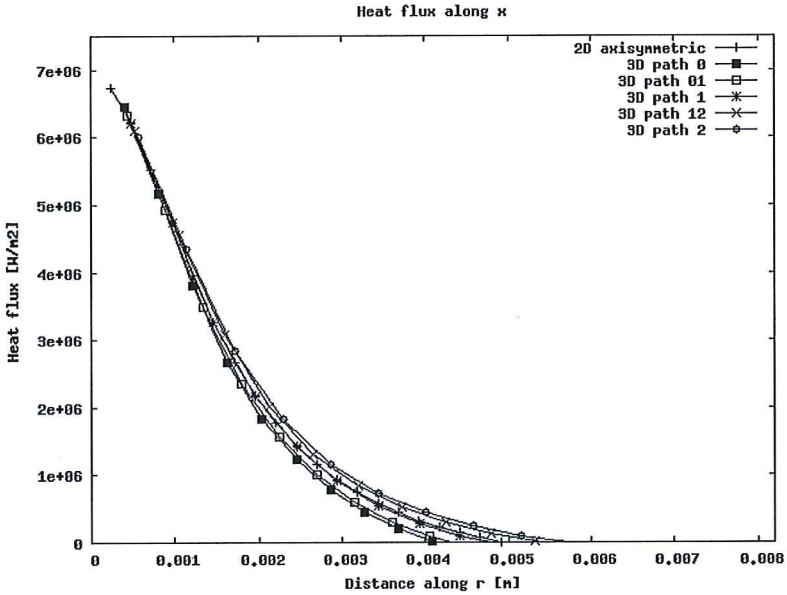


Figure 43: Heat flux calculated for the 0° tilt case, and for the 12° tilt case along the different paths of Figure 42.

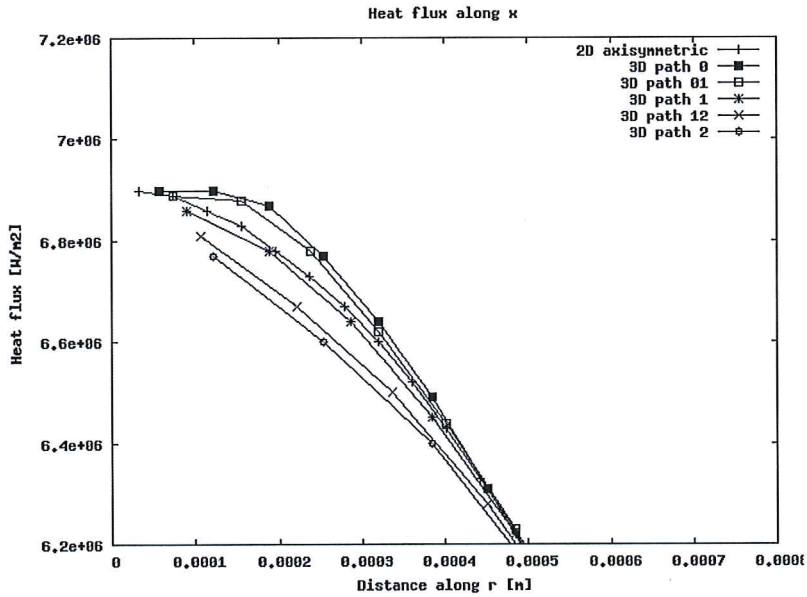


Figure 44: Zoom for r = 0 mm to r = 0.8 mm of the heat flux calculated for the 0° tilt case, and for the 12° tilt case along the different paths of Figure 42.

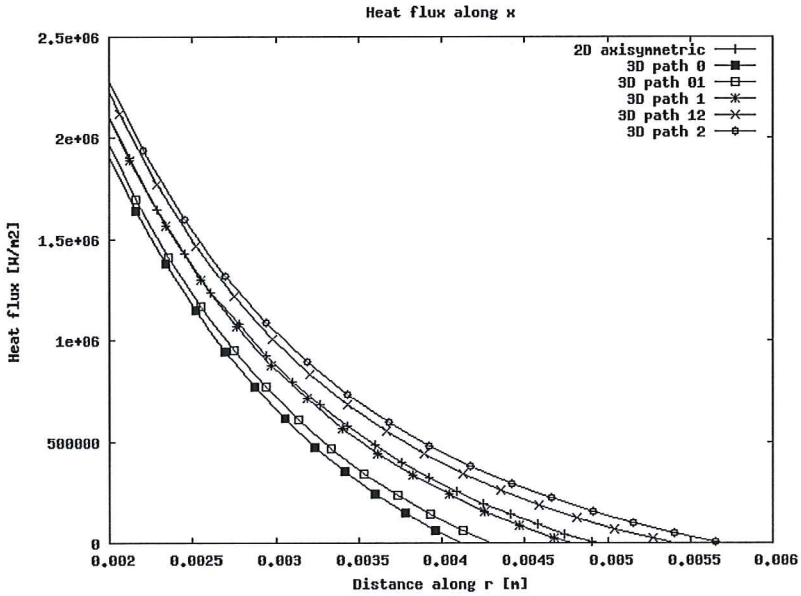


Figure 45: Zoom for r = 2 mm to r = 6 mm of the heat flux calculated for the 0° tilt case, and for the 12° tilt case along the different paths of Figure 42.

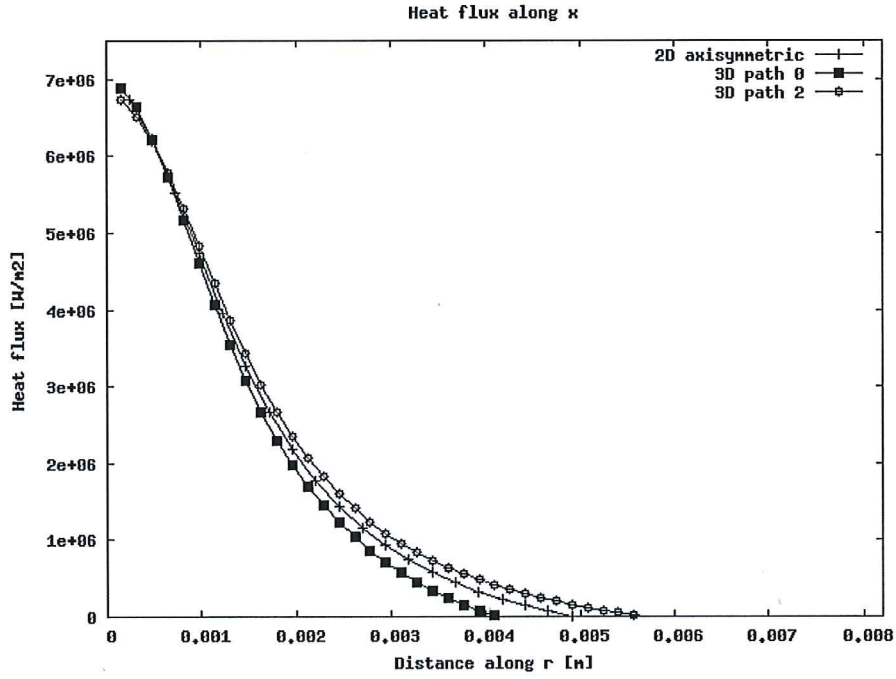


Figure 46: Heat flux calculated for the 0° tilt case, and for the 12° tilt case along the paths 0 and 2 of Figure 42.

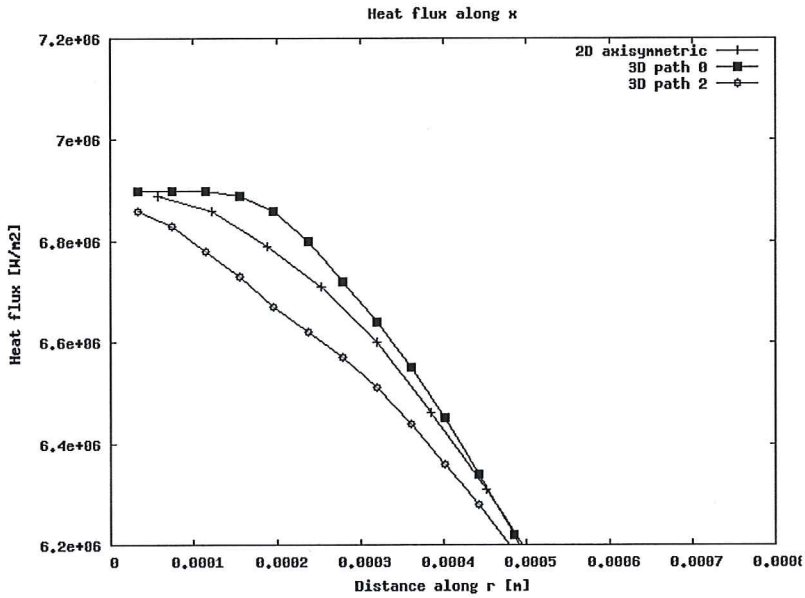


Figure 47: Zoom for r = 0 mm to r = 0.8 mm of the heat flux calculated for the 0° tilt case, and for the 12° tilt case along the paths 0 and 1 of Figure 42.

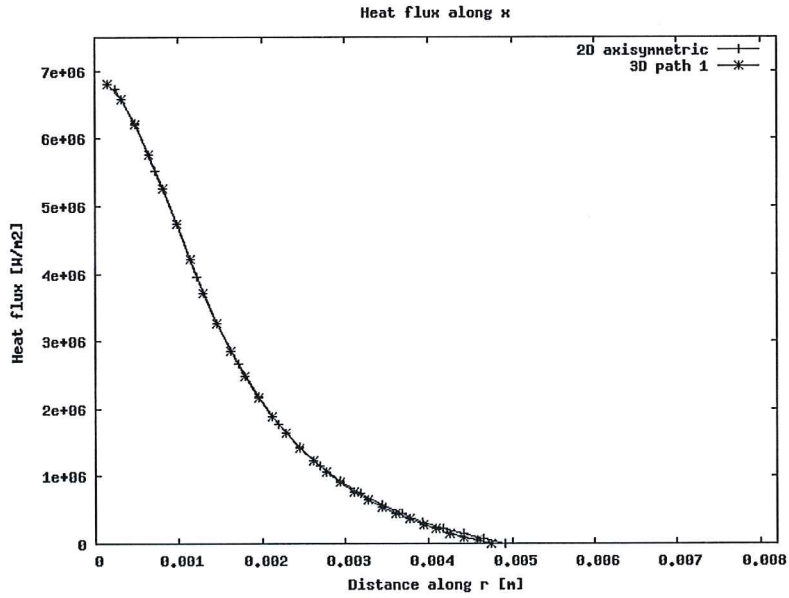


Figure 48: Heat flux calculated for the 0° tilt case, and for the 12° tilt case along the path 1 of Figure 42.

To conclude, the simulation results show that an electrode tilt angle of 12° rather than 0° has an influence on the temperature field, the velocity field and the heat distribution just above the base metal. The symmetry of all the fields calculated for a 0° tilt is not maintained when these fields are calculated accounting for a 12° electrode tilt. As a result the heat flux distribution above the base metal cannot be a Gaussian when the electrode is tilted by 12°.

6 Conclusions and future work

This study was divided into different steps: the bibliographic study, the development of a methodology, the study of a simplified electromagnetic problem and finally the GTAW problem.

The study of the simpler electromagnetic problem and its application to an infinite conducting rod did allow understanding how the 3D mesh for GTAW with a 12° electrode tilt could be done in the region with largest electric conduction (i.e. in the volume below the electrode tip up to the base metal). The simulation results of the infinite rod test case 2 showed a reasonable convergence in mesh and an acceptable agreement to the analytical solution. The mesh for the second test case was then used as basis for the preparing the GTAW mesh.

The GTAW test case was run for a 0° and for 12° electrode tilt. The 0° test case was converged in mesh; this was known from a former study [9]. However, because of memory limitations faced when running on the cluster, the 12° test case could not be run with as many mesh cells as initially desired. So work still remains to be done to check the convergence in mesh of this test case.

Based on the simulation results obtained for the GTAW test cases, it can be concluded that an electrode tilt angle of 12° rather than 0° has an influence on the temperature field and the velocity field within the arc, and an influence on the heat distribution just above the base metal. The symmetry of all the fields calculated for a 0° tilt is not maintained when accounting for a 12° electrode tilt. As a result the heat flux distribution above the base metal cannot be a Gaussian when the electrode is tilted by 12°. These results are however qualitative rather than quantitative because of the simplifications done when specifying the boundary conditions. The problem should thus be further investigated to determine with more precision the error done when assuming a Gaussian heat distribution for a 12° electrode tilt as for a for a 0° electrode tilt.

This could be done improving the boundary conditions, as suggested in Appendix H.

A better way (more accurate) would be to combine the magneto hydrodynamic model used in this study with an anode layer model and a cathode layer model. This would indeed allow calculating the temperature and current density distribution on the anode and cathode surface, rather than setting them based on extrapolated experimental data. Finally, to validate the GTAW simulation results the energy transferred to the base metal could be measured experimentally and compared to simulation results. However, this comparison of the total energy transferred (rather than a local transfer) may not be sufficient to conclude. Temperature measured on the base metal using thermocouples could allow obtaining interesting local data useful for validating the simulations.

References

1. Weman, K. (2003). *Welding processes handbook.* Cambridge: Woodhead Publishing Ltd
2. Joy Varghese, V. M., Suresh, M. R. & Siva Kumar, D. (2013). Recent developments in modeling of heat transfer during TIG welding – review. The International Journal of Advanced Manufacturing Technology, vol. 64, pp. 749-754
3. Wenchao, D., Shanping, L., Dianzhong, L. & Yiyi, L. (2011). GTAW liquid pool convections and the weld shape variations under helium gas shielding. International Journal of Heat and Mas Transfer, vol. 54, 1420-1431.
4. Murphy, A. B., Tanaka, M., Yamamoto, K., Tashiro, S., Sato, T., & Lowke, J. J. (2009). Modelling of thermal plasmas for arc welding: the role of the shieling gas properties and of metal vapour. Journal of physics D: Applied Physics, Vol. 42, 194006.
5. Murphy, A. B. (2010). The effects of metal vapour in arc welding. Journal of physics D: Applied Physics, Vol. 43, 434001.
6. Xu, G., Hu, J. & Tsai, H. L. (2009). Three-dimensional modeling of arc plasma and metal transfer in gas metal arc welding. International Journal of Heat and Mas Transfer, vol. 52, 1709-1724.
7. Rat, V., Murphy, A. B., Aubreton, J., Elchinger, M. F., & Fauchais, P. (2008). Treatment of non-equilibrium phenomena in thermal plasma flows. Journal of physics D: Applied Physics, Vol. 41, 183001.
8. Baeva, M., Kozakov, R., Gorchakov, S., & Uhrlandt, D. (2012). Two-temperature chemically non-equilibrium modeling of transferred arcs. Plasma Sources Science and Technology, Vol. 21, 055027.
9. Choquet, I., Javidi Shirvan, A. & Nilsson, H. (2012). On the choice of electromagnetic model for short high-intensity arcs, applied to welding. Journal of physics D: Applied Physics, Vol. 45, 205203.
10. Benilov, M S (2008). Understanding and modeling plasma-electrode interaction in high-pressure arc discharges: a review. Journal of physics D: Applied Physics, Vol. 41, 144001.
11. Sass-Tisovskaya, M. (2009). *Plasma Arc Welding Simulation with OpenFOAM.* Göteborg: Chalmers University of Technology, 2009.
12. OpenFOAM. (2012). [Electric] *User Guide Version 2.1.1.* OpenFOAM Foundation. Available: http://www.openfoam.org/docs/user [2013-05-08]
13. Versteeg, H. K. & Malalasekera (2007). *An Introduction to computational fluid dynamics the finite volume method.* 2. ed. Harlow: Pearson Prentice Hall.
14. White, F. M. (2010). *Fluid Mechanics.* 7. ed. Maidenhead: McGraw-Hill Higher Education
15. OpenFOAM. (2012). *Programmer's Guide Version 2.1.0.* OpenFOAM Foundation Available: http://www.openfoam.org, when downloading OpenFoam [2013-06-03]

A. Calculation of the increase in size between adjacent cells

```
function IPCCalc(L,NOC,ER)
% Calculation of the increase per cell for meshes with expansion ratio
% and number of cells as input.
% Made by Johanna Matsfelt

%Input:
%    L = Length of that side of that block
%    NOC = Number of cells along chosen side of that block
%    ER = Expansion ratio

%Output displayed:
%    IPC = Increase per cell, allowed to be max 1,1
%    ERc = Calculated length, will be equal to L if the formula works

% 2013-05-18

IPC=ER^(1/(NOC-1));

Nxs=0;  %Only to start the for loop

for j=1:NOC  % Prepare for calculation of xs
             %    j=Number of the current cell
    Nxsj=IPC^(j-1);
    Nxs=Nxs+Nxsj;
end

xs=L/Nxs; % The length of the cell in the start

xe=xs*IPC^(NOC-1);  % The length of the cell in the end

ERc=xe/xs; % Calculate expansion ratio to make sure the equation works

if IPC<=1.1
    display(['The increase per cell = ' num2str(IPC) ' which is fine,
Calculated expansion ratio = ' num2str(ERc)]);
else
    display(['The increase per cell = ' num2str(IPC) ' which is to
large, Calculated expansion ratio = ' num2str(ERc)]);
end
```

B. Calculation of the increase per cell belonging to adjacent blocks

```
function IPCCalcLimit(L1,NOC1,ER1,L2,NOC2,ER2)
% Calculation of the increase per cell for meshes with expansion ratio
% and number of cells as input in the limit of two blocks.
% OBS! Important that the last cell in the first block and the first
% cell in the second block are neighbors!
% Made by Johanna Matsfelt

%Input:
%   L1 = Length of that side of the first block
%   NOC1 = Number of cells along chosen side of the first block
%   ER1 = Expansion ratio for the first block
%   L2 = Length of that side of the second block
%   NOC2 = Number of cells along chosen side of the second block
%   ER2 = Expansion ratio for the second block

%Output displayed:
%   IPC = Increase per cell, allowed to be max 1,1
%   ERc1 = Calculated length, will be equal to L1 if the formula works
%   ERc2 = Calculated length, will be equal to L2 if the formula works

% 2013-05-21

% Calculations for the first block
IPC1=ER1^(1/(NOC1-1));

Nxs1=0;  %Only to start the for loop

for j=1:NOC1 % Prepare for calculation of xs
             %    j=Number of the current cell
    Nxsj1=IPC1^(j-1);
    Nxs1=Nxs1+Nxsj1;
end

xs1=L1/Nxs1; % The length of the cell in the start

xe1=xs1*IPC1^(NOC1-1);  % The length of the cell in the end

ERc1=xe1/xs1; %Calculate expansion ratio to make sure valid equation


% Calculations for the second block
IPC2=ER2^(1/(NOC2-1));

Nxs2=0;  %Only to start the for loop

for j=1:NOC2 % Prepare for calculation of xs, related to equation (20)
             %    j=Number of the current cell
    Nxsj2=IPC2^(j-1);
    Nxs2=Nxs2+Nxsj2;
end
```

```
xs2=L2/Nxs2; % The length of the cell in the start

xe2=xs2*IPC2^(NOC2-1);  % The length of the cell in the end

ERc2=xe2/xs2;  % Calculate expansion ratio to make sure valid equation

% Calculation of the increase per cell between the blocks
IPC=xs2/xe1;


if IPC<=1.1
    display(['The increase per cell = ' num2str(IPC) ' which is fine,
Calculated expansion ratio first block= ' num2str(ERc1) ', Calculated
expansion ratio second block= ' num2str(ERc2)]);
else
    display(['The increase per cell = ' num2str(IPC) ' which is to
large , Calculated expansion ratio first block= ' num2str(ERc1) ',
Calculated expansion ratio second block= ' num2str(ERc2)]);
end
```

C. Code for the whole 3D mesh of the infinite rod

```
// Infinite rod whole 3D Mesh

//         Created by Johanna Matsfelt

//Run using:
//m4 -P blockMeshDict.m4 > blockMeshDict

//m4 definitions:
m4_changecom(//)m4_changequote([,])
m4_define(calc, [m4_esyscmd(perl -e 'use Math::Trig; printf
($1)')])
m4_define(VCOUNT, 0)
m4_define(vlabel, [[// ]Vertex $1 = VCOUNT m4_define($1,
VCOUNT)m4_define([VCOUNT], m4_incr(VCOUNT))])

//Mathematical constants:
m4_define(pi, 3.1415926536)

//Geometry
m4_define(wedgeAngle, 90.0)
m4_define(rExterior,160)
m4_define(rRod,10)
m4_define(rSquare,5)

//Grid points (integers!):
m4_define(rRodNumberOfCells, 100)
m4_define(rNumberOfCells, 100)
m4_define(xABnumberOfCells, 50)
m4_define(rGrading, 1)

//Plane A:
m4_define(xA, 0)

//Plane B:
m4_define(xB, 100)


/*---------------------------------------------------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  1.4.1
|
|   \\  /    A nd           | Web:      http://www.openfoam.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------*/
```

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1e-4;

vertices
(
    //Plane A: (S=Square, R=Rod, E=Exterior, the
location(number) of the point is the number multiplied with the
wedge angle)
    (xA 0 0) vlabel(A0)

    (xA rSquare 0) vlabel(AS0)
    (xA calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(AS1)
    (xA calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(AS2)
    (xA calc(rSquare*cos(3*wedgeAngle*pi/180.0))
calc(rSquare*sin(3*wedgeAngle*pi/180.0))) vlabel(AS3)

    (xA rRod 0) vlabel(AR0)
    (xA calc(rRod*cos(wedgeAngle*pi/180.0))
calc(rRod*sin(wedgeAngle*pi/180.0))) vlabel(AR1)
    (xA calc(rRod*cos(2*wedgeAngle*pi/180.0))
calc(rRod*sin(2*wedgeAngle*pi/180.0))) vlabel(AR2)
    (xA calc(rRod*cos(3*wedgeAngle*pi/180.0))
calc(rRod*sin(3*wedgeAngle*pi/180.0))) vlabel(AR3)

    (xA rExterior 0) vlabel(AE0)
    (xA calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(AE1)
    (xA calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(AE2)
    (xA calc(rExterior*cos(3*wedgeAngle*pi/180.0))
calc(rExterior*sin(3*wedgeAngle*pi/180.0))) vlabel(AE3)


    //Plane B:
    (xB 0 0) vlabel(B0)

    (xB rSquare 0) vlabel(BS0)
    (xB calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(BS1)
    (xB calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(BS2)
```

```
    (xB calc(rSquare*cos(3*wedgeAngle*pi/180.0))
calc(rSquare*sin(3*wedgeAngle*pi/180.0))) vlabel(BS3)

    (xB rRod 0) vlabel(BR0)
    (xB calc(rRod*cos(wedgeAngle*pi/180.0))
calc(rRod*sin(wedgeAngle*pi/180.0))) vlabel(BR1)
    (xB calc(rRod*cos(2*wedgeAngle*pi/180.0))
calc(rRod*sin(2*wedgeAngle*pi/180.0))) vlabel(BR2)
    (xB calc(rRod*cos(3*wedgeAngle*pi/180.0))
calc(rRod*sin(3*wedgeAngle*pi/180.0))) vlabel(BR3)

    (xB rExterior 0) vlabel(BE0)
    (xB calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(BE1)
    (xB calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(BE2)
    (xB calc(rExterior*cos(3*wedgeAngle*pi/180.0))
calc(rExterior*sin(3*wedgeAngle*pi/180.0))) vlabel(BE3)

);

// Defining blocks:
blocks
(

    // block0
    hex (AS2 BS2 BS3 AS3 AS1 BS1 BS0 AS0)
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block1
    hex (AS1 BS1 BS0 AS0 AR1 BR1 BR0 AR0)
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block2
    hex (AS2 BS2 BS1 AS1 AR2 BR2 BR1 AR1)
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block3
    hex (AR3 BR3 BS3 AS3 AR2 BR2 BS2 AS2)
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block4
    hex (AR3 BR3 BR0 AR0 AS3 BS3 BS0 AS0)
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block5
    hex (AR1 BR1 BR0 AR0 AE1 BE1 BE0 AE0)
    (xABnumberOfCells rNumberOfCells rNumberOfCells)
```

```
    simpleGrading (1 rGrading 1)

    // block6
    hex (AR2 BR2 BR1 AR1 AE2 BE2 BE1 AE1)
    (xABnumberOfCells rNumberOfCells rNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block7
    hex (AE3 BE3 BR3 AR3 AE2 BE2 BR2 AR2)
    (xABnumberOfCells rNumberOfCells rNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block8
    hex (AE3 BE3 BE0 AE0 AR3 BR3 BR0 AR0)
    (xABnumberOfCells rNumberOfCells rNumberOfCells)
    simpleGrading (1 rGrading 1)

);

edges
(
    //Plane A:
    line AS0 AS1
    line AS1 AS2
    line AS2 AS3
    line AS3 AS0

    line AS0 AR0
    line AS1 AR1
    line AS2 AR2
    line AS3 AR3

    arc AR0 AR1 (xA calc(rRod*cos((1/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc AR1 AR2 (xA calc(rRod*cos((3/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc AR2 AR3 (xA calc(rRod*cos((5/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((5/2)*wedgeAngle*pi/180.0)))     //(5/2)*90
degrees
    arc AR3 AR0 (xA calc(rRod*cos((7/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees

    line AR0 AE0
    line AR1 AE1
    line AR2 AE2
    line AR3 AE3

    arc AE0 AE1 (xA
calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
```

```
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc AE1 AE2 (xA
calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc AE2 AE3 (xA
calc(rExterior*cos((5/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/2)*wedgeAngle*pi/180.0)))     //(5/2)*90
degrees
    arc AE3 AE0 (xA
calc(rExterior*cos((7/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees


    //Plane B:
    line BS0 BS1
    line BS1 BS2
    line BS2 BS3
    line BS3 BS0

    line BS0 BR0
    line BS1 BR1
    line BS2 BR2
    line BS3 BR3

    arc BR0 BR1 (xB calc(rRod*cos((1/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((1/2)*wedgeAngle*pi/180.0)))      //(1/2)*90
degrees
    arc BR1 BR2 (xB calc(rRod*cos((3/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((3/2)*wedgeAngle*pi/180.0)))      //(3/2)*90
degrees
    arc BR2 BR3 (xB calc(rRod*cos((5/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((5/2)*wedgeAngle*pi/180.0)))      //(5/2)*90
degrees
    arc BR3 BR0 (xB calc(rRod*cos((7/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((7/2)*wedgeAngle*pi/180.0)))      //(7/2)*90
degrees

    line BR0 BE0
    line BR1 BE1
    line BR2 BE2
    line BR3 BE3

    arc BE0 BE1 (xB
calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc BE1 BE2 (xB
calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
```

```
    arc BE2 BE3 (xB
calc(rExterior*cos((5/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc BE3 BE0 (xB
calc(rExterior*cos((7/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/2)*wedgeAngle*pi/180.0)))    //(7/2)*90
degrees


    //From plane A to B:
    line AS0 BS0
    line AS1 BS1
    line AS2 BS2
    line AS3 BS3

    line AR0 BR0
    line AR1 BR1
    line AR2 BR2
    line AR3 BR3

    line AE0 BE0
    line AE1 BE1
    line AE2 BE2
    line AE3 BE3

);

// Defining patches:
patches
(

    patch rodBottom
    (
       (AS0 AS1 AS2 AS3)
       (AS0 AS1 AR1 AR0)
       (AS1 AS2 AR2 AR1)
       (AS2 AS3 AR3 AR2)
       (AS3 AS0 AR0 AR3)
    )

    patch BoundaryBottom
    (
       (AR1 AR0 AE0 AE1)
       (AR2 AR1 AE1 AE2)
       (AR3 AR2 AE2 AE3)
       (AR0 AR3 AE3 AE0)
    )

    patch rodTop
    (
       (BS0 BS3 BS2 BS1)
       (BS0 BR0 BR1 BS1)
```

```
        (BS1 BR1 BR2 BS2)
        (BS2 BR2 BR3 BS3)
        (BS3 BR3 BR0 BS0)
    )

    patch BoundaryTop
    (
        (BR0 BR1 BE1 BE0)
        (BR1 BR2 BE2 BE1)
        (BR2 BR3 BE3 BE2)
        (BR3 BR0 BE0 BE3)
    )

   patch Exterior
    (
        (AE1 AE0 BE0 BE1)
        (AE2 AE1 BE1 BE2)
        (AE3 AE2 BE2 BE3)
        (AE0 AE3 BE3 BE0)
    )

);

mergePatchPairs
(
);

//************************************************************* //
```

D. Code for the half 3D mesh of the infinite rod

```
// Infinite Rod half 3D mesh

//          Created by Johanna Matsfelt

//Run using:
//m4 -P blockMeshDict.m4 > blockMeshDict

//m4 definitions:
m4_changecom(//)m4_changequote([,])
m4_define(calc, [m4_esyscmd(perl -e 'use Math::Trig; printf
($1)')])
m4_define(VCOUNT, 0)
m4_define(vlabel, [[// ]Vertex $1 = VCOUNT m4_define($1,
VCOUNT)m4_define([VCOUNT], m4_incr(VCOUNT))])

//Mathematical constants:
m4_define(pi, 3.1415926536)

//Geometry
m4_define(wedgeAngle, 90.0)
m4_define(rExterior,160)
m4_define(rRod,10)
m4_define(rSquare,4.5)

//Grid points (integers!):
// TEST 1
m4_define(rRodNumberOfCells, 2)
m4_define(rRodSNumberOfCells, 2)
m4_define(rNumberOfCells, 2)
m4_define(rNumberOfCellsExterior, 2)
m4_define(xABnumberOfCells, 100)
m4_define(rGrading, 1)
m4_define(rGradingExterior, 5)

// TEST 2
//m4_define(rRodNumberOfCells, 20)
//m4_define(rRodSNumberOfCells, 20)
//m4_define(rNumberOfCells, 20)
//m4_define(rNumberOfCellsExterior, 200)
//m4_define(xABnumberOfCells, 100)
//m4_define(rGrading, 1)
//m4_define(rGradingExterior, 5)

// TEST 3
//m4_define(rRodNumberOfCells, 40)
//m4_define(rRodSNumberOfCells, 40)
//m4_define(rNumberOfCells, 40)
//m4_define(rNumberOfCellsExterior, 400)
//m4_define(xABnumberOfCells, 50)
//m4_define(rGrading, 1)
```

```
//m4_define(rGradingExterior, 5)

// TEST 4
//m4_define(rRodNumberOfCells, 50)
//m4_define(rRodSNumberOfCells, 50)
//m4_define(rNumberOfCells, 50)
//m4_define(rNumberOfCellsExterior, 500)
//m4_define(xABnumberOfCells, 50)
//m4_define(rGrading, 1)
//m4_define(rGradingExterior, 5)

//Plane A:
m4_define(xA, 0)

//Plane B:
m4_define(xB, 100)


/*--------------------------------------------------------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  1.4.1
|
|   \\  /    A nd           | Web:      http://www.openfoam.org
|
|    \\/     M anipulation  |
|
\*-------------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1e-4;

vertices
(
    //Plane A: (S=Square, R=Rod, E=Exterior, the
location(number) of the point is the number multiplied with the
wedgeangle)
    (xA 0 0) vlabel(A0)

    (xA rSquare 0) vlabel(AS0)
```

```
    (xA rSquare calc(rSquare*sin(wedgeAngle*pi/180.0)))
vlabel(AS01)
    (xA calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(AS1)
    (xA calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(AS12)
    (xA calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(AS2)

    (xA rRod 0) vlabel(AR0)
    (xA calc(rRod*cos((1/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(AR01)
    (xA calc(rRod*cos(wedgeAngle*pi/180.0))
calc(rRod*sin(wedgeAngle*pi/180.0))) vlabel(AR1)
    (xA calc(rRod*cos((3/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(AR12)
    (xA calc(rRod*cos(2*wedgeAngle*pi/180.0))
calc(rRod*sin(2*wedgeAngle*pi/180.0))) vlabel(AR2)

    (xA rExterior 0) vlabel(AE0)
    (xA calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(AE01)
    (xA calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(AE1)
    (xA calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(AE12)
    (xA calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(AE2)


    //Plane B:
    (xB 0 0) vlabel(B0)

    (xB rSquare 0) vlabel(BS0)
    (xB rSquare calc(rSquare*sin(wedgeAngle*pi/180.0)))
vlabel(BS01)
    (xB calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(BS1)
    (xB calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(BS12)
    (xB calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(BS2)

    (xB rRod 0) vlabel(BR0)
    (xB calc(rRod*cos((1/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(BR01)
    (xB calc(rRod*cos(wedgeAngle*pi/180.0))
calc(rRod*sin(wedgeAngle*pi/180.0))) vlabel(BR1)
    (xB calc(rRod*cos((3/2)*wedgeAngle*pi/180.0))
calc(rRod*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(BR12)
    (xB calc(rRod*cos(2*wedgeAngle*pi/180.0))
calc(rRod*sin(2*wedgeAngle*pi/180.0))) vlabel(BR2)
```

```
    (xB rExterior 0) vlabel(BE0)
    (xB calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(BE01)
    (xB calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(BE1)
    (xB calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(BE12)
    (xB calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(BE2)


);

// Defining blocks:
blocks
(

    // block0
    hex (A0 B0 BS0 AS0 AS1 BS1 BS01 AS01) rod0
    (xABnumberOfCells rRodSNumberOfCells rRodSNumberOfCells)
    simpleGrading (1 1 1)

    // block1
    hex (AS2 BS2 B0 A0 AS12 BS12 BS1 AS1) rod1
    (xABnumberOfCells rRodSNumberOfCells rRodSNumberOfCells)
    simpleGrading (1 1 1)

    // block2
    hex (AS01 BS01 BS0 AS0 AR01 BR01 BR0 AR0) rod2
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading rGrading)

    // block3
    hex (AS1 BS1 BS01 AS01 AR1 BR1 BR01 AR01) rod3
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading rGrading)

    // block4
    hex (AS12 BS12 BS1 AS1 AR12 BR12 BR1 AR1) rod4
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading rGrading)

    // block5
    hex (AS2 BS2 BS12 AS12 AR2 BR2 BR12 AR12) rod5
    (xABnumberOfCells rRodNumberOfCells rRodNumberOfCells)
    simpleGrading (1 rGrading rGrading)

    // block6
    hex (AR01 BR01 BR0 AR0 AE01 BE01 BE0 AE0)
    (xABnumberOfCells rNumberOfCells rNumberOfCellsExterior)
    simpleGrading (1 rGrading rGradingExterior)

    // block7
```

```
    hex (AR1 BR1 BR01 AR01 AE1 BE1 BE01 AE01)
    (xABnumberOfCells rNumberOfCells rNumberOfCellsExterior)
    simpleGrading (1 rGrading rGradingExterior)

    // block8
    hex (AR12 BR12 BR1 AR1 AE12 BE12 BE1 AE1)
    (xABnumberOfCells rNumberOfCells rNumberOfCellsExterior)
    simpleGrading (1 rGrading rGradingExterior)

    // block9
    hex (AR2 BR2 BR12 AR12 AE2 BE2 BE12 AE12)
    (xABnumberOfCells rNumberOfCells rNumberOfCellsExterior)
    simpleGrading (1 rGrading rGradingExterior)

);

edges
(
    //Plane A:
    line A0 AS0
    line AS0 AS01
    line AS01 AS1
    line AS1 A0
    line AS1 AS12
    line AS12 AS2

    line AS0 AR0
    line AS01 AR01
    line AS1 AR1
    line AS12 AR12
    line AS2 AR2

    arc AR0 AR01 (xA calc(rRod*cos((1/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc AR01 AR1 (xA calc(rRod*cos((3/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc AR1 AR12 (xA calc(rRod*cos((5/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc AR12 AR2 (xA calc(rRod*cos((7/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees

    line AR0 AE0
    line AR01 AE01
    line AR1 AE1
    line AR12 AE12
    line AR2 AE2

    arc AE0 AE01 (xA
calc(rExterior*cos((1/4)*wedgeAngle*pi/180.0))
```

```
calc(rExterior*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc AE01 AE1 (xA
calc(rExterior*cos((3/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc AE1 AE12 (xA
calc(rExterior*cos((5/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc AE12 AE2 (xA
calc(rExterior*cos((7/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees


    //Plane B:
    line B0 BS0
    line BS0 BS01
    line BS01 BS1
    line BS1 B0
    line BS1 BS12
    line BS12 BS2

    line BS0 BR0
    line BS01 BR01
    line BS1 BR1
    line BS12 BR12
    line BS2 BR2

    arc BR0 BR01 (xB calc(rRod*cos((1/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((1/4)*wedgeAngle*pi/180.0)))      //(1/4)*90
degrees
    arc BR01 BR1 (xB calc(rRod*cos((3/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((3/4)*wedgeAngle*pi/180.0)))      //(3/4)*90
degrees
    arc BR1 BR12 (xB calc(rRod*cos((5/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((5/4)*wedgeAngle*pi/180.0)))      //(5/4)*90
degrees
    arc BR12 BR2 (xB calc(rRod*cos((7/4)*wedgeAngle*pi/180.0))
calc(rRod*sin((7/4)*wedgeAngle*pi/180.0)))      //(7/4)*90
degrees

    line BR0 BE0
    line BR01 BE01
    line BR1 BE1
    line BR12 BE12
    line BR2 BE2

    arc BE0 BE01 (xB
calc(rExterior*cos((1/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
```

```
    arc BE01 BE1 (xB
calc(rExterior*cos((3/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/4)*wedgeAngle*pi/180.0)))    //(3/4)*90
degrees
    arc BE1 BE12 (xB
calc(rExterior*cos((5/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/4)*wedgeAngle*pi/180.0)))    //(5/4)*90
degrees
    arc BE12 BE2 (xB
calc(rExterior*cos((7/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/4)*wedgeAngle*pi/180.0)))    //(7/4)*90
degrees


    //From plane A to B:
    line A0 B0

    line AS0 BS0
    line AS01 BS01
    line AS1 BS1
    line AS12 BS12
    line AS2 BS2

    line AR0 BR0
    line AR01 BR01
    line AR1 BR1
    line AR12 BR12
    line AR2 BR2

    line AE0 BE0
    line AE01 BE01
    line AE1 BE1
    line AE12 BE12
    line AE2 BE2


);

// Defining patches:
patches
(

    patch leftWall
    (
        (AS0 A0 AS1 AS01)
        (A0 AS2 AS12 AS1)
        (AS0 AS01 AR01 AR0)
        (AS01 AS1 AR1 AR01)
        (AS1 AS12 AR12 AR1)
        (AS12 AS2 AR2 AR12)
    )

    patch atmosphereLeft
```

```
    (
        (AR0 AR01 AE01 AE0)
        (AR01 AR1 AE1 AE01)
        (AR1 AR12 AE12 AE1)
        (AR12 AR2 AE2 AE12)
    )

    patch rightWall
    (
        (BS0 BS01 BS1 B0)
        (B0 BS1 BS12 BS2)
        (BS0 BR0 BR01 BS01)
        (BS01 BR01 BR1 BS1)
        (BS1 BR1 BR12 BS12)
        (BS12 BR12 BR2 BS2)
    )

    patch atmosphereRight
    (
        (BR0 BE0 BE01 BR01)
        (BR01 BE01 BE1 BR1)
        (BR1 BE1 BE12 BR12)
        (BR12 BE12 BE2 BR2)
    )

   patch atmosphereTop
    (
        (AE0 AE01 BE01 BE0)
        (AE01 AE1 BE1 BE01)
        (AE1 AE12 BE12 BE1)
        (AE12 AE2 BE2 BE12)
    )
   symmetryPlane halfModelPlane
    (
        (AE0 BE0 BR0 AR0)
        (AR0 BR0 BS0 AS0)
        (AS0 BS0 B0 A0)
        (A0 B0 BS2 AS2)
        (AS2 BS2 BR2 AR2)
        (AR2 BR2 BE2 AE2)
    )

);

mergePatchPairs
(
);

//************************************************************** //
```

E. Code for the whole 3D mesh of GTAW with zero degree tilt angle

```
// GTAW zero degree tilt angle whole 3D Mesh

//          Created by Johanna Matsfelt

//Run using:
//m4 -P blockMeshDict.m4 > blockMeshDict

//m4 definitions:
m4_changecom(//)m4_changequote([,])
m4_define(calc, [m4_esyscmd(perl -e 'use Math::Trig; printf
($1)')])
m4_define(VCOUNT, 0)
m4_define(vlabel, [[// ]Vertex $1 = VCOUNT m4_define($1,
VCOUNT)m4_define([VCOUNT], m4_incr(VCOUNT))])

//Mathematical constants:
m4_define(pi, 3.1415926536)

//----- Geometry

m4_define(wedgeAngle, 90.0)
m4_define(tiltAngle, 0.0)


m4_define(rSquare,0.1)
m4_define(rTip, 0.2)
m4_define(rCathode, 1.6)
m4_define(rNozzle, 5)
m4_define(rExterior, 8.2)

//-------------------Grid points (integers!):
// x-direction   dx=0.02
m4_define(xTipNumberOfCells, 100)
m4_define(xCathode2NumberOfCells, 121)
m4_define(xCathode1NumberOfCells, 55)
m4_define(xCathodeInletNumberOfCells, 150)


//y-direction
m4_define(rTipNumberOfCells, 10)
m4_define(rCathodeNumberOfCells, 270)
m4_define(rOutletNumberOfCells, 95)


//z-direction
m4_define(zNumberOfCells, 10)

m4_define(rGrading, 1)
```

```
//Plane A:
m4_define(xA, 0)

//Plane T:
m4_define(xT, 2)

//Plane C:
m4_define(xC, 4.42)

//Plane N:
m4_define(xN, 5.52)

//Plane I:
m4_define(xI, 10.52)
```

```
/*--------------------------------------------------------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  1.4.1
|
|   \\  /    A nd           | Web:      http://www.openfoam.org
|
|    \\/     M anipulation  |
|
\*--------------------------------------------------------------*/

FoamFile
{
    version         2.0;
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 1e-3;

vertices
(
    //Plane A: (S=Square, T=Tip, N=Nozzle, the location(number)
of the point is the number multiplied with the wedge angle)
    (xA 0 0) vlabel(A0)

    (xA rSquare 0) vlabel(AS0)
    (xA calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(AS1)
```

```
    (xA calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(AS2)
    (xA calc(rSquare*cos(3*wedgeAngle*pi/180.0))
calc(rSquare*sin(3*wedgeAngle*pi/180.0))) vlabel(AS3)

    (xA rTip 0) vlabel(AT0)
    (xA calc(rTip*cos(wedgeAngle*pi/180.0))
calc(rTip*sin(wedgeAngle*pi/180.0))) vlabel(AT1)
    (xA calc(rTip*cos(2*wedgeAngle*pi/180.0))
calc(rTip*sin(2*wedgeAngle*pi/180.0))) vlabel(AT2)
    (xA calc(rTip*cos(3*wedgeAngle*pi/180.0))
calc(rTip*sin(3*wedgeAngle*pi/180.0))) vlabel(AT3)

    (xA rNozzle 0) vlabel(AN0)
    (xA calc(rNozzle*cos(wedgeAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(AN1)
    (xA calc(rNozzle*cos(2*wedgeAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(AN2)
    (xA calc(rNozzle*cos(3*wedgeAngle*pi/180.0))
calc(rNozzle*sin(3*wedgeAngle*pi/180.0))) vlabel(AN3)

    (xA rExterior 0) vlabel(AE0)
    (xA calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(AE1)
    (xA calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(AE2)
    (xA calc(rExterior*cos(3*wedgeAngle*pi/180.0))
calc(rExterior*sin(3*wedgeAngle*pi/180.0))) vlabel(AE3)


    //Plane T:
    (xT 0 0) vlabel(T0)

    (xT rSquare 0) vlabel(TS0)
    (xT calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(TS1)
    (xT calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(TS2)
    (xT calc(rSquare*cos(3*wedgeAngle*pi/180.0))
calc(rSquare*sin(3*wedgeAngle*pi/180.0))) vlabel(TS3)

    (xT rTip 0) vlabel(TT0)
    (xT calc(rTip*cos(wedgeAngle*pi/180.0))
calc(rTip*sin(wedgeAngle*pi/180.0))) vlabel(TT1)
    (xT calc(rTip*cos(2*wedgeAngle*pi/180.0))
calc(rTip*sin(2*wedgeAngle*pi/180.0))) vlabel(TT2)
    (xT calc(rTip*cos(3*wedgeAngle*pi/180.0))
calc(rTip*sin(3*wedgeAngle*pi/180.0))) vlabel(TT3)

    (xT rNozzle 0) vlabel(TN0)
    (xT calc(rNozzle*cos(wedgeAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(TN1)
```

```
    (xT calc(rNozzle*cos(2*wedgeAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(TN2)
    (xT calc(rNozzle*cos(3*wedgeAngle*pi/180.0))
calc(rNozzle*sin(3*wedgeAngle*pi/180.0))) vlabel(TN3)

    (xT rExterior 0) vlabel(TE0)
    (xT calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(TE1)
    (xT calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(TE2)
    (xT calc(rExterior*cos(3*wedgeAngle*pi/180.0))
calc(rExterior*sin(3*wedgeAngle*pi/180.0))) vlabel(TE3)


    //Plane C:
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT) calc((xC-
xT)*sin(tiltAngle*pi/180.0)) 0) vlabel(C0)

    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*sin(tiltAngle*pi/180.0)) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(tiltAngle*pi/180.0))
0) vlabel(CC0)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rCathode*cos(wedgeAngle*pi/180.0))+(xC-
xT)*sin(tiltAngle*pi/180.0))
calc(rCathode*sin(wedgeAngle*pi/180.0))) vlabel(CC1)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(2*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(2*wedgeAngle*pi/180.0))) vlabel(CC2)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(3*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(3*wedgeAngle*pi/180.0))) vlabel(CC3)

    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*sin(tiltAngle*pi/180.0)) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(tiltAngle*pi/180.0)) 0)
vlabel(CN0)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rNozzle*cos(wedgeAngle*pi/180.0))+(xC-
xT)*sin(tiltAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(CN1)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(2*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(CN2)
```

```
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(3*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(3*wedgeAngle*pi/180.0))) vlabel(CN3)

    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*sin(tiltAngle*pi/180.0)) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos(tiltAngle*pi/180.0))
0) vlabel(CE0)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rExterior*cos(wedgeAngle*pi/180.0))+(xC-
xT)*sin(tiltAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(CE1)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos(2*wedgeAngle*pi/180.0
)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(CE2)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos(3*wedgeAngle*pi/180.0
)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin(3*wedgeAngle*pi/180.0))) vlabel(CE3)


    //Plane N:
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT) calc((xN-
xT)*sin(tiltAngle*pi/180.0)) 0) vlabel(N0)

    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*sin(tiltAngle*pi/180.0)) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(tiltAngle*pi/180.0))
0) vlabel(NC0)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rCathode*cos(wedgeAngle*pi/180.0))+(xN-
xT)*sin(tiltAngle*pi/180.0))
calc(rCathode*sin(wedgeAngle*pi/180.0))) vlabel(NC1)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(2*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(2*wedgeAngle*pi/180.0))) vlabel(NC2)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(3*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(3*wedgeAngle*pi/180.0))) vlabel(NC3)
```

```
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0)+xT)-
rNozzle*sin(tiltAngle*pi/180.0)) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(tiltAngle*pi/180.0)) 0)
vlabel(NN0)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rNozzle*cos(wedgeAngle*pi/180.0))+(xN-
xT)*sin(tiltAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(NN1)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(2*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(NN2)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(3*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(3*wedgeAngle*pi/180.0))) vlabel(NN3)

    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*sin(tiltAngle*pi/180.0)) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos(tiltAngle*pi/180.0))
0) vlabel(NE0)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rExterior*cos(wedgeAngle*pi/180.0))+(xN-
xT)*sin(tiltAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(NE1)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos(2*wedgeAngle*pi/180.0
)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(NE2)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos(3*wedgeAngle*pi/180.0
)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin(3*wedgeAngle*pi/180.0))) vlabel(NE3)


    //Plane I:
    (calc((xI-xT)*cos(tiltAngle*pi/180.0)+xT) calc((xI-
xT)*sin(tiltAngle*pi/180.0)) 0) vlabel(I0)

    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*sin(tiltAngle*pi/180.0)) calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(tiltAngle*pi/180.0))
0) vlabel(IC0)
```

```
    (calc((xI-xT)*cos(tiltAngle*pi/180.0)+xT)
calc((rCathode*cos(wedgeAngle*pi/180.0))+(xI-
xT)*sin(tiltAngle*pi/180.0))
calc(rCathode*sin(wedgeAngle*pi/180.0))) vlabel(IC1)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(2*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(2*wedgeAngle*pi/180.0))) vlabel(IC2)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(3*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(3*wedgeAngle*pi/180.0))) vlabel(IC3)

    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*sin(tiltAngle*pi/180.0)) calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(tiltAngle*pi/180.0)) 0)
vlabel(IN0)
    (calc((xI-xT)*cos(tiltAngle*pi/180.0)+xT)
calc((rNozzle*cos(wedgeAngle*pi/180.0))+(xI-
xT)*sin(tiltAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(IN1)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(2*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(IN2)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(3*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(3*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(3*wedgeAngle*pi/180.0))) vlabel(IN3)


);

// Defining blocks:
blocks
(

    // block0
    hex (AS2 TS2 TS3 AS3 AS1 TS1 TS0 AS0)
    (xTipNumberOfCells rTipNumberOfCells zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block1
    hex (AS1 TS1 TS0 AS0 AT1 TT1 TT0 AT0)
    (xTipNumberOfCells rTipNumberOfCells zNumberOfCells)
```

```
simpleGrading (1 rGrading 1)

// block2
hex (AS2 TS2 TS1 AS1 AT2 TT2 TT1 AT1)
(xTipNumberOfCells rTipNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block3
hex (AT3 TT3 TS3 AS3 AT2 TT2 TS2 AS2)
(xTipNumberOfCells rTipNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block4
hex (AT3 TT3 TT0 AT0 AS3 TS3 TS0 AS0)
(xTipNumberOfCells rTipNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block5
hex (AT1 TT1 TT0 AT0 AN1 TN1 TN0 AN0)
(xTipNumberOfCells rCathodeNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block6
hex (AT2 TT2 TT1 AT1 AN2 TN2 TN1 AN1)
(xTipNumberOfCells rCathodeNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block7
hex (AN3 TN3 TT3 AT3 AN2 TN2 TT2 AT2)
(xTipNumberOfCells rCathodeNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block8
hex (AN3 TN3 TN0 AN0 AT3 TT3 TT0 AT0)
(xTipNumberOfCells rCathodeNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block9
hex (AN1 TN1 TN0 AN0 AE1 TE1 TE0 AE0)
(xTipNumberOfCells rOutletNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block10
hex (AN2 TN2 TN1 AN1 AE2 TE2 TE1 AE1)
(xTipNumberOfCells rOutletNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block11
hex (AE3 TE3 TN3 AN3 AE2 TE2 TN2 AN2)
(xTipNumberOfCells rOutletNumberOfCells zNumberOfCells)
simpleGrading (1 rGrading 1)

// block12
```

```
    hex (AE3 TE3 TE0 AE0 AN3 TN3 TN0 AN0)
    (xTipNumberOfCells rOutletNumberOfCells zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block13
    hex (TT1 CC1 CC0 TT0 TN1 CN1 CN0 TN0)
    (xCathode2NumberOfCells rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block14
    hex (TT2 CC2 CC1 TT1 TN2 CN2 CN1 TN1)
    (xCathode2NumberOfCells rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block15
    hex (TN3 CN3 CC3 TT3 TN2 CN2 CC2 TT2)
    (xCathode2NumberOfCells rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block16
    hex (TN3 CN3 CN0 TN0 TT3 CC3 CC0 TT0)
    (xCathode2NumberOfCells rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block17
    hex (TN1 CN1 CN0 TN0 TE1 CE1 CE0 TE0)
    (xCathode2NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block18
    hex (TN2 CN2 CN1 TN1 TE2 CE2 CE1 TE1)
    (xCathode2NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block19
    hex (TE3 CE3 CN3 TN3 TE2 CE2 CN2 TN2)
    (xCathode2NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block20
    hex (TE3 CE3 CE0 TE0 TN3 CN3 CN0 TN0)
    (xCathode2NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block21
```

```
    hex (CC1 NC1 NC0 CC0 CN1 NN1 NN0 CN0)
    (xCathode1NumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block22
    hex (CC2 NC2 NC1 CC1 CN2 NN2 NN1 CN1)
    (xCathode1NumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block23
    hex (CN3 NN3 NC3 CC3 CN2 NN2 NC2 CC2)
    (xCathode1NumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block24
    hex (CN3 NN3 NN0 CN0 CC3 NC3 NC0 CC0)
    (xCathode1NumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block25
    hex (CN1 NN1 NN0 CN0 CE1 NE1 NE0 CE0)
    (xCathode1NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block26
    hex (CN2 NN2 NN1 CN1 CE2 NE2 NE1 CE1)
    (xCathode1NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block27
    hex (CE3 NE3 NN3 CN3 CE2 NE2 NN2 CN2)
    (xCathode1NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block28
    hex (CE3 NE3 NE0 CE0 CN3 NN3 NN0 CN0)
    (xCathode1NumberOfCells rOutletNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block29
    hex (NC1 IC1 IC0 NC0 NN1 IN1 IN0 NN0)
    (xCathodeInletNumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)
```

```
    // block30
    hex (NC2 IC2 IC1 NC1 NN2 IN2 IN1 NN1)
    (xCathodeInletNumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block31
    hex (NN3 IN3 IC3 NC3 NN2 IN2 IC2 NC2)
    (xCathodeInletNumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)

    // block32
    hex (NN3 IN3 IN0 NN0 NC3 IC3 IC0 NC0)
    (xCathodeInletNumberOfCells  rCathodeNumberOfCells
zNumberOfCells)
    simpleGrading (1 rGrading 1)


);

edges
(
    //Plane A:
    line AS0 AS1
    line AS1 AS2
    line AS2 AS3
    line AS3 AS0

    line AS0 AT0
    line AS1 AT1
    line AS2 AT2
    line AS3 AT3

    arc AT0 AT1 (xA calc(rTip*cos((1/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc AT1 AT2 (xA calc(rTip*cos((3/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc AT2 AT3 (xA calc(rTip*cos((5/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((5/2)*wedgeAngle*pi/180.0)))     //(5/2)*90
degrees
    arc AT3 AT0 (xA calc(rTip*cos((7/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees

    line AT0 AN0
    line AT1 AN1
    line AT2 AN2
    line AT3 AN3
```

```
    arc AN0 AN1 (xA calc(rNozzle*cos((1/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0)))    //(1/2)*90
degrees
    arc AN1 AN2 (xA calc(rNozzle*cos((3/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0)))    //(3/2)*90
degrees
    arc AN2 AN3 (xA calc(rNozzle*cos((5/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc AN3 AN0 (xA calc(rNozzle*cos((7/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((7/2)*wedgeAngle*pi/180.0)))    //(7/2)*90
degrees

    line AN0 AE0
    line AN1 AE1
    line AN2 AE2
    line AN3 AE3

    arc AE0 AE1 (xA
calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0)))    //(1/2)*90
degrees
    arc AE1 AE2 (xA
calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0)))    //(3/2)*90
degrees
    arc AE2 AE3 (xA
calc(rExterior*cos((5/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc AE3 AE0 (xA
calc(rExterior*cos((7/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/2)*wedgeAngle*pi/180.0)))    //(7/2)*90
degrees


    //Plane T:
    line TS0 TS1
    line TS1 TS2
    line TS2 TS3
    line TS3 TS0

    line TS0 TT0
    line TS1 TT1
    line TS2 TT2
    line TS3 TT3

    arc TT0 TT1 (xT calc(rTip*cos((1/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc TT1 TT2 (xT calc(rTip*cos((3/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
```

```
    arc TT2 TT3 (xT calc(rTip*cos((5/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((5/2)*wedgeAngle*pi/180.0)))      //(5/2)*90
degrees
    arc TT3 TT0 (xT calc(rTip*cos((7/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((7/2)*wedgeAngle*pi/180.0)))      //(7/2)*90
degrees

    line TT0 TN0
    line TT1 TN1
    line TT2 TN2
    line TT3 TN3

    arc TN0 TN1 (xT calc(rNozzle*cos((1/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0)))    //(1/2)*90
degrees
    arc TN1 TN2 (xT calc(rNozzle*cos((3/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0)))    //(3/2)*90
degrees
    arc TN2 TN3 (xT calc(rNozzle*cos((5/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc TN3 TN0 (xT calc(rNozzle*cos((7/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((7/2)*wedgeAngle*pi/180.0)))    //(7/2)*90
degrees

    line TN0 TE0
    line TN1 TE1
    line TN2 TE2
    line TN3 TE3

    arc TE0 TE1 (xT
calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0)))    //(1/2)*90
degrees
    arc TE1 TE2 (xT
calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0)))    //(3/2)*90
degrees
    arc TE2 TE3 (xT
calc(rExterior*cos((5/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc TE3 TE0 (xT
calc(rExterior*cos((7/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/2)*wedgeAngle*pi/180.0)))    //(7/2)*90
degrees


    //Plane C:
    arc CC0 CC1 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/2)*wedgeAngle*pi/18
```

```
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/2)*wedgeAngle*pi/180.0)))      //(1/2)*90
degrees
    arc CC1 CC2 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/2)*wedgeAngle*pi/180.0)))      //(3/2)*90
degrees
    arc CC2 CC3  (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((5/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((5/2)*wedgeAngle*pi/180.0)))      //(5/2)*90
degrees
    arc CC3 CC0 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((7/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((7/2)*wedgeAngle*pi/180.0)))      //(7/2)*90
degrees

    line CC0 CN0
    line CC1 CN1
    line CC2 CN2
    line CC3 CN3

    arc CN0 CN1 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0)))    //(1/2)*90
degrees
    arc CN1 CN2 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0)))    //(3/2)*90
degrees
    arc CN2 CN3 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((5/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc CN3 CN0 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
```

```
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((7/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees

    line CN0 CE0
    line CN1 CE1
    line CN2 CE2
    line CN3 CE3

    arc CE0 CE1 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((1/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc CE1 CE2 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((3/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc CE2 CE3 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((5/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((5/2)*wedgeAngle*pi/180.0)))     //(5/2)*90
degrees
    arc CE3 CE0 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((7/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees


    //Plane N:
    arc NC0 NC1 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc NC1 NC2 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/2)*wedgeAngle*pi/18
```

```
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/2)*wedgeAngle*pi/180.0)))      //(3/2)*90
degrees
    arc NC2 NC3 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((5/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((5/2)*wedgeAngle*pi/180.0)))      //(5/2)*90
degrees
    arc NC3 NC0 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((7/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((7/2)*wedgeAngle*pi/180.0)))      //(7/2)*90
degrees

    line NC0 NN0
    line NC1 NN1
    line NC2 NN2
    line NC3 NN3

    arc NN0 NN1 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc NN1 NN2 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc NN2 NN3 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((5/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((5/2)*wedgeAngle*pi/180.0)))     //(5/2)*90
degrees
    arc NN3 NN0 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((7/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees

    line NN0 NE0
```

```
    line NN1 NE1
    line NN2 NE2
    line NN3 NE3

    arc NE0 NE1 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((1/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0)))    //(1/2)*90
degrees
    arc NE1 NE2 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((3/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0)))    //(3/2)*90
degrees
    arc NE2 NE3 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((5/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((5/2)*wedgeAngle*pi/180.0)))    //(5/2)*90
degrees
    arc NE3 NE0 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rExterior*cos((7/2)*wedgeAngle*pi/1
80.0)*cos(tiltAngle*pi/180.0))
calc(rExterior*sin((7/2)*wedgeAngle*pi/180.0)))    //(7/2)*90
degrees



    //Plane I:
    arc IC0 IC1 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc IC1 IC2 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc IC2 IC3 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
```

```
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((5/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((5/2)*wedgeAngle*pi/180.0)))      //(5/2)*90
degrees
    arc IC3 IC0 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((7/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((7/2)*wedgeAngle*pi/180.0)))      //(7/2)*90
degrees

    line IC0 IN0
    line IC1 IN1
    line IC2 IN2
    line IC3 IN3

    arc IN0 IN1 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0)))     //(1/2)*90
degrees
    arc IN1 IN2 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0)))     //(3/2)*90
degrees
    arc IN2 IN3 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((5/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((5/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((5/2)*wedgeAngle*pi/180.0)))     //(5/2)*90
degrees
    arc IN3 IN0 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((7/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((7/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((7/2)*wedgeAngle*pi/180.0)))     //(7/2)*90
degrees


    //From plane A to T:
    line AS0 TS0
    line AS1 TS1
    line AS2 TS2
    line AS3 TS3
```

```
line AT0 TT0
line AT1 TT1
line AT2 TT2
line AT3 TT3

line AN0 TN0
line AN1 TN1
line AN2 TN2
line AN3 TN3

line AE0 TE0
line AE1 TE1
line AE2 TE2
line AE3 TE3


//From plane T to C:
line TT0 CC0
line TT1 CC1
line TT2 CC2
line TT3 CC3

line TN0 CN0
line TN1 CN1
line TN2 CN2
line TN3 CN3

line TE0 CE0
line TE1 CE1
line TE2 CE2
line TE3 CE3


//From plane C to N:
line CC0 NC0
line CC1 NC1
line CC2 NC2
line CC3 NC3

line CN0 NN0
line CN1 NN1
line CN2 NN2
line CN3 NN3

line CE0 NE0
line CE1 NE1
line CE2 NE2
line CE3 NE3


//From plane N to I:
line NC0 IC0
line NC1 IC1
```

```
    line NC2 IC2
    line NC3 IC3

    line NN0 IN0
    line NN1 IN1
    line NN2 IN2
    line NN3 IN3


);

// Defining patches:
patches
(

    patch anodeTipWall
    (
        (AS0 AS1 AS2 AS3)
        (AS0 AS1 AT1 AT0)
        (AS1 AS2 AT2 AT1)
        (AS2 AS3 AT3 AT2)
        (AS3 AS0 AT0 AT3)
    )

    patch anodeWall
    (
        (AT0 AT1 AN1 AN0)
        (AT1 AT2 AN2 AN1)
        (AT2 AT3 AN3 AN2)
        (AT3 AT0 AN0 AN3)
    )

    patch anodeOutletWall
    (
        (AN0 AN1 AE1 AE0)
        (AN1 AN2 AE2 AE1)
        (AN2 AN3 AE3 AE2)
        (AN3 AN0 AE0 AE3)
    )

   patch OutletAnode
    (
         (AE0 AE1 TE1 TE0)
         (AE1 AE2 TE2 TE1)
         (AE2 AE3 TE3 TE2)
         (AE3 AE0 TE0 TE3)
    )

   patch OutletNozzle
    (
         (TE0 TE1 CE1 CE0)
         (TE1 TE2 CE2 CE1)
         (TE2 TE3 CE3 CE2)
```

```
        (TE3 TE0 CE0 CE3)
        (CE0 CE1 NE1 NE0)
        (CE1 CE2 NE2 NE1)
        (CE2 CE3 NE3 NE2)
        (CE3 CE0 NE0 NE3)
    )

    patch NozzleWall
    (
        (NN0 NE0 NE1 NN1)
        (NN1 NE1 NE2 NN2)
        (NN2 NE2 NE3 NN3)
        (NN3 NE3 NE0 NN0)
        (NN1 IN1 IN0 NN0)
        (NN2 IN2 IN1 NN1)
        (NN3 IN3 IN2 NN2)
        (NN0 IN0 IN3 NN3)
    )

    patch InletSides
    (
        (IC1 IC0 IN0 IN1)
        (IC2 IC1 IN1 IN2)
        (IC3 IC2 IN2 IN3)
        (IC0 IC3 IN3 IN0)
    )

    patch CathodeWall1
    (
        (IC0 IC1 NC1 NC0)
        (IC1 IC2 NC2 NC1)
        (IC2 IC3 NC3 NC2)
        (IC3 IC0 NC0 NC3)
        (NC0 NC1 CC1 CC0)
        (NC1 NC2 CC2 CC1)
        (NC2 NC3 CC3 CC2)
        (NC3 NC0 CC0 CC3)
    )

    patch CathodeWall2
    (
        (TT0 CC0 CC1 TT1)
        (TT1 CC1 CC2 TT2)
        (TT2 CC2 CC3 TT3)
        (TT3 CC3 CC0 TT0)
    )

    patch TipWall
    (
        (TS0 TS3 TS2 TS1)
        (TS0 TT0 TT1 TS1)
        (TS1 TT1 TT2 TS2)
        (TS2 TT2 TT3 TS3)
```

```
        (TS3 TT3 TT0 TS0)
    )

);

mergePatchPairs
(
);

// ************************************************************************
//
```

F. Code to the half 3D mesh of GTAW twelve degree tilt angle

```
// GTAW twelve degree tilt angle half 3D mesh

//          Created by Johanna Matsfelt

//Run using:
//m4 -P blockMeshDict.m4 > blockMeshDict

//m4 definitions:
m4_changecom(//)m4_changequote([,])
m4_define(calc, [m4_esyscmd(perl -e 'use Math::Trig; printf
($1)')])
m4_define(VCOUNT, 0)
m4_define(vlabel, [[// ]Vertex $1 = VCOUNT m4_define($1,
VCOUNT)m4_define([VCOUNT], m4_incr(VCOUNT))])

//Mathematical constants:
m4_define(pi, 3.1415926536)


//----- Geometry

m4_define(wedgeAngle, 90.0)
m4_define(tiltAngle, 12.0)


m4_define(rSquare, 0.1)
m4_define(rTip, 0.2)
m4_define(rCathode, 1.6)
m4_define(rNozzle, 5)
m4_define(rExterior, 8.2)

//-------------------Grid points (integers!):
// x direction dx=0.02
m4_define(xTipNumberOfCells, 100)
m4_define(xCathode2NumberOfCells, 121)
m4_define(xCathode1NumberOfCells, 55)
m4_define(xCathodeInletNumberOfCells, 150)


//y and z direction, both to generate square mesh in the middle
m4_define(rTipNumberOfCells, 10)


// z direction, is along the radii of the domain
m4_define(rCathodeNumberOfCells, 270)
m4_define(rOutletNumberOfCells, 95)
```

```
//---------------- Expansion ratios
//x-direction
m4_define(xGradingCathode2, 1)
m4_define(xGradingCathode1, 1)
m4_define(xGradingCathodeInlet, 2)

//z-direction
m4_define(rGradingCathode, 2.3)
m4_define(rGradingOutlet, 2.25)



//Plane A:
m4_define(xA, 0)

//Plane T:
m4_define(xT, 2)

//Plane C:
m4_define(xC, 4.42)

//Plane N:
m4_define(xN, 5.52)

//Plane I:
m4_define(xI, 10.52)



// Sum of planes, for the ratio calculation
m4_define(xsum, calc(xC+xN))   //At 0 tilt angle



/*---------------------------------------------------------------
-------------*\
| =========                 |
|
| \\      /  F ield         | OpenFOAM: The Open Source CFD
Toolbox           |
|  \\    /   O peration     | Version:  1.4.1
|
|   \\  /    A nd           | Web:      http://www.openfoam.org
|
|    \\/     M anipulation  |
|
\*---------------------------------------------------------------
-------------*/

FoamFile
{
    version         2.0;
```

```
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* * * * * * //

convertToMeters 1e-3;

vertices
(
    //Plane A: (S=Square, T=Tip, N=Nozzle, the location(number)
of the point is the number multipied with the wedgeangle)
    (xA 0 0) vlabel(A0)

    (xA rSquare 0) vlabel(AS0)
    (xA rSquare calc(rSquare*sin(wedgeAngle*pi/180.0)))
vlabel(AS01)
    (xA calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(AS1)
    (xA calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(AS12)
    (xA calc(rSquare*cos(2*wedgeAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(AS2)

    (xA rTip 0) vlabel(AT0)
    (xA calc(rTip*cos((1/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(AT01)
    (xA calc(rTip*cos(wedgeAngle*pi/180.0))
calc(rTip*sin(wedgeAngle*pi/180.0))) vlabel(AT1)
    (xA calc(rTip*cos((3/2)*wedgeAngle*pi/180.0))
calc(rTip*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(AT12)
    (xA calc(rTip*cos(2*wedgeAngle*pi/180.0))
calc(rTip*sin(2*wedgeAngle*pi/180.0))) vlabel(AT2)

    (xA rNozzle 0) vlabel(AN0)
    (xA calc(rNozzle*cos((1/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(AN01)
    (xA calc(rNozzle*cos(wedgeAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(AN1)
    (xA calc(rNozzle*cos((3/2)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(AN12)
    (xA calc(rNozzle*cos(2*wedgeAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(AN2)

    (xA rExterior 0) vlabel(AE0)
    (xA calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(AE01)
    (xA calc(rExterior*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(AE1)
    (xA calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(AE12)
```

```
    (xA calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(AE2)


    //Plane T:
    (xT 0 0) vlabel(T0)

    (calc(xT-rSquare*sin(tiltAngle*pi/180.0))
calc(rSquare*cos(tiltAngle*pi/180.0)) 0) vlabel(TS0)
    (calc(xT-rSquare*sin(tiltAngle*pi/180.0))
calc(rSquare*cos(tiltAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(TS01)
    (xT calc(rSquare*cos(wedgeAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(TS1)
    (calc(xT+rSquare*sin(tiltAngle*pi/180.0))
calc(rSquare*cos(2*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0))
calc(rSquare*sin(wedgeAngle*pi/180.0))) vlabel(TS12)
    (calc(xT+rSquare*sin(tiltAngle*pi/180.0))
calc(rSquare*cos(2*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0))
calc(rSquare*sin(2*wedgeAngle*pi/180.0))) vlabel(TS2)

    (calc(xT-rTip*sin(tiltAngle*pi/180.0))
calc(rTip*cos(tiltAngle*pi/180.0)) 0) vlabel(TT0)
    (calc(xT-
rTip*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rTip*cos((1/2)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0)
) calc(rTip*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(TT01)
    (xT calc(rTip*cos(wedgeAngle*pi/180.0))
calc(rTip*sin(wedgeAngle*pi/180.0))) vlabel(TT1)

(calc(xT+rTip*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/18
0.0))
calc(rTip*cos((3/2)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0)
) calc(rTip*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(TT12)
    (calc(xT+rTip*sin(tiltAngle*pi/180.0))
calc(rTip*cos(2*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0))
calc(rTip*sin(2*wedgeAngle*pi/180.0))) vlabel(TT2)

    (calc(xT-rTip*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos(tiltAngle*pi/180.0)) 0) vlabel(TN0)
    (calc(xT-
rTip*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180
.0)) calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(TN01)
    (xT calc(rTip*cos(wedgeAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(TN1)

(calc(xT+rTip*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/18
0.0))
calc(rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180
.0)) calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(TN12)
```

```
    (calc(xT+rTip*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos(2*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(TN2)

    (calc(xT-rTip*sin(tiltAngle*pi/180.0)) rExterior 0)
vlabel(TE0)
    (calc(xT-
rTip*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(TE01)
    (xT calc(rTip*cos(wedgeAngle*pi/180.0))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(TE1)

(calc(xT+rTip*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/18
0.0)) calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(TE12)
    (calc(xT+rTip*sin(tiltAngle*pi/180.0))
calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(TE2)


    //Plane C:
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT) calc((xC-
xT)*sin(tiltAngle*pi/180.0)) 0) vlabel(C0)

    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*sin(tiltAngle*pi/180.0)) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(tiltAngle*pi/180.0))
0) vlabel(CC0)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(CC01)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rCathode*cos(wedgeAngle*pi/180.0))+(xC-
xT)*sin(tiltAngle*pi/180.0))
calc(rCathode*sin(wedgeAngle*pi/180.0))) vlabel(CC1)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(CC12)
    (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(2*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(2*wedgeAngle*pi/180.0))) vlabel(CC2)
```

```
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*sin(tiltAngle*pi/180.0))) calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(tiltAngle*pi/180.0)) 0)
vlabel(CN0)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(CN01)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT))
calc((rNozzle*cos(wedgeAngle*pi/180.0))+(xC-
xT)*sin(tiltAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(CN1)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(CN12)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(2*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(CN2)

    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*sin(tiltAngle*pi/180.0))) rExterior 0) vlabel(CE0)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(CE01)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT))
calc((rExterior*cos(wedgeAngle*pi/180.0)))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(CE1)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(CE12)
    (calc((xC/xN)*(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(CE2)


    //Plane N:
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT) calc((xN-
xT)*sin(tiltAngle*pi/180.0)) 0) vlabel(N0)

    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*sin(tiltAngle*pi/180.0)) calc(((xN-
```

```
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(tiltAngle*pi/180.0))
0) vlabel(NC0)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(NC01)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT)
calc((rCathode*cos(wedgeAngle*pi/180.0))+(xN-
xT)*sin(tiltAngle*pi/180.0))
calc(rCathode*sin(wedgeAngle*pi/180.0))) vlabel(NC1)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(NC12)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(2*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(2*wedgeAngle*pi/180.0))) vlabel(NC2)

    (calc((((xN-xT)*cos(tiltAngle*pi/180.0)+xT)-
rNozzle*sin(tiltAngle*pi/180.0))) calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(tiltAngle*pi/180.0)) 0)
vlabel(NN0)
    (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(NN01)
    (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT))
calc((rNozzle*cos(wedgeAngle*pi/180.0))+(xN-
xT)*sin(tiltAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(NN1)
    (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(NN12)
    (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(2*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(NN2)
```

```
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*sin(tiltAngle*pi/180.0))) rExterior 0) vlabel(NE0)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((1/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(NE01)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT))
calc((rExterior*cos(wedgeAngle*pi/180.0)))
calc(rExterior*sin(wedgeAngle*pi/180.0))) vlabel(NE1)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((3/2)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(NE12)
    (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(rExterior*cos(2*wedgeAngle*pi/180.0))
calc(rExterior*sin(2*wedgeAngle*pi/180.0))) vlabel(NE2)


    //Plane I:
    (calc((xI-xT)*cos(tiltAngle*pi/180.0)+xT) calc((xI-
xT)*sin(tiltAngle*pi/180.0)) 0) vlabel(I0)

    (calc((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*sin(tiltAngle*pi/180.0)) calc((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(tiltAngle*pi/180.0))
0) vlabel(IC0)
    (calc((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(IC01)
    (calc((xI-xT)*cos(tiltAngle*pi/180.0)+xT)
calc((rCathode*cos(wedgeAngle*pi/180.0))+(xI-
xT)*sin(tiltAngle*pi/180.0))
calc(rCathode*sin(wedgeAngle*pi/180.0))) vlabel(IC1)
    (calc((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/2)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(IC12)
    (calc((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos(2*wedgeAngle*pi/180.0)
*cos(tiltAngle*pi/180.0))
calc(rCathode*sin(2*wedgeAngle*pi/180.0))) vlabel(IC2)

    (calc((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*sin(tiltAngle*pi/180.0)) calc((xI-
```

```
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(tiltAngle*pi/180.0)) 0)
vlabel(IN0)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/2)*wedgeAngle*pi/180.0))) vlabel(IN01)
    (calc((xI-xT)*cos(tiltAngle*pi/180.0)+xT)
calc((rNozzle*cos(wedgeAngle*pi/180.0))+(xI-
xT)*sin(tiltAngle*pi/180.0))
calc(rNozzle*sin(wedgeAngle*pi/180.0))) vlabel(IN1)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/2)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/2)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/2)*wedgeAngle*pi/180.0))) vlabel(IN12)
    (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos(2*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos(2*wedgeAngle*pi/180.0)*
cos(tiltAngle*pi/180.0))
calc(rNozzle*sin(2*wedgeAngle*pi/180.0))) vlabel(IN2)


);

// Defining blocks:
blocks
(

    // block0
    hex (A0 T0 TS0 AS0 AS1 TS1 TS01 AS01)
    (xTipNumberOfCells rTipNumberOfCells rTipNumberOfCells)
    simpleGrading (1 1 1)

    // block1
    hex (AS2 TS2 T0 A0 AS12 TS12 TS1 AS1)
    (xTipNumberOfCells rTipNumberOfCells rTipNumberOfCells)
    simpleGrading (1 1 1)

    // block2
    hex (AS01 TS01 TS0 AS0 AT01 TT01 TT0 AT0)
    (xTipNumberOfCells rTipNumberOfCells rTipNumberOfCells)
    simpleGrading (1 1 1)

    // block3
    hex (AS1 TS1 TS01 AS01 AT1 TT1 TT01 AT01)
    (xTipNumberOfCells rTipNumberOfCells rTipNumberOfCells)
    simpleGrading (1 1 1)

    // block4
```

```
    hex (AS12 TS12 TS1 AS1 AT12 TT12 TT1 AT1)
    (xTipNumberOfCells rTipNumberOfCells rTipNumberOfCells)
    simpleGrading (1 1 1)

    // block5
    hex (AS2 TS2 TS12 AS12 AT2 TT2 TT12 AT12)
    (xTipNumberOfCells rTipNumberOfCells rTipNumberOfCells)
    simpleGrading (1 1 1)

    // block6
    hex (AT01 TT01 TT0 AT0 AN01 TN01 TN0 AN0)
    (xTipNumberOfCells rTipNumberOfCells rCathodeNumberOfCells)
    simpleGrading (1 1 rGradingCathode)

    // block7
    hex (AT1 TT1 TT01 AT01 AN1 TN1 TN01 AN01)
    (xTipNumberOfCells rTipNumberOfCells rCathodeNumberOfCells)
    simpleGrading (1 1 rGradingCathode)

    // block8
    hex (AT12 TT12 TT1 AT1 AN12 TN12 TN1 AN1)
    (xTipNumberOfCells rTipNumberOfCells rCathodeNumberOfCells)
    simpleGrading (1 1 rGradingCathode)

    // block9
    hex (AT2 TT2 TT12 AT12 AN2 TN2 TN12 AN12)
    (xTipNumberOfCells rTipNumberOfCells rCathodeNumberOfCells)
    simpleGrading (1 1 rGradingCathode)

    // block10
    hex (AN01 TN01 TN0 AN0 AE01 TE01 TE0 AE0)
    (xTipNumberOfCells rTipNumberOfCells rOutletNumberOfCells)
    simpleGrading (1 1 rGradingOutlet)

    // block11
    hex (AN1 TN1 TN01 AN01 AE1 TE1 TE01 AE01)
    (xTipNumberOfCells rTipNumberOfCells rOutletNumberOfCells)
    simpleGrading (1 1 rGradingOutlet)

    // block12
    hex (AN12 TN12 TN1 AN1 AE12 TE12 TE1 AE1)
    (xTipNumberOfCells rTipNumberOfCells rOutletNumberOfCells)
    simpleGrading (1 1 rGradingOutlet)

    // block13
    hex (AN2 TN2 TN12 AN12 AE2 TE2 TE12 AE12)
    (xTipNumberOfCells rTipNumberOfCells rOutletNumberOfCells)
    simpleGrading (1 1 rGradingOutlet)

    // block14
    hex (TT01 CC01 CC0 TT0 TN01 CN01 CN0 TN0)
    (xCathode2NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
```

```
    simpleGrading (xGradingCathode2 1 rGradingCathode)

    // block15
    hex (TT1 CC1 CC01 TT01 TN1 CN1 CN01 TN01)
    (xCathode2NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingCathode)

    // block16
    hex (TT12 CC12 CC1 TT1 TN12 CN12 CN1 TN1)
    (xCathode2NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingCathode)

    // block17
    hex (TT2 CC2 CC12 TT12 TN2 CN2 CN12 TN12)
    (xCathode2NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingCathode)

    // block18
    hex (TN01 CN01 CN0 TN0 TE01 CE01 CE0 TE0)
    (xCathode2NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingOutlet)

    // block19
    hex (TN1 CN1 CN01 TN01 TE1 CE1 CE01 TE01)
    (xCathode2NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingOutlet)

    // block20
    hex (TN12 CN12 CN1 TN1 TE12 CE12 CE1 TE1)
    (xCathode2NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingOutlet)

    // block21
    hex (TN2 CN2 CN12 TN12 TE2 CE2 CE12 TE12)
    (xCathode2NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode2 1 rGradingOutlet)

    // block22
    hex (CC01 NC01 NC0 CC0 CN01 NN01 NN0 CN0)
    (xCathode1NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingCathode)

    // block23
    hex (CC1 NC1 NC01 CC01 CN1 NN1 NN01 CN01)
```

```
    (xCathode1NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingCathode)

    // block24
    hex (CC12 NC12 NC1 CC1 CN12 NN12 NN1 CN1)
    (xCathode1NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingCathode)

    // block25
    hex (CC2 NC2 NC12 CC12 CN2 NN2 NN12 CN12)
    (xCathode1NumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingCathode)

    // block26
    hex (CN01 NN01 NN0 CN0 CE01 NE01 NE0 CE0)
    (xCathode1NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingOutlet)

    // block27
    hex (CN1 NN1 NN01 CN01 CE1 NE1 NE01 CE01)
    (xCathode1NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingOutlet)

    // block28
    hex (CN12 NN12 NN1 CN1 CE12 NE12 NE1 CE1)
    (xCathode1NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingOutlet)

    // block29
    hex (CN2 NN2 NN12 CN12 CE2 NE2 NE12 CE12)
    (xCathode1NumberOfCells rTipNumberOfCells
rOutletNumberOfCells)
    simpleGrading (xGradingCathode1 1 rGradingOutlet)

    // block30
    hex (NC01 IC01 IC0 NC0 NN01 IN01 IN0 NN0)
    (xCathodeInletNumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathodeInlet 1 rGradingCathode)

    // block31
    hex (NC1 IC1 IC01 NC01 NN1 IN1 IN01 NN01)
    (xCathodeInletNumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathodeInlet 1 rGradingCathode)

    // block32
```

```
    hex (NC12 IC12 IC1 NC1 NN12 IN12 IN1 NN1)
    (xCathodeInletNumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathodeInlet 1 rGradingCathode)

    // block33
    hex (NC2 IC2 IC12 NC12 NN2 IN2 IN12 NN12)
    (xCathodeInletNumberOfCells rTipNumberOfCells
rCathodeNumberOfCells)
    simpleGrading (xGradingCathodeInlet 1 rGradingCathode)


);

edges
(
    //Plane A:
    line A0 AS0
    line AS0 AS01
    line AS01 AS1
    line AS1 A0
    line AS1 AS12
    line AS12 AS2
    line AS2 A0

    line AS0 AT0
    line AS01 AT01
    line AS1 AT1
    line AS12 AT12
    line AS2 AT2

    arc AT0 AT01 (xA calc(rTip*cos((1/4)*wedgeAngle*pi/180.0))
calc(rTip*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc AT01 AT1 (xA calc(rTip*cos((3/4)*wedgeAngle*pi/180.0))
calc(rTip*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc AT1 AT12 (xA calc(rTip*cos((5/4)*wedgeAngle*pi/180.0))
calc(rTip*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc AT12 AT2 (xA calc(rTip*cos((7/4)*wedgeAngle*pi/180.0))
calc(rTip*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees

    line AT0 AN0
    line AT01 AN01
    line AT1 AN1
    line AT12 AN12
    line AT2 AN2

    arc AN0 AN01 (xA
calc(rNozzle*cos((1/4)*wedgeAngle*pi/180.0))
```

```
calc(rNozzle*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc AN01 AN1 (xA
calc(rNozzle*cos((3/4)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc AN1 AN12 (xA
calc(rNozzle*cos((5/4)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc AN12 AN2 (xA
calc(rNozzle*cos((7/4)*wedgeAngle*pi/180.0))
calc(rNozzle*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees

    line AN0 AE0
    line AN01 AE01
    line AN1 AE1
    line AN12 AE12
    line AN2 AE2

    arc AE0 AE01 (xA
calc(rExterior*cos((1/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc AE01 AE1 (xA
calc(rExterior*cos((3/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc AE1 AE12 (xA
calc(rExterior*cos((5/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc AE12 AE2 (xA
calc(rExterior*cos((7/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees


    //Plane T:
    line T0 TS0
    line TS0 TS01
    line TS01 TS1
    line TS1 T0
    line TS1 TS12
    line TS12 TS2
    line TS2 T0

    line TS0 TT0
    line TS01 TT01
    line TS1 TT1
    line TS12 TT12
    line TS2 TT2
```

```
    arc TT0 TT01 (calc(xT-
rTip*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rTip*cos((1/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0)
) calc(rTip*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc TT01 TT1 (calc(xT-
rTip*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rTip*cos((3/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0)
) calc(rTip*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc TT1 TT12 (calc(xT-
rTip*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rTip*cos((5/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0)
) calc(rTip*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc TT12 TT2 (calc(xT-
rTip*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rTip*cos((7/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180.0)
) calc(rTip*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees

    line TT0 TN0
    line TT01 TN01
    line TT1 TN1
    line TT12 TN12
    line TT2 TN2

    arc TN0 TN01 (calc(xT-
rTip*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos((1/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180
.0)) calc(rNozzle*sin((1/4)*wedgeAngle*pi/180.0)))
//(1/4)*90 degrees
    arc TN01 TN1 (calc(xT-
rTip*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos((3/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180
.0)) calc(rNozzle*sin((3/4)*wedgeAngle*pi/180.0)))
//(3/4)*90 degrees
    arc TN1 TN12 (calc(xT-
rTip*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos((5/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180
.0)) calc(rNozzle*sin((5/4)*wedgeAngle*pi/180.0)))
//(5/4)*90 degrees
    arc TN12 TN2 (calc(xT-
rTip*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rNozzle*cos((7/4)*wedgeAngle*pi/180.0)*cos(tiltAngle*pi/180
.0)) calc(rNozzle*sin((7/4)*wedgeAngle*pi/180.0)))
//(7/4)*90 degrees

    line TN0 TE0
    line TN01 TE01
    line TN1 TE1
```

```
    line TN12 TE12
    line TN2 TE2

    arc TE0 TE01 (calc(xT-
rTip*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rExterior*cos((1/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc TE01 TE1 (calc(xT-
rTip*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rExterior*cos((3/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc TE1 TE12 (calc(xT-
rTip*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rExterior*cos((5/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc TE12 TE2 (calc(xT-
rTip*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(rExterior*cos((7/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees

    //Plane C:
    arc CC0 CC01 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc CC01 CC1 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc CC1 CC12  (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((5/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc CC12 CC2 (calc(((xC-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((7/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees
```

```
    line CC0 CN0
    line CC01 CN01
    line CC1 CN1
    line CC12 CN12
    line CC2 CN2

    arc CN0 CN01 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/4)*wedgeAngle*pi/180.0)))    //(1/4)*90
degrees
    arc CN01 CN1 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/4)*wedgeAngle*pi/180.0)))    //(3/4)*90
degrees
    arc CN1 CN12 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((5/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((5/4)*wedgeAngle*pi/180.0)))    //(5/4)*90
degrees
    arc CN12 CN2 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xC-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((7/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((7/4)*wedgeAngle*pi/180.0)))    //(7/4)*90
degrees

    line CN0 CE0
    line CN01 CE01
    line CN1 CE1
    line CN12 CE12
    line CN2 CE2

    arc CE0 CE01 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((1/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/4)*wedgeAngle*pi/180.0)))    //(1/4)*90
degrees
```

```
    arc CE01 CE1 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((3/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc CE1 CE12 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((5/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc CE12 CE2 (calc((xC/xN)*(((xN-
xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((7/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees


    //Plane N:
    arc NC0 NC01 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/4)*wedgeAngle*pi/180.0)))      //(1/4)*90
degrees
    arc NC01 NC1 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/4)*wedgeAngle*pi/180.0)))      //(3/4)*90
degrees
    arc NC1 NC12 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((5/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((5/4)*wedgeAngle*pi/180.0)))      //(5/4)*90
degrees
    arc NC12 NC2 (calc(((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((7/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((7/4)*wedgeAngle*pi/180.0)))      //(7/4)*90
degrees

    line NC0 NN0
    line NC01 NN01
    line NC1 NN1
```

```
    line NC12 NN12
    line NC2 NN2

    arc NN0 NN01 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/4)*wedgeAngle*pi/180.0)))    //(1/4)*90
degrees
    arc NN01 NN1 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/4)*wedgeAngle*pi/180.0)))    //(3/4)*90
degrees
    arc NN1 NN12 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((5/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((5/4)*wedgeAngle*pi/180.0)))    //(5/4)*90
degrees
    arc NN12 NN2 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)))
calc(((xN-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((7/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((7/4)*wedgeAngle*pi/180.0)))    //(7/4)*90
degrees

    line NN0 NE0
    line NN01 NE01
    line NN1 NE1
    line NN12 NE12
    line NN2 NE2

    arc NE0 NE01 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((1/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((1/4)*wedgeAngle*pi/180.0)))    //(1/4)*90
degrees
    arc NE01 NE1 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((3/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((3/4)*wedgeAngle*pi/180.0)))    //(3/4)*90
degrees
    arc NE1 NE12 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((5/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((5/4)*wedgeAngle*pi/180.0)))    //(5/4)*90
degrees
```

```
    arc NE12 NE2 (calc((((xN-xT)*cos(tiltAngle*pi/180.0))+xT-
rExterior*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0)
)) calc(rExterior*cos((7/4)*wedgeAngle*pi/180.0))
calc(rExterior*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees


    //Plane I:
    arc IC0 IC01 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((1/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((1/4)*wedgeAngle*pi/180.0)))      //(1/4)*90
degrees
    arc IC01 IC1 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((3/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((3/4)*wedgeAngle*pi/180.0)))      //(3/4)*90
degrees
    arc IC1 IC12 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((5/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((5/4)*wedgeAngle*pi/180.0)))      //(5/4)*90
degrees
    arc IC12 IC2 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rCathode*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rCathode*cos((7/4)*wedgeAngle*pi/18
0.0)*cos(tiltAngle*pi/180.0))
calc(rCathode*sin((7/4)*wedgeAngle*pi/180.0)))      //(7/4)*90
degrees

    line IC0 IN0
    line IC01 IN01
    line IC1 IN1
    line IC12 IN12
    line IC2 IN2

    arc IN0 IN01 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((1/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((1/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((1/4)*wedgeAngle*pi/180.0)))     //(1/4)*90
degrees
    arc IN01 IN1 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((3/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
```

```
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((3/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((3/4)*wedgeAngle*pi/180.0)))     //(3/4)*90
degrees
    arc IN1 IN12 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((5/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((5/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((5/4)*wedgeAngle*pi/180.0)))     //(5/4)*90
degrees
    arc IN12 IN2 (calc(((xI-xT)*cos(tiltAngle*pi/180.0))+xT-
rNozzle*cos((7/4)*wedgeAngle*pi/180.0)*sin(tiltAngle*pi/180.0))
calc(((xI-
xT)*sin(tiltAngle*pi/180.0))+rNozzle*cos((7/4)*wedgeAngle*pi/180
.0)*cos(tiltAngle*pi/180.0))
calc(rNozzle*sin((7/4)*wedgeAngle*pi/180.0)))     //(7/4)*90
degrees


    //From plane A to T:
    line A0 T0

    line AS0 TS0
    line AS01 TS01
    line AS1 TS1
    line AS12 TS12
    line AS2 TS2

    line AT0 TT0
    line AT01 TT01
    line AT1 TT1
    line AT12 TT12
    line AT2 TT2

    line AN0 TN0
    line AN01 TN01
    line AN1 TN1
    line AN12 TN12
    line AN2 TN2

    line AE0 TE0
    line AE01 TE01
    line AE1 TE1
    line AE12 TE12
    line AE2 TE2


    //From plane T to C:
    line TT0 CC0
    line TT01 CC01
    line TT1 CC1
```

```
    line TT12 CC12
    line TT2 CC2

    line TN0 CN0
    line TN01 CN01
    line TN1 CN1
    line TN12 CN12
    line TN2 CN2

    line TE0 CE0
    line TE01 CE01
    line TE1 CE1
    line TE12 CE12
    line TE2 CE2


    //From plane C to N:
    line CC0 NC0
    line CC01 NC01
    line CC1 NC1
    line CC12 NC12
    line CC2 NC2

    line CN0 NN0
    line CN01 NN01
    line CN1 NN1
    line CN12 NN12
    line CN2 NN2

    line CE0 NE0
    line CE01 NE01
    line CE1 NE1
    line CE12 NE12
    line CE2 NE2


    //From plane N to I:
    line NC0 IC0
    line NC01 IC01
    line NC1 IC1
    line NC12 IC12
    line NC2 IC2

    line NN0 IN0
    line NN01 IN01
    line NN1 IN1
    line NN12 IN12
    line NN2 IN2


);

// Defining patches:
```

```
patches
(

    patch anodeTipWall
    (
        (AS0 A0 AS1 AS01)
        (A0 AS2 AS12 AS1)
        (AS0 AS01 AT01 AT0)
        (AS01 AS1 AT1 AT01)
        (AS1 AS12 AT12 AT1)
        (AS12 AS2 AT2 AT12)
    )

    patch anodeWall
    (
        (AT0 AT01 AN01 AN0)
        (AT01 AT1 AN1 AN01)
        (AT1 AT12 AN12 AN1)
        (AT12 AT2 AN2 AN12)
    )

    patch anodeOutletWall
    (
        (AN0 AN01 AE01 AE0)
        (AN01 AN1 AE1 AE01)
        (AN1 AN12 AE12 AE1)
        (AN12 AN2 AE2 AE12)
    )

   patch OutletAnode
    (
         (AE0 AE01 TE01 TE0)
         (AE01 AE1 TE1 TE01)
         (AE1 AE12 TE12 TE1)
         (AE12 AE2 TE2 TE12)
    )

   patch OutletNozzle
    (
         (TE0 TE01 CE01 CE0)
         (TE01 TE1 CE1 CE01)
         (TE1 TE12 CE12 CE1)
         (TE12 TE2 CE2 CE12)
         (CE0 CE01 NE01 NE0)
         (CE01 CE1 NE1 NE01)
         (CE1 CE12 NE12 NE1)
         (CE12 CE2 NE2 NE12)
    )

   patch NozzleWall
    (
        (NN0 NE0 NE01 NN01)
        (NN01 NE01 NE1 NN1)
```

```
        (NN1 NE1 NE12 NN12)
        (NN12 NE12 NE2 NN2)
        (NN01 IN01 IN0 NN0)
        (NN1 IN1 IN01 NN01)
        (NN12 IN12 IN1 NN1)
        (NN2 IN2 IN12 NN12)
    )

    patch InletSides
    (
        (IC01 IC0 IN0 IN01)
        (IC1 IC01 IN01 IN1)
        (IC12 IC1 IN1 IN12)
        (IC2 IC12 IN12 IN2)
    )

    patch CathodeWall1
    (
        (IC0 IC01 NC01 NC0)
        (IC01 IC1 NC1 NC01)
        (IC1 IC12 NC12 NC1)
        (IC12 IC2 NC2 NC12)
        (NC0 NC01 CC01 CC0)
        (NC01 NC1 CC1 CC01)
        (NC1 NC12 CC12 CC1)
        (NC12 NC2 CC2 CC12)
    )

    patch CathodeWall2
    (
        (TT0 CC0 CC01 TT01)
        (TT01 CC01 CC1 TT1)
        (TT1 CC1 CC12 TT12)
        (TT12 CC12 CC2 TT2)
    )

    patch TipWall
    (
        (TS0 TS01 TS1 T0)
        (T0 TS1 TS12 TS2)
        (TS0 TT0 TT01 TS01)
        (TS01 TT01 TT1 TS1)
        (TS1 TT1 TT12 TS12)
        (TS12 TT12 TT2 TS2)
    )

    symmetryPlane halfModelPlane
    (
        (AE2 AN2 TN2 TE2)
        (AN2 AT2 TT2 TN2)
        (AT2 AS2 TS2 TT2)
        (AS2 A0 T0 TS2)
        (A0 AS0 TS0 T0)
```

```
        (AS0 AT0 TT0 TS0)
        (AT0 AN0 TN0 TT0)
        (AN0 AE0 TE0 TN0)
        (TE2 TN2 CN2 CE2)
        (TN2 TT2 CC2 CN2)
        (TT0 TN0 CN0 CC0)
        (TN0 TE0 CE0 CN0)
        (CE2 CN2 NN2 NE2)
        (CN2 CC2 NC2 NN2)
        (CC0 CN0 NN0 NC0)
        (CN0 CE0 NE0 NN0)
        (NN2 NC2 IC2 IN2)
        (NC0 NN0 IN0 IC0)
    )

);

mergePatchPairs
(
);

//
*********************************************************************
********* //
```

G. Heat transfer towards the base metal – additional figures

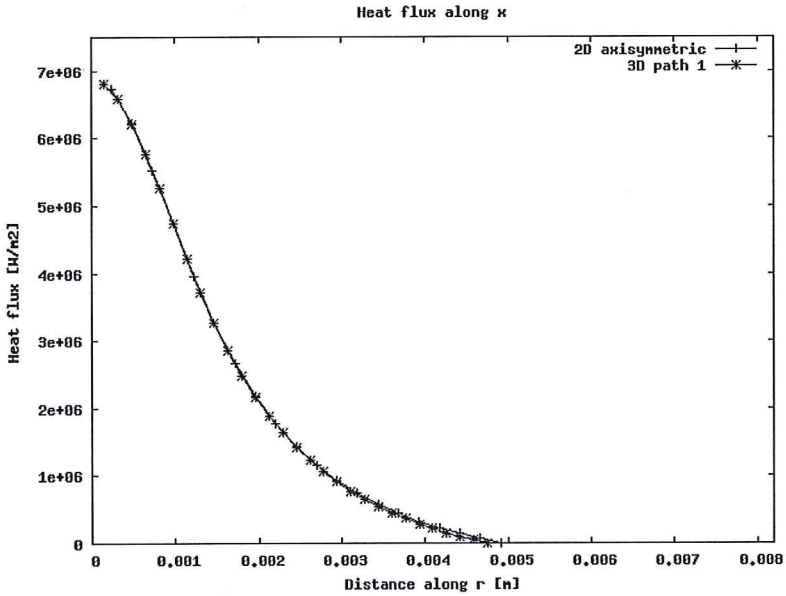


Figure 49: The heat flux along the x direction plotted for the 2D axisymmetric simulation and along path 1 in the 3D simulation

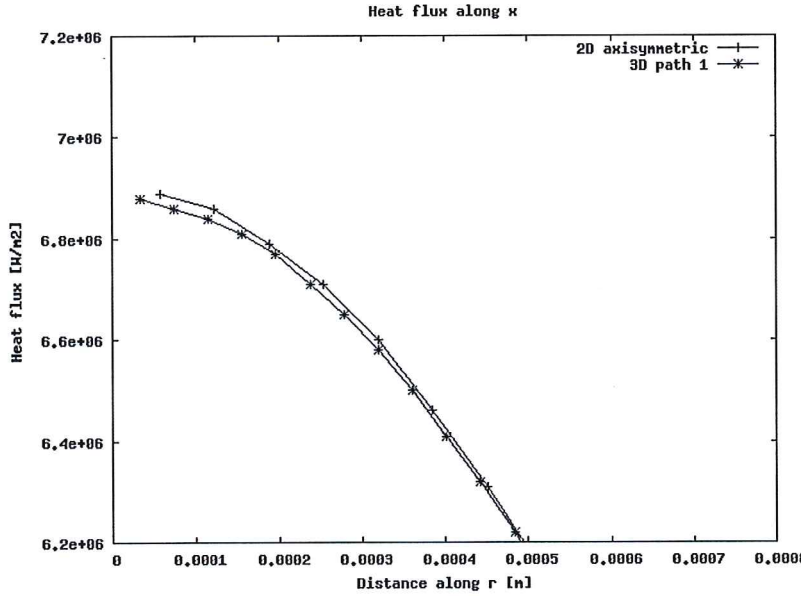


Figure 50: The heat flux along the x direction plotted for the 2D axisymmetric simulation and along path 1 in the 3D simulation zoomed from r = 0 mm to r = 0.8 mm.

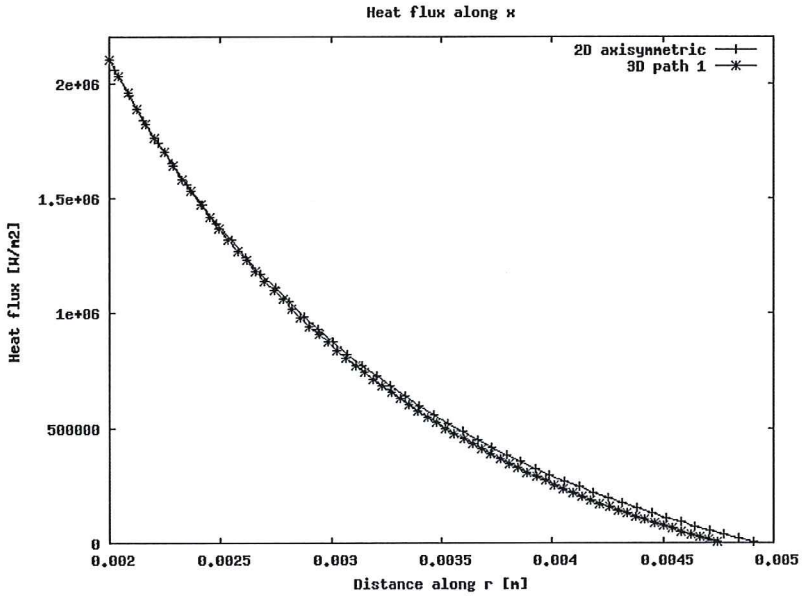


Figure 51: The heat flux along the x direction plotted for the 2D axisymmetric simulation and along path 1 in the 3D simulation zoomed from r = 2 mm to r= 5 mm.

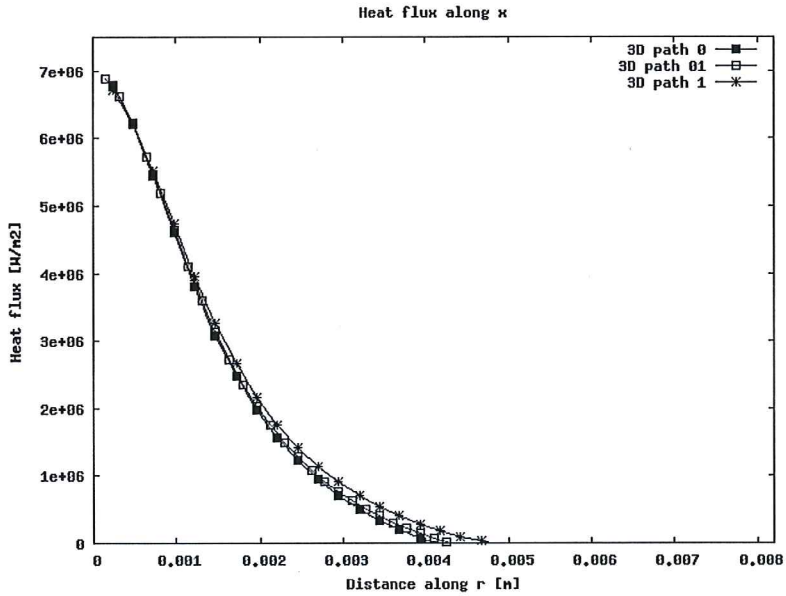


Figure 52: The heat flux along the x direction plotted along paths 0, 01 and 1 in the 3D simulation

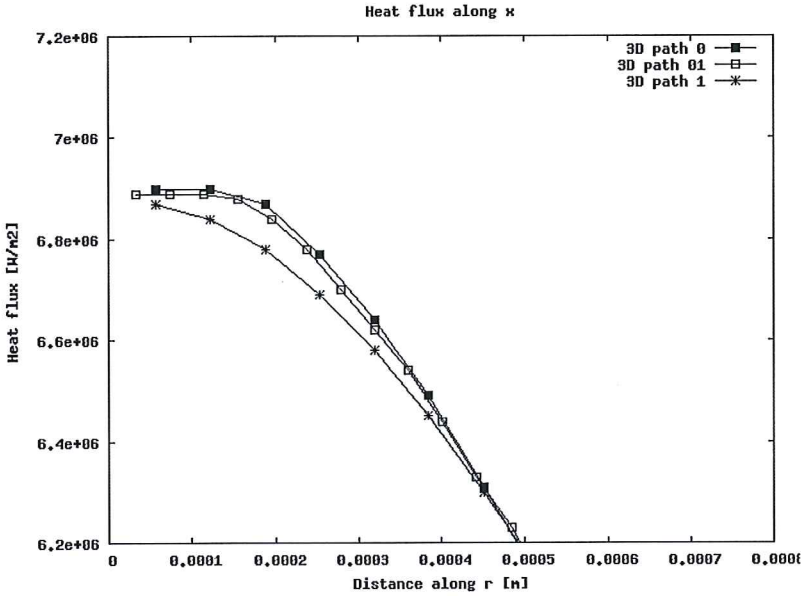


Figure 53: The heat flux along the x direction plotted along paths 0, 01 and 1 in the 3D simulation zoomed from r = 0 to r = 0.8 mm

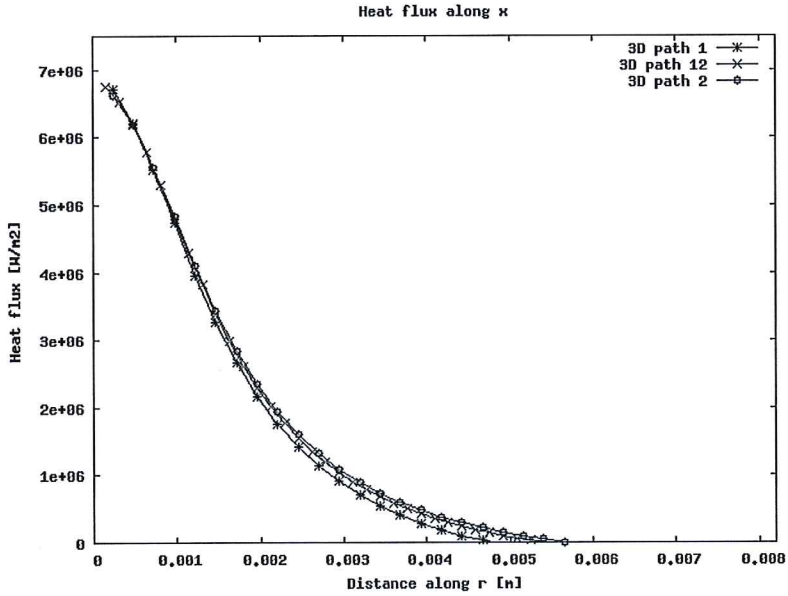


Figure 54: The heat flux along the x direction plotted along paths 1, 12 and 2 in the 3D simulation

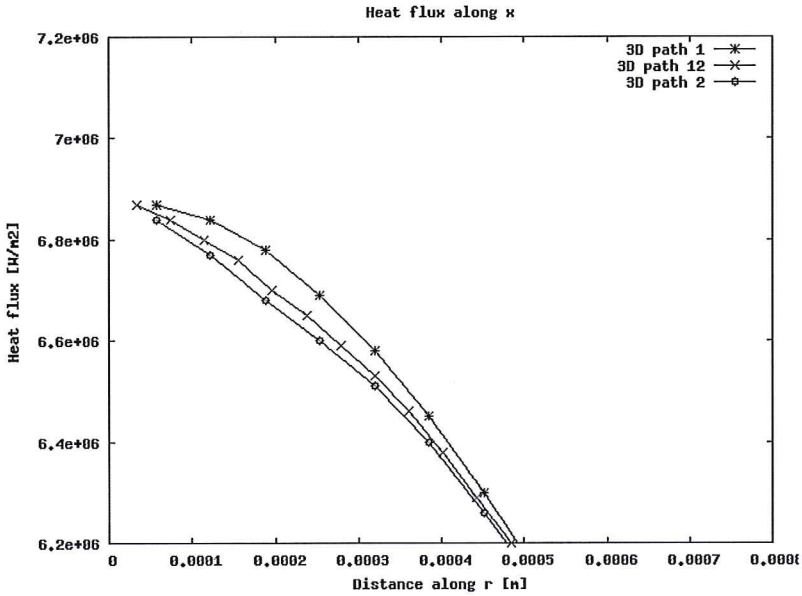


Figure 55: The heat flux along the x direction plotted along paths 1, 12 and 2 in the 3D simulation zoomed from r = 0 to r = 0.8 mm

H. New boundary conditions

To increase the accuracy of the temperature boundary condition the assumption of zero gradient on the patches around the cathode could be replaced by a temperature distribution changind linearly along the electrode as similar input data can be found in the literature. The temperature boundary conditions on the patches around the electrode could thus be implemented in a new way.

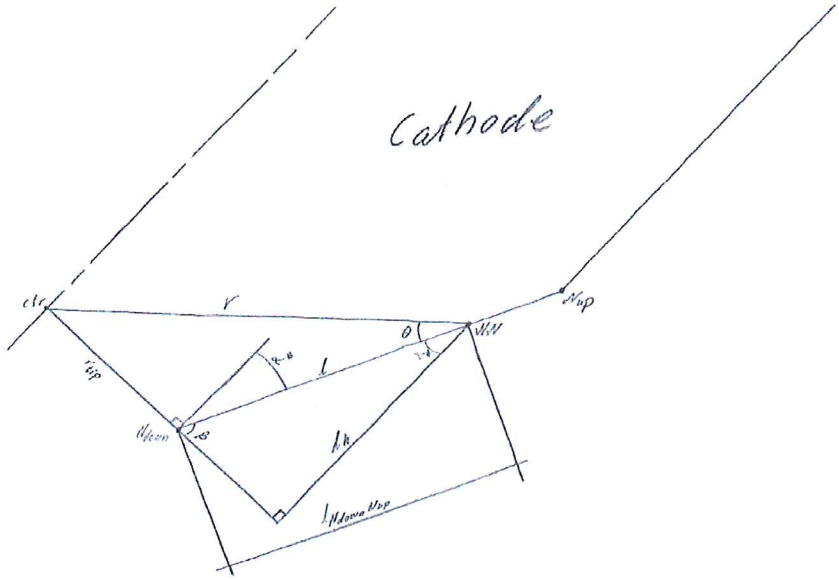


Figure 56: An example of how the implementation of the temperature boundary condition on the patch "CathodeWall2" is implemented for the GTAW with a non-zero tilt angle

For the patch "CathodeWall2" shown in Figure 15 it is an example of how the temperature at a point NN can be calculated in Figure 56. The temperature is known in the points N_{down} and N_{up} as can be seen in Figure 56.

$$T(l) = \frac{T_{up} - T_{down}}{l_{N_{down}N_{up}}} l + T_{down} \qquad (31)$$

From (31) the temperature at node NN can be calculated by linear interpolation on the distance from point N_{down} noted by l in (31) and can be seen in Figure 56. T_{up} is 3200 K and T_{down} is 20000 K in (31) and are the temperatures in the points N_{up} and N_{down} in Figure 56. The distance between the points N_{down} and N_{up} is denoted by $l_{N_{down}N_{up}}$ in (31) and can also be seen in Figure 56. The unknown quantity in (31) is the length from point N_{down} noted by l. It can be calculated by use of the two right-angled triangles shown in Figure 56.

$$l_h = r * \cos(\theta + \alpha^*) = l * \cos(\alpha^*) \qquad (32)$$

In (32) l_h is the height in both the two right-angled triangles shown in Figure 56. The angle α^* is half of the angle of the tip of the cathode shown in Figure 14. r is the distance between the know centrum of the electrode noted by ctr in Figure 56 and the point which the temperature should be calculated for NN.

$$r = \sqrt{(x_{NN} - x_{ctr})^2 + (y_{NN} - y_{ctr})^2 + (z_{NN} - z_{ctr})^2} \qquad (33)$$

The Euclidean distance used to calculate r in (33) is between the points NN with coordinates (x_{NN}, y_{NN}, z_{NN}) and ctr with coordinates (x_{ctr}, y_{ctr}, z_{ctr}). To obtain the angle θ seen in Figure 56 the sine formula is used in the triangle with base r.

$$\frac{\sin(\alpha^*+90^\circ)}{r} = \frac{\sin(\theta)}{r_{tip}} \qquad (34)$$

In (34) r_{tip} is the radius of the tip of the electrode.

$$l = \frac{r*\cos\left(\sin^{-1}\left(\frac{r_{tip}*\sin(\alpha+90^\circ)}{r}\right)+\alpha\right)}{\cos(\alpha)} \qquad (35)$$

By the use of equation (32) to (34) and rearranging the equation for l can be obtained. The equation to obtain l is shown is (35).

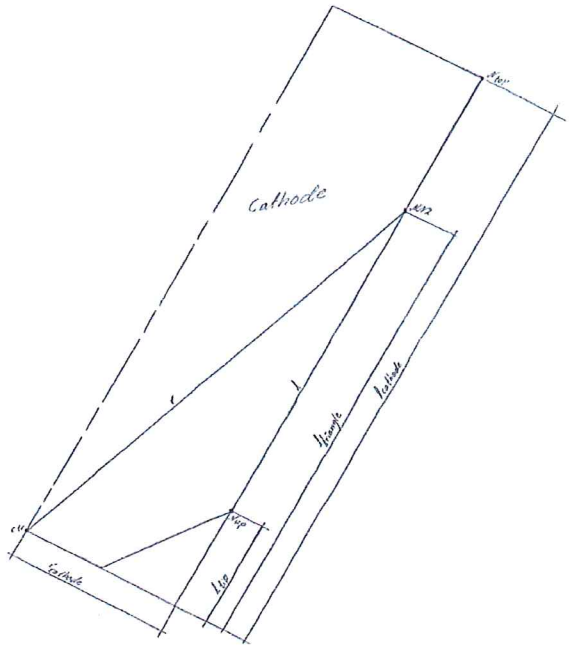


Figure 57: An example of how the implementation of the temperature boundary condition on the patch "CathodeWall1" is implemented for the GTAW with a non-zero tilt angle

An example of how the temperature boundary condition on the patch "CathodeWall1" can be obtained for a point NN2 can be seen in Figure 57. At this patch the temperature at point N_{top} and N_{up} is known from experiments to be 300 K and 3200 K (the same as used when calculating the temperature at patch "CathodeWall2"). From experiments it is also known that the change in temperature between these points is linear

$$T(l) = \frac{T_{top}-T_{up}}{l_{cathode}-l_{tip}} l + T_{up} \qquad (36)$$

In (36) T_{top} is the temperature at point N_{top} and T_{up} at point N_{up}. The length of the cathode and tip of the cathode are in (36) denoted by $l_{cathode}$ and l_{tip}. The wanted length l can be

obtained from the Pythagorean theorem for the triangle with base r as can be seen in Figure 57.

$$r^2 = r_{cathode}{}^2 + \left(l + l_{tip}\right)^2 \qquad (37)$$

The radius of the cathode is in (37) denoted by $r_{cathode}$. The distance between the points NN2 with coordinates (x_{NN2}, y_{NN2}, z_{NN2}) and the centrum of the electrode ctr with coordinates (x_{ctr}, y_{ctr}, z_{ctr}) is denoted r in Figure 57 and obtained from the Euclidean distance.

$$r = \sqrt{(x_{NN2} - x_{ctr})^2 + (y_{NN2} - y_{ctr})^2 + (z_{NN2} - z_{ctr})^2} \qquad (38)$$

By the use of equations (37) and (38) and solve for l the temperature at point NN2 can be obtained from (36).

To increase the accuracy of the results obtained from the GTAW simulation the inviscid inlet flow of argon can be further developed by assuming a laminar viscous flow. When assuming laminar viscous flow the velocity need to fulfill the no slip condition at the sides of the inlet flow and the boundary condition needs to be implemented in a new way [14]. Then defining the boundary condition the velocity in each point in the patch "InletSides" as can be seen in Figure 15 need to be found.

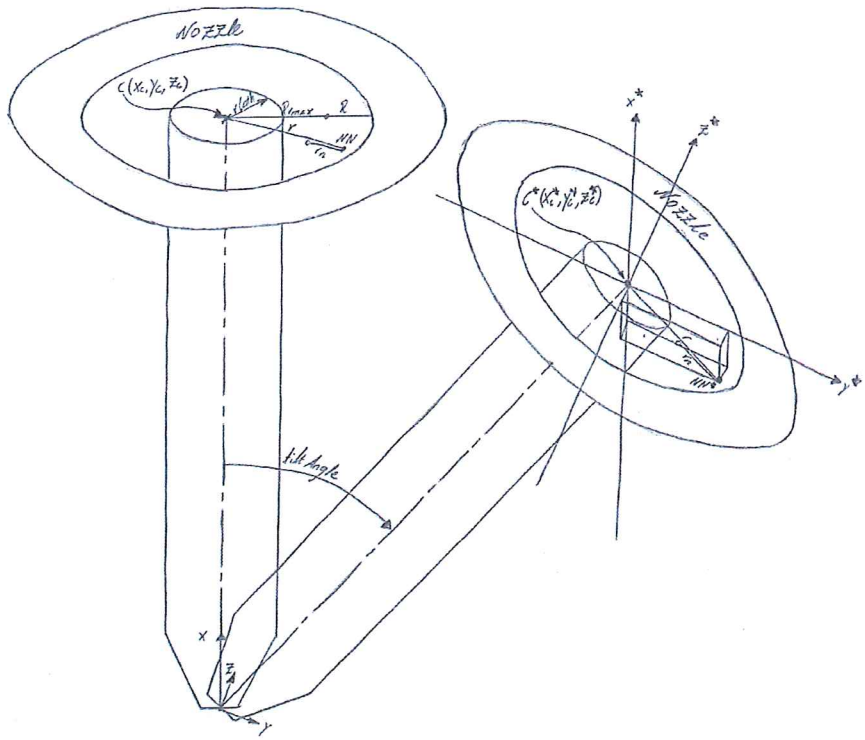


Figure 58: An example of how the implementation of the velocity boundary condition is implemented for the GTAW with a non-zero tilt angle

The velocity for each point is calculated by using the profile of the fully developed flow which is parabolic

$$u(r_n) = u_{max}\left(1 - \left(\frac{r_n}{R}\right)^2\right) \qquad (39)$$

The maximum velocity as denoted by u_{max} in (39) can be found in the middle between the electrode and the nozzle [14]. The distance from the centrum of the electrode to where the maximum velocity occurs is denoted by $R_{v_{max}}$ in Figure 58. The distance from $R_{v_{max}}$ to the nozzle is denoted by R in (39) and can be seen in Figure 58. The critical distance for each point in the patch on the top of the computational domain i.e. the patch "InletSides" as can be seen in Figure 15 is the distance from $R_{v_{max}}$ which is denoted by r_n in (39) and seen in Figure 58. In OpenFOAM the input to the implementation of the velocity boundary condition is the Cartesian coordinates of the points on the patch "InletSides" [15]. The implementation is described by an example as can be seen in Figure 58 which shows how a point NN with the coordinates (x_{NN}, y_{NN}, z_{NN}) is moved by the tilt angle to a new location with the coordinates (x^*_{NN}, y^*_{NN}, z^*_{NN}) and renamed NN^*. The centrum of the electrode is also relocated from (x_C, y_C, z_C) to (x^*_C, y^*_C, z^*_C) and renamed from C to C^*. This new location can be calculated by equations (28) and (29) and the z-coordinate is the same. The Euclidean distance between the points NN^* and C^* can be calculated.

$$r = \sqrt{(x^*_{NN} - x^*_C)^2 + \left(y^*_{NN} - y^*_C\right)^2 + (z^*_{NN} - z^*_C)^2} \qquad (40)$$

The Euclidean distance is in (40) denoted by r which also can be seen in Figure 58 for both the tilted and untitled GTAW sketch. This distance is a further development of the Pythagorean formula in higher dimensions as can be seen by the orthotope attached to the points C^* and NN^*. The location of the points has no more any meaning.

$$r_n = r - R_{v_{max}} \qquad (41)$$

The distance $R_{v_{max}}$ can then be removed as seen in (41). The wanted distance in (39) is then obtained and can be used in the equation to obtain the velocity for that location. This is done for all the points in the patch "InletSides" to obtain the velocity at that current point.