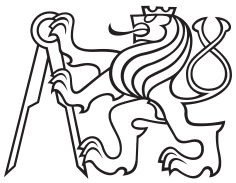
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Electromagnetic Field

Volumetric Method of Moments and Post-Processing of Results

Vojtěch Neuman

Supervisor: Doc. Ing. Miloslav Čapek Ph.D.
Supervisor–specialist: Doc. Ing. Lukáš Jelínek Ph.D.
May 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Neuman** Jméno: **Vojtěch** Osobní číslo: **465820**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektromagnetického pole**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Objemová metoda momentů a grafické zpracování jejích výsledků

Název bakalářské práce anglicky:

Volumetric method of moments and post-processing of results

Pokyny pro vypracování:

Popište princip metody momentů a její implementaci pro objemové zdrojové veličiny, včetně volby bázových a testovacích funkcí a jejich vlastností. Dále shrňte existující přístupy k diskretizaci struktur s nenulovým objemem do tetrahedronů. Zaměřte se na obecné, snadno dostupné, či implementovatelné postupy. Popište jak provádět booleovské operace a získat diskretizační mříž ve formátu NASTRAN. Zvolte jednu z možností a vytvořte nástroj/rozhraní, které umožní diskretizovat základní objekty (koule, válec), případně jejich booleovské produkty. V závěru projektu se věnujte integraci všech součástí do v současnosti na katedře vyvíjené objemové metody momentů.
Vedoucí projektu-specialista: doc. Ing. Lukáš Jelínek, Ph.D.

Seznam doporučené literatury:

[1] Schaubert, D. H., Wilton, D. R., Glisson, A. W.: A Tetrahedral Modeling Method for Electromagnetic Scattering by Arbitrarily Shaped Inhomogeneous Dielectric Bodies, IEEE Trans. Antennas and Propagation, Vol. 32, No. 1., pp. 77-85, Jan. 1984.
[2] Gmsh – A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities (2018), [on-line]: http://gmsh.info/
[3] Sancer, M. I., Sertel, K., Volakis, J. L.: On Volume Integral Equations, Vol. 54, No. 5, pp. 1488-1495, May 2006.
[4] Chew, W. Ch., Tong, M. S., Hu, B.: Integral Equation Methods for Electromagnetic and Elastic Waves, Morgan & Claypool Publishers, 2009.
[5] Makarov, S. N.: Antenna and EM Modeling with MATLAB, Wiley, 2002.
[6] MathWorks, MATLAB (2018), [on-line]: https://www.mathworks.com/products/matlab.html

Jméno a pracoviště vedoucí(ho) bakalářské práce:

doc. Ing. Miloslav Čapek, Ph.D., katedra elektromagnetického pole FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **08.02.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

doc. Ing. Miloslav Čapek, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Acknowledgements

I would like to thank my family, especially my mother and father, for their endless support at this hard times. Words cannot express how grateful I am for the all help and knowledge I have gained from my supervisors. I would also like to appreciate all the people who done the work which inspired mine. To FEL*i*ta, I am so glad you are here with me. And I wish the travelling salesman to finally solve all his problems.

Declaration

I declare that I have written submitted thesis by myself and that I have listed all information sources in accordance with Methodical Guideline on Compliance with Ethical Principles.

In Prague, 24 May 2019

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 24. května 2019

Abstract

The goal of this thesis is to describe some problems related to the implementation of volumetric method of moments. In particular, difficulties with discretization of the model with two suggested open-source solutions are described . The second part of the thesis deals with visualisation of tetrahedra-based results. The last part presents interface between MATLAB and LaTeX/Ti*k*Z generating source code for mesh, current density, and charge density plots. Within this part, an algorithm for investigation of direct visibility of triangles is portrayed. The results of the thesis are implemented in academic MATLAB toolbox AToM (Antenna Toobox for MATLAB) and in internal project focused on development of fast and efficient volumetric method of moments for non-homogeneous bodies.

Keywords: Numerical methods, Computational electromagnetism, Field scattering, Method of moments, Volumetric integral equations, Data visualisation, Simulations.

Supervisor: Doc. Ing. Miloslav Čapek Ph.D.
Faculty of Electrical Engineering,
Technická 2,
160 00 Praha 6

Abstrakt

Cílem této práce je popis problémů spojených s implementací objemové metody momentů. Konkrétně jsou zmíněny problémy spojené se získáním diskretizace tělesa a jejich dvě možná řešení. Druhá část práce se věnuje grafickému zobrazení výsledků na tetrahedronech. Poslední část popisuje rozhraní mezi MATLAB a LaTeX/Ti*k*Z převádějící grafické výsledky do `.pdf` formátu. Výsledky práce jsou implementovány do akademického toolboxu *AToM* (Antenna Toolbox for MATLAB) a do interního projektu zaměřeného na vývoj rychlého a efektivního solveru pro nehomogenní tělesa založeném na objemové metodě momentů.

Klíčová slova: Numerické metody, Výpočetní elektromagnetismus, Rozptylové pole, Metoda momentů, Objemové integrální rovnice, vykreslování, Simulace

Překlad názvu: Objemová metoda momentů a grafické zpracování jejich výsledků

Contents

Figures

Tables

Chapter 1

Introduction

Numerous analytic solution methods were developed since year 1873, when James Clerk Maxwell published his *A treatise on electricity and magnetism*. One of the most elegant results of classical electrodynamics is Gustav Mie's scattering theory, which describe scattering of a harmonic plane wave on a sphere and a cylinder. However, analytical approaches fail in general problems, where objects are arbitrarily shaped and contain inhomogeneous material. Such problems are only solvable via numerical methods which gained attention with the advent of new generation computers in 1970s. Two most common numerical methods [1], [2] are *Finite Elements Method* advantageous especially for closed problems and *Method of Moments*, which is powerful for open (radiating) problems.

The goal of an internal project, in this work referred as MOOPEL, of Computational Electromagnetic Group at the Department of Electromagnetic Field is implementation of electromagnetic full-wave simulator in MATLAB environment based on volumetric *Method of Moments*. Majority of functions are still under development and not publicly accessible. Chapter 2 describes physical and mathematical theory background required to solve the goals of the thesis. Discretization into tetrahedra, an important step within volumetric method of moments, is discussed in chapter 3, emphasising its problematic aspects. Main part of this work, post-processing of results, is presented in chapter 4.

This text is typeset in Donald Knuth's TEX typesetting system with use of Leslie Lamport's LATEX macros. All pictures and plots were done in Ti*k*Z and `pgfplot` packages. Customized Tomáš Hejda's `ctuthesis` is used as the document template. All codes are implemented in MATLAB language. Used colourmaps are taken from `brewermap()`. The MATLAB code is cited through `mcode` package.

Chapter 2

Volumetric Method of Moments

Volumetric Method of Moments (VMoM) is computational method for solving volumetric integral equations. Derivation of electric field equation for polarisation current and basic principles of Method of Moments (MoM) are described in this chapter. A choice of basis and testing functions and other technicalities like efficient evaluation of interaction integrals are introduced latter in this chapter.

2.1 Volumetric Integral Equations for Electrical Current

Permittivity is assumed in form of a scalar field $\varepsilon(\boldsymbol{r}) = \varepsilon_0 \varepsilon_\mathrm{r}(\boldsymbol{r})$ and permeability is taken as $\mu(\boldsymbol{r}) = \mu_0$, *i.e.*, the magnetic materials not considered. Further assumption is a time-harmonic steady state operation at angular frequency ω with time convention $\partial/\partial t \to \mathrm{i}\omega$, with i being imaginary unit. Consequently, equation for electric field $\boldsymbol{E}(\boldsymbol{r})$ expressed in terms of vector potential $\boldsymbol{A}(\boldsymbol{r})$ reads

$$\boldsymbol{E}(\boldsymbol{r}) = -\mathrm{i}\omega \left(\boldsymbol{A}(\boldsymbol{r}) + \frac{1}{k_0^2} \nabla\nabla \cdot \boldsymbol{A}(\boldsymbol{r}) \right), \tag{2.1}$$

where $k_0 = \omega\sqrt{\varepsilon_0 \mu_0}$ denotes free-space wave-number, and Lorentz gauge condition [3] is assumed.

Furthermore, the vector potential $\boldsymbol{A}(\boldsymbol{r})$ expressed due to current densities is

$$\boldsymbol{A}(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \int_{V'} \left(\boldsymbol{J}_\mathrm{s}(\boldsymbol{r}') + \mathrm{i}\omega \boldsymbol{P}(\boldsymbol{r}') \right) \frac{\mathrm{e}^{-\mathrm{i}k_0 \|\boldsymbol{r}-\boldsymbol{r}'\|}}{\|\boldsymbol{r}-\boldsymbol{r}'\|} \mathrm{d}V', \tag{2.2}$$

where the first integrand is directly proportional to known incident field $\boldsymbol{E}^\mathrm{inc}(\boldsymbol{r})$, and the second one is impressed field due to induced polarisation current in the object

$$\boldsymbol{J}_\mathrm{b}(\boldsymbol{r}) = \mathrm{i}\omega \boldsymbol{P}(\boldsymbol{r}) = \mathrm{i}\omega \left(\varepsilon(\boldsymbol{r}) - \varepsilon_0 \right) \boldsymbol{E}(\boldsymbol{r}). \tag{2.3}$$

Combining all these statements together with the use of vector identities, the final formulation reads

$$\left(1+\frac{1}{\kappa\left(\boldsymbol{r}\right)}\right)\boldsymbol{J}_{\mathrm{b}}\left(\boldsymbol{r}\right)=\mathrm{i}\omega\varepsilon_{0}\boldsymbol{E}^{\mathrm{inc}}\left(\boldsymbol{r}\right)+\nabla\times\nabla\times\int_{V'}\boldsymbol{J}_{\mathrm{b}}\left(\boldsymbol{r'}\right)\mathrm{G}\left(\boldsymbol{r},\boldsymbol{r'}\right)\mathrm{d}V' \quad (2.4)$$

where $\mathrm{G}\left(\boldsymbol{r},\boldsymbol{r'}\right)=\exp\left(-\mathrm{i}k_{0}\left\|\boldsymbol{r}-\boldsymbol{r'}\right\|\right)/4\pi\left\|\boldsymbol{r}-\boldsymbol{r'}\right\|$ stands for Green's function, $\kappa\left(\boldsymbol{r}\right)=\left(\varepsilon\left(\boldsymbol{r}\right)-\varepsilon_{0}\right)/\varepsilon_{0}$ defines contrast ratio, and polarisation current $\boldsymbol{J}_{\mathrm{b}}\left(\boldsymbol{r}\right)$ is taken as the unknown. Formula (2.4) is known as Volumetric Electrical Field Equation for Electrical Current (VEFIE-J), [4]. This formulation does not require any specific conditions for basis functions unlike the $\boldsymbol{D}\left(\boldsymbol{r}\right)$ and $\boldsymbol{E}\left(\boldsymbol{r}\right)$ formulations [5].

The evaluation of (2.4) for arbitrarily shaped objects is impossible without appropriate numerical treatment which is introduced in the following section.

2.2 Method of Moments

Method of Moments (MoM) is a procedure for solving systems in a form of

$$\mathscr{L}\,\boldsymbol{f}\left(\boldsymbol{r}\right)=\boldsymbol{g}\left(\boldsymbol{r}\right), \quad (2.5)$$

where $\mathscr{L}$ is a linear operator, $\boldsymbol{g}$ is a known excitation function, and $\boldsymbol{f}$ is a function to be found. Function $\boldsymbol{f}$ can be rewritten as a linear combination of N known and well-chosen basis functions $\boldsymbol{\psi_n}$ with N unknown coefficients f_n. Together with the use of linearity of operator $\mathscr{L}$, (2.5) expressed as

$$\sum_{n=1}^{N}f_{n}\mathscr{L}\boldsymbol{\psi}_{n}\left(\boldsymbol{r}\right)\simeq\boldsymbol{g}\left(\boldsymbol{r}\right), \quad (2.6)$$

where approximation becomes equality for $N\to\infty$.

The whole problem is converted into a system of linear equations with the use of $M=N$ known testing functions $\boldsymbol{w}_{m}\left(\boldsymbol{r}\right)$ by means of the inner product defined as

$$\left\langle\boldsymbol{\xi},\boldsymbol{\zeta}\right\rangle=\int_{V}\boldsymbol{\xi}^{*}\left(\boldsymbol{r}\right)\cdot\boldsymbol{\zeta}\left(\boldsymbol{r}\right)\mathrm{d}V,\quad\boldsymbol{r}\in V, \quad (2.7)$$

with $\boldsymbol{\xi}^{*}$ being complex conjugate of $\boldsymbol{\xi}$. The system (2.5) tested by functions $\boldsymbol{w}_{m}\left(\boldsymbol{r}\right)$ is then expressed as follows

$$\begin{pmatrix}\left\langle\boldsymbol{w}_{1},\mathscr{L}\boldsymbol{\psi}_{1}\right\rangle & \dots & \left\langle\boldsymbol{w}_{1},\mathscr{L}\boldsymbol{\psi}_{N}\right\rangle\\ \vdots & \ddots & \vdots\\ \left\langle\boldsymbol{w}_{N},\mathscr{L}\boldsymbol{\psi}_{1}\right\rangle & \dots & \left\langle\boldsymbol{w}_{N},\mathscr{L}\boldsymbol{\psi}_{N}\right\rangle\end{pmatrix}\begin{pmatrix}f_{1}\\ \vdots\\ f_{N}\end{pmatrix}=\begin{pmatrix}\left\langle\boldsymbol{w}_{1},\boldsymbol{g}\right\rangle\\ \vdots\\ \left\langle\boldsymbol{w}_{N},\boldsymbol{g}\right\rangle\end{pmatrix}, \quad (2.8)$$

and symbolically in algebraic form as

$$\mathbf{Lf}=\mathbf{g}. \quad (2.9)$$

In order to solve the system (2.9) properly, basis functions $\boldsymbol{\psi}_n(\boldsymbol{r})$ and testing functions $\boldsymbol{w}_m(\boldsymbol{r})$ have to generate linearly independent system of equations. Numerous efficient and reliable algorithms were implemented for solving linear systems in a form of (2.9), [6].

Following the Galerkin procedure [7], *i.e.* choosing the same set of linearly independent functions both for basis and testing functions, (2.4) is expressed element-wise as

$$\left\langle \boldsymbol{\psi}_m(\boldsymbol{r}), \left(1+\frac{1}{\kappa(\boldsymbol{r})}\right)\boldsymbol{\psi}_n(\boldsymbol{r})\right\rangle J_n - \left\langle \boldsymbol{\psi}_m(\boldsymbol{r}), \nabla\times\nabla\times\int_{V'}\boldsymbol{\psi}_n(\boldsymbol{r}')\,\mathrm{G}(\boldsymbol{r},\boldsymbol{r}')\,\mathrm{d}V'\right\rangle J_n = \left\langle \boldsymbol{\psi}_m(\boldsymbol{r}), \mathrm{i}\omega\varepsilon_0\boldsymbol{E}^{\mathrm{inc}}(\boldsymbol{r})\right\rangle \quad (2.10)$$

and symbolically reduced as

$$\mathbf{Zj} = \mathbf{e}. \quad (2.11)$$

Fast, efficient, and precise evaluation of impedance matrix $\mathbf{Z}$ is the key task of VMoM.

2.3 Basis Functions and Discretization

Considering particular choice of VEFIE-J, basis functions do not have to meet any specific restrictions. Different approaches of choosing basis and testing functions are discussed in [8], [9], and [10]. Majority of the possible basis functions is based on discretization of computational domain into a set of tetrahedral elements.

For the in-house VMoM code, developed at the department, piecewise constant basis and testing functions are used. Each tetrahedron contains three basis functions, which can be chosen as user-defined unit vectors or triplets $\boldsymbol{\psi}_n(\boldsymbol{r}) = \{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$. The choice of this type of basis functions allows for immediate utilisation of the advantages of $\nabla\times\nabla\times$ formulation. To do so, the impedance matrix $\mathbf{Z}$ is decomposed into

$$\mathbf{Z} = \mathbf{Z}_{\mathrm{mat}} + \mathbf{Z}_{\mathrm{vac}}, \quad (2.12)$$

where $\mathbf{Z}_{\mathrm{mat}}$ stands for the material part[1] and $\mathbf{Z}_{\mathrm{vac}}$ is being the vacuum part. The first matrix reads

$$Z_{\mathrm{mat},mn} = \int_{V_m-}\boldsymbol{\psi}_m(\boldsymbol{r})\left(\mathbb{I}+\frac{1}{\kappa(\boldsymbol{r})}\right)\cdot\boldsymbol{\psi}_n(\boldsymbol{r})\,\mathrm{d}V, \quad (2.13)$$

[1] Only the part with $\kappa(\boldsymbol{r})$ is actually material term, however it is convenient to merge this term with $\mathbb{I}$ part.

and the vacuum term is [4]

$$Z_{\mathrm{vac},mn} = \oint_{S_m-} \oint_{S_n-} \widehat{\boldsymbol{n}}_m(\boldsymbol{r}) \cdot (\boldsymbol{\psi}_m(\boldsymbol{r}) \times (\boldsymbol{\psi}_n(\boldsymbol{r}') \cdot \widehat{\boldsymbol{n}}_n(\boldsymbol{r}'))) \, \mathrm{G}(\boldsymbol{r}, \boldsymbol{r}') \, \mathrm{d}S' \, \mathrm{d}S. \tag{2.14}$$

Since the basis functions are chosen as the Cartesian canonical triplets, the vacuum term significantly reduces. The computationally favourable form reads

$$Z_{\mathrm{vac},mn} = \oint_{S_m-} \oint_{S_n-} \boldsymbol{\psi}_m(\boldsymbol{r}) \cdot \widehat{\boldsymbol{n}}_n(\boldsymbol{r}') \, \boldsymbol{\psi}_n(\boldsymbol{r}') \cdot \widehat{\boldsymbol{n}}_m(\boldsymbol{r}) \, \mathrm{G}(\boldsymbol{r}, \boldsymbol{r}') \, \mathrm{d}S' \, \mathrm{d}S. \tag{2.15}$$

Accuracy of the results and speed of the evaluation of Z_{mn} terms highly depend on the number of discretization elements and the order of used numerical method.

2.4 Terms Evaluation

The material and vacuum terms of VMoM are evaluated separately via functions `computeZmat()` and `computeZvac()`, respectively.

Particularly, the vacuum terms are evaluated as

$$Z_{\mathrm{vac},mn} = \sum_{j=1}^{4} \sum_{k=1}^{4} \widehat{\boldsymbol{n}}_m^j \cdot \left(\boldsymbol{\psi}_m \times \left(\boldsymbol{\psi}_n \times \widehat{\boldsymbol{n}}_n^k \right) \right) \int_{\Delta_m^j} \int_{\Delta_n^j} \mathrm{G}(\boldsymbol{r}, \boldsymbol{r}') \, \mathrm{d}S' \mathrm{d}S, \tag{2.16}$$

where Δ_μ^ν is surface area of ν-th face of μ-th tetrahedron. A problem arises for diagonal terms, *i.e.*, $\|\boldsymbol{r} - \boldsymbol{r}'\| \to 0$. Various techniques of dealing with the singularities in a form of Green's function were described in [11]. Singularity extraction approach

$$\mathrm{G}(\boldsymbol{r}, \boldsymbol{r}') = \frac{\mathrm{e}^{-\mathrm{i}k_0\|\boldsymbol{r}-\boldsymbol{r}'\|} - 1}{4\pi \|\boldsymbol{r} - \boldsymbol{r}'\|} + \frac{1}{4\pi \|\boldsymbol{r} - \boldsymbol{r}'\|} \tag{2.17}$$

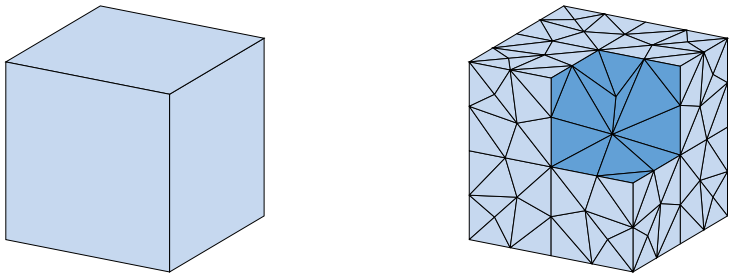
is followed in this work. Integration of both parts of (2.17) is described in detail in [12]. The remaining terms of matrix $\mathbf{Z}_{\mathrm{vac}}$ are evaluated numerically applying Gauss-Legendre quadrature rule in barycentric coordinates [13]. The elements of $\mathbf{Z}_{\mathrm{mat}}$ are evaluated analytically as products of scalar material function with volume of particular tetrahedron.

The core of VMoM was described above. Though the complete theory covers wider area of interest, it is not purpose of this work to introduce it whole.

Chapter 3

Tetrahedral Discretization

Delaunay triangulation is described in this chapter together with its extension for tetrahedral discretization. Two workflows for tetrahedral discretization of general three-dimensional objects are introduced at the end of the chapter.



(a): **(b):**

Figure 3.1: An example of tetrahedral discertization. (a) original object, (b) discretized object.

3.1 Delaunay Triangulation

Delaunay triangulation is the most common algorithm for discretization of continuous volume. Basic principle is to divide n-dimensional object into simplices[1] of this space. The principle behind this process is explained on a two dimensional object, where the simplices are triangles.

Consider a set of N points $S \in \mathbb{R}^2$. The Voronoi region of $\boldsymbol{p}_n \in S$ is defined as [14]

$$V_{\boldsymbol{p}_n} = \left\{ \boldsymbol{x} \in \mathbb{R}^2 | \, \|\boldsymbol{x} - \boldsymbol{p}_n\| \leq \|\boldsymbol{x} - \boldsymbol{q}\| \, , \forall \boldsymbol{q} \in S \right\}. \tag{3.1}$$

Each Voronoi region is a convex polygon bounded with at most $N - 1$ edges or unbounded. The Voronoi diagram consists of Voronoi regions for each point of S together with their shared edges and vertices, see Figure 3.2.

[1] A simplex of n-dimensional space is a convex hull of $n + 1$ vertices of this space

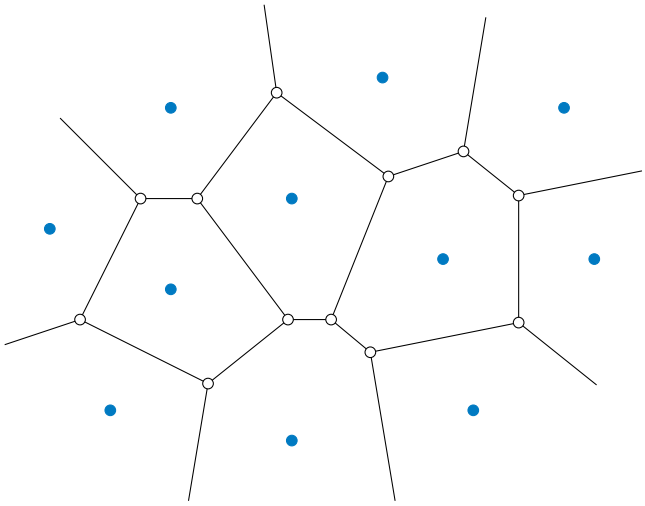


Figure 3.2: Voronoi diagram of a set of points $S \subset \mathbb{R}^2$. Blue dots represent points of S and white dots are Voronoi points.

Each Voronoi point lies in the centroid of a circle circumscribed to triangle $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ and connects three Voronoi regions $V_{\boldsymbol{p}_n}$. The edge between points $\boldsymbol{p}_j$ a $\boldsymbol{p}_k$ is called Delaunay edge if it intersects common edge of Voronoi regions of these two points. Convex hull of S is decomposed with Delaunay edges into triangular regions. This decomposition is dual to Voronoi diagram and is referred to as Delaunay triangulation [14], see Figure 3.3.

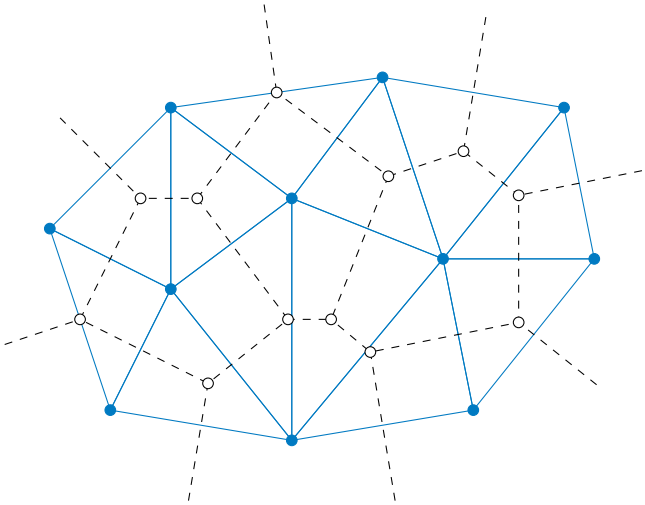


Figure 3.3: Delaunay triangulation presented as a dual to the Voronoi diagram. Dashed lines represent former Voronoi diagram while solid lines represent triangulation.

Among all possible triangulations, Delaunay triangulation maximizes minimal angle [14].

Let $\boldsymbol{p}_2\boldsymbol{p}_3$ belongs to triangles $\boldsymbol{p}_1\boldsymbol{p}_2\boldsymbol{p}_3$ and $\boldsymbol{p}_2\boldsymbol{p}_3\boldsymbol{p}_4$ which form a convex quadrangle. Edge-flip is an operation which swaps edge $\boldsymbol{p}_2\boldsymbol{p}_3$ for $\boldsymbol{p}_1\boldsymbol{p}_4$, see Figure 3.4 for illustration.

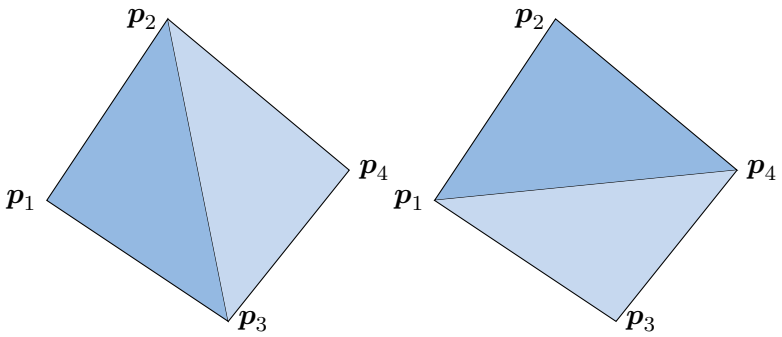


Figure 3.4: The left panel shows the former triangulation of four points $\boldsymbol{p}_1$, $\boldsymbol{p}_2$, $\boldsymbol{p}_3$ and $\boldsymbol{p}_4$. The right panel shows new triangulation after swapping the common edge.

Fast algorithm for constructing Delaunay triangulation is based on interleaving edge-flipping algorithm with addition of points. Points $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, and $\boldsymbol{q}_3$ spanning the convex hull of S are added into S. These points form initial triangulation of S noted as D_0. In each iteration of for-loop one point $\boldsymbol{p}_j$ of S is added, which divides the original triangle $\tau_k \in D_{j-1}$ containing $\boldsymbol{p}_j$ into three triangles, see Figure 3.5. New Delaunay triangulation D_j of $\boldsymbol{q}_1$, $\boldsymbol{q}_2$, $\boldsymbol{q}_3$, $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_j$ is created with edge-flipping.

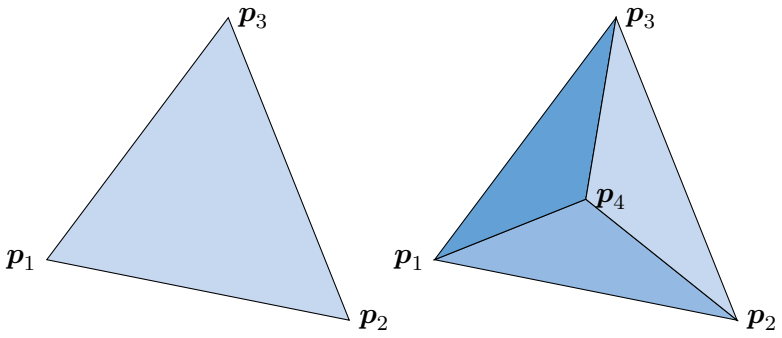
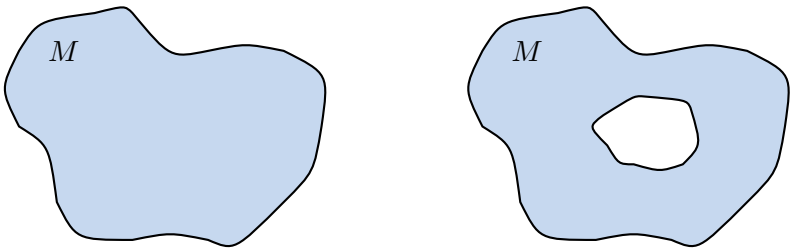


Figure 3.5: The left panel shows former triangulation of four points a,b,c and d. The right panel shows new triangulation after addition of a point.

Delaunay triangulation is able to solve discretization problem in situation in which set $M \subset \mathbb{R}^2$ representing bounded surface is simply connected. In opposite case, constrained Delaunay triangulation, which takes a set of points $\boldsymbol{p}_n \in S$ together with a set of line segments L connecting these points, have to be used. Both, the Voronoi diagram concept and the edge-flipping algorithm remain valid in this case with only minor changes [14].

A common meshing problem is a discretization of a set $M \subset \mathbb{R}^2$ which is not simply connected, see Figure 3.6. This surface object is fully represented by its boundary ∂M. Inner points of M are no longer parts of the input. Instead, boundary ∂M decomposed into line segments serves as an input. Resulting triangulation suffers from bad quality of triangles and is unacceptable for numeric methods. New points of M are thus added to improve resulting mesh

with Delaunay refinement algorithm [14]. Constrained Delaunay triangulation itself does not address slivers and holes carved into the object. For proper triangulation of such structures additional algorithms must be used [15].



(a) : Simply connected set. **(b) :** Not simply connected set.

Figure 3.6: Example of simply connected and not simply connected set.

A Delaunay tetrahedralization stands on the same foundation as previously described Delaunay triangulation. However, it is considerably harder problem to solve. In [16] it is proven that Delaunay tetrahedralization is a NP-complete decision problem. Numerous algorithms for Delaunay tetrahedralization have already been implemented. Following section describe two which will be used in this thesis.

3.2 Tetrahedral Mesh Generation

Considering knowledge gained from the previous work [17], following statements can be given. Triangulation obtained from MATLAB functions [18] significantly deforms the original object, it is thus not sufficient for purposes of MOOPEL project. Programs *gmsh* [19] and *TetGen* [20] seem to be the best option for obtaining discretization. Both of them have their pros and cons, therefore, it is not possible to determine conclusively better option. An output of discretization process contains information about vertices of tetrahedrons and their connectivity list. The following code snippet demonstrates how to load data from file and to obtain `mesh` structure.

```
1 [Points, Edges, Triangles, ConnectivityList,...
2     fileIsReadable] = mesh.importNastran('filename.stl');
3 Mesh = prepareMesh(Points, ConnectivityList, Regions);
```

Figure 3.7: Generation of `mesh` structure.

NASTRAN `.nas` format [21] is chosen for representing the data in files. The `mesh` structure serves as MATLAB data structure describing geometry of discretized object, see Table 3.1.

Variable	Description	Format
`Points`	point location	(x, y, z)
`ConnectivityList`	tet. point ownership	(p_1, p_2, p_3, p_4)
`TriPoints`	tri. point ownership	(p_1, p_2, p_3)
`TriCenter`	tri. center	(x, y, z)
`TriNormal`	tri. normal vector	(n_x, n_y, n_z)
`TriArea`	triangle area	A
`TetCenter`	tet. center	(x, y, z)
`TetVolume`	tetrahedron volume	V
`TetTriangles`	tet. tri ownership	(t_1, t_2, t_3, t_4)
`TetSigns`	relative signs of tri. norm.	(s_1, s_2, s_3, s_4)
`TetRegion`	region assignment of tet.	r
`nTetra`	number of tetrahedra	N_{tets}
`nTria`	number of triangles	N_{tris}
`nRegions`	number of regions	N_{reg}

Table 3.1: Description of `mesh` structure (tri. stands for triangles, tet. stands for tetrahedrons).

Three-dimensional object modelling in MATLAB is highly restricted [18]. Set of MATLAB high-level graphic functions allows good visualisation and manipulation with object. However, the tools specialised for modelling are absent. It is technically viable to develop modeller similar to one in *AToM* [22]. But implementation and subsequent optimisation would need unacceptable amount of time and human resources, which could be used more effectively on more important tasks. It is necessary to choose an alternative most possible a third party package. Two feasible ways of modelling are described in the following subsections, based on different tetrahedral mesh generators.

3.2.1 TetGen

The *TetGen* package generates tetrahedral mesh from given (discretized) triangular boundary. Though it has its own input file format, the `.stl` [23] file can be used instead. *TetGen* has no internal computer aided graphics (CAD) editor and relies on external software, which generates input with desired file format.

CAD programs offer wide range of tools for modelling arbitrarily shaped 3D objects and lot of them offer excellent technical support and online community. User has thus freedom to choose software familiar to him or her and export data to `.stl`. The function `importNastran()` used for data loading have to be modified due to different input file formats [20],[17]. Serious problem arises in situation of multilayered media, where *TetGen* does not distinguish objects with different properties.

3.2.2 gmsh

The *gmsh* package supports both 3D object drawing and following discretization into tetrahedrons. Object modelling can be done in available graphic user interface or with commands in *gmsh*'s scripting language. Resulting mesh is then saved in `.bdf` file, which is compatible with `importNastran()` function without any changes.

However, modelling is restricted with narrow range of tools against commercial CAD programs. The graphic user interface at itself is not too much user-friendly. Severe drawback is absence of keyboard shortcuts – every wrong move have to be complicatedly repaired.

Both of the discussed programs are distributed under the terms of public licences. The execution time of both is almost negligible. Correctly implemented Delaunay tetrahedralization is also a possible option. Nevertheless, this algorithm is demanding for computational resources and it can be assumed that it would take probably more than 80 % of whole running time of VMoM program.

Chapter 4

Post-Processing of Results

Various methods for visualisation of VMoM results and their export are described in this chapter. Section 4.2 serves as documentation for implemented functions. An algorithm for direct visibility of triangles is described latter in this chapter. This algorithm is essential for correct vector representation of three-dimensional vector fields.

4.1 Used Principles

Two common principles are used for geometric manipulation with mesh. The first one serves for view angle manipulation and the second one applies for mesh filtration. These principles are used with slight changes throughout the codes.

The different camera settings (MATLAB vs. LaTeX/Ti*k*Z) imply complicated parametrization of a scene depending on a user-defined view angle. In this manner an object is static and camera is rotated. However, it is convenient to rotate the object and let camera being static instead. This is done with centering of the object and then use of two rotation matrices

$$R_z = \begin{pmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix},$$

where φ stands for azimuth and θ for elevation. This particular task is done in MATLAB with the following code.

```
1 phi = -azim*pi/180; tht = elev*pi/180;
2 Rz = [cos(phi) sin(phi) 0; -sin(phi) cos(phi) 0; 0 0 1];
3 Rx = [1 0 0; 0 cos(tht) sin(tht); 0 -sin(tht) cos(tht)];
4 Mesh.Points = Mesh.Points*Rz*Rx;
```

Figure 4.1: Rotation matrices and their utilisation for rotation of mesh grid.

The matrices are applied with respect to horizontal vector representation. It is able to rotate object back to the original setup with the use of inverse matrices.

The second used method is a triangle filtration. All vertices lying above a user-defined plane are removed in the first step. Thereafter, triangles having two vertices below the plane are marked as the ones intersecting the plane. The entire process is performed as shown in the code snippet 4.2 and schematically depicted in figures 4.3 and 4.4 as well.

```
% plNrm contains plane coeffients, vert = Mesh.Points
object_cut = (vert*plNrm(1:3)' >= plNorm(4)).*(1:size(vert,1))';
faces_c = faces(sum(ismember(faces(1:end,:),object_cut),2) <= 2 ...
    & sum(ismember(faces(1:end,:),object_cut),2) >= 1,:);
```

Figure 4.2: Triangles filtration code.

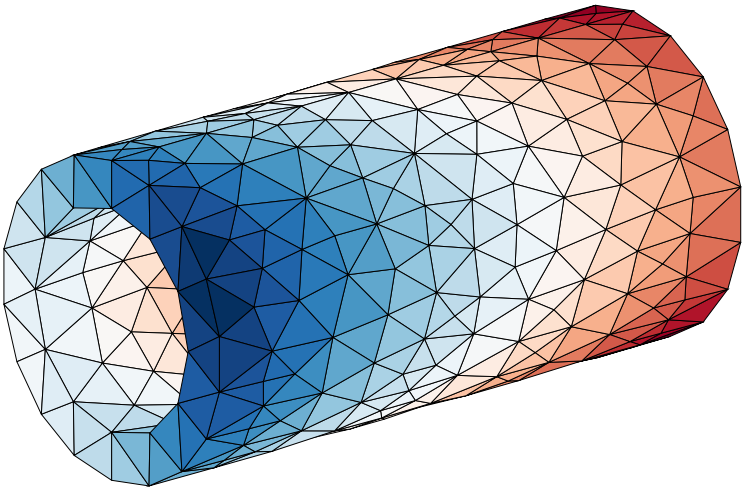


Figure 4.3: Filtration demonstrated on a hollow cylinder.

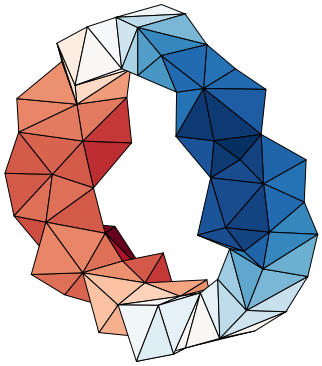


Figure 4.4: A cut along $x = 0$ plane taken from the object depicted in Figure 4.3. The x-coordinate points along the cylinder.

Exactly the same principle can be utilized for tetrahedral filtration. Notice that the method is not restricted to cutting planes only. Other shapes such as cylinders or spheres can be used as well.

4.2 Visualisation

Visualisation of results is an important feature of every computer simulator software. In contrast to surface integral formulation, the VEFIE-J method requires possibility to depict inner field too. Since the current density is complex vector function, both the surface and arrow plots are needed. Two functions for visualisation and one data export function are implemented, incorporating the same features as the functions already existing in the *AToM* package for surface objects.

4.2.1 Function plotCurrent()

Functionality is similar to the function `plotCurrent()` already implemented in *AToM*. It is possible to plot current on the boundary of the volumetric object (implicit) or to specify a cutting surface. The mandatory arguments are `mesh` and basisFunctions structures together with vector of coefficients J_n of (2.10). The list of the optional parameters is depicted in Table 4.1.

Parameter	Description
'`PlaneNormal`'	Specification of a visualised surface. A slicing plane is given by coefficients a-d in $[a\ b\ c\ d]$ matrix, $ax + by + cz + d$. Boundary of object is set as default.
'`Relation`'	Specifies the relation operator in previous parameter. The choices are '=', '>', and '<'. By default, operator '=' is utilised.
'`Part`'	Determines which part of the complex vector function is depicted. The options are: '`real`' (default), '`imag`', and '`abs`'.
'`Handles`'	If utilised, the existing figure and axes objects are reset with actual data (instead of creating new ones).
'`TikZSave`'	Allows direct conversion of a generated figure via Ti*k*Z macros shown below. Possible options are '`no`' (default), '`yes`' and '`forced`'. See documentation of the `patches2TikZ()` function for further information.

Table 4.1: Optional parameters of the `plotCurrent()` function.

A regular call of the function `plotCurrent()` is done with following code.

```
1 J = Z\E; % Z is impedance matrix and E is excitation
2 hndl = results.plotCurrent(Mesh,BasisFun,J);
```

Figure 4.5: `plotCurrent()` syntax example.

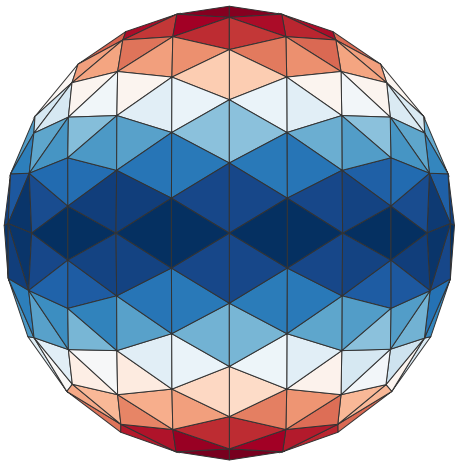


Figure 4.6: `plotCurrent()` output example.

4.2.2 Interpolation of the complex vector function

The function `plotSlice()` was originally implemented for Schaubert-Wilton-Glisson basis functions [24]. Resulting vector function (*i.e.*, current density, electrical field intensity, ...) is plotted on plane discretized into squares. The object is sliced with user-defined plane. The latter is optimised and discretized into square patches, which are assigned to particular tetrahedron, see Figure 4.7. The details were already described in [17].

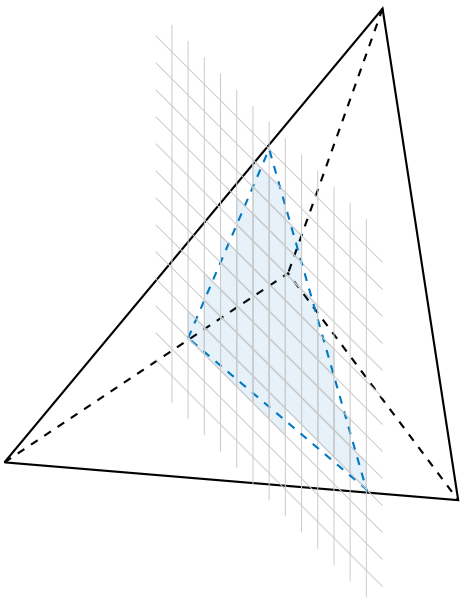


Figure 4.7: To the interpolation process of the `plotSlice()` function.

The mandatory arguments are `mesh` and basisFunctions structures together with vector of coefficients J_n of (2.10). Table below introduces optional arguments of `plotSlice()`.

Parameter	Description
'PatchesCeiling'	The maximum number of squares in one row.
'PlaneNormal'	A specification of the slicing plane. Multiple choices are possible. The coefficient matrix $[a\ b\ c\ d]$ sets plane $a\,x + b\,y + c\,z = d$. If this matrix has only three columns $[a\ b\ c]$, plane $a\,x + b\,y + c\,z = 0$ is chosen instead. Options 'x', 'y', and 'z' sets $x = 0$, $y = 0$, and $z = 0$ (default), respectively.
'Part'	Determines which part of the complex vector function is depicted. The options are: 'real' (default), 'imag', and 'abs'.
'Handles'	If utilised, the existing figure and axes objects are reset with actual data (instead of creating new ones).
'TikZSave'	Allows direct conversion of a generated figure via TikZ macros shown below. Possible options are 'no' (default), 'yes' and 'forced'. See documentation of the patches2TikZ() function for further information.

Table 4.2: plotSlice() optional arguments.

The following code shows how to call the function plotSlice().

```
J = Z\E; % Z is impedance matrix and E is excitation
hndl = results.plotSlice(Mesh,BasisFun,J);
```

Figure 4.8: plotSlice() syntax example.

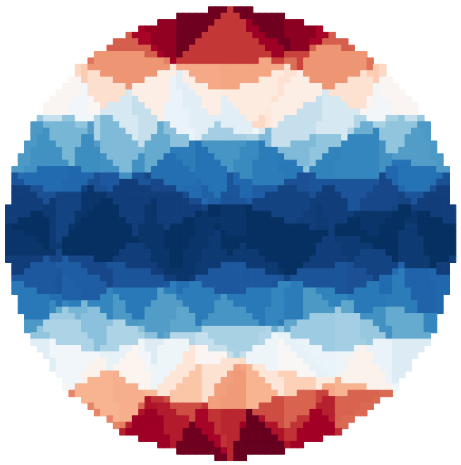


Figure 4.9: plotSlice() example.

4.2.3 Conversion of the Matlab current density plot into TikZ

The function `patches2TikZ()` converts the MATLAB current density plot to a Ti*k*Z macro which is ready to be compiled with LaTeX. In contrast to a direct MATLAB `.pdf` export, this function has lower memory requirements, it is fully scalable, and its output reaches higher visual qualities. A huge advantage is that all the relevant graphical objects are converted from MATLAB figure to LaTeX/Ti*k*Z commands, which are editable by the user. The references of the `figure`, `axes`, and `patch` objects are mandatory arguments. The function offers some advanced features, see Table 4.3 for more details.

Parameter	Description
'FileName'	Specification of an output file name. By default, 'resultsPlot' is used.
'Output'	In the case of large number of graphical primitives (more than four thousand), function does not produce any output. If option 'forced' is set, output will be produced irrespective of the number of graphical primitives.
'Faces'	A connectivity list of the patches.
'Vert'	A list of vertices of the patches.
'ColourVec'	Color specification of patches.
'ColourMap'	Specification of the color map.
'Azimuth'	Specification of the azimuth angle.
'Elevation'	Specification of the elevation angle.

Table 4.3: Optional parameters of the function `patches2TikZ()`.

All previous pictures were done with this functions. See Appendix A for other examples. The function is called with code in Figure 4.10.

```
1 hndl = results.plotCurrent(Mesh,BasisFun,J);
2 patches2TikZ(hndl);
```

Figure 4.10: `patches2TikZ()` syntax example.

4.3 Visibility of Triangles

The *AToM* package has a utility, which converts an output of the function `plotCurrent()` into Ti*k*Z macros. However, data for triangles have to be adjusted before conversion so that the triangles closer to the observation point (camera) are the last ones and the triangles placed furthermost are the first ones. Otherwise, LaTeX interpretation of an unsorted data set causes serious problems with the perspective. Need of data reduction exists for

larger mesh grids. A method outlined in this section effectively reduces the number of plotted triangles only to those truly being visible from a given view observation point. The following script is an example of use of conversion process.

```
hndl = results.plotCurrent(Mesh, 'basisFcns', BF,...
    'Ivec', I(:, end),'part', 'abs', 'scale',...
    'linear', 'arrowScale', 'proportional');

[data, settings] = prepareDataForTikz(hndl);
[data] = reduceTriangle(data,settings);
processSnapshot(cd,'file_name', data, settings);
```

Figure 4.11: Example of conversion snippet.

The function `processSnapshot()` produces a complete `.tex` file. The use of `reduceTriangle()` shortens compilation time and reduces memory size of `.pdf` output. Amount of memory size reduction and compilation time is dependent on particular mesh structure.

The following animation shows different triangles visibility dependent on view angle.

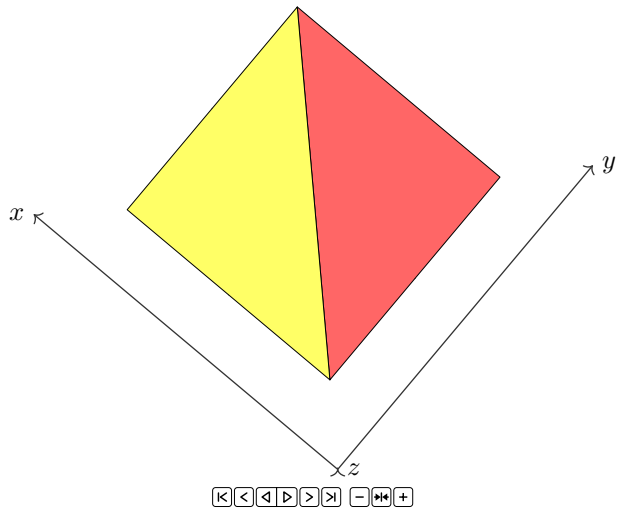


Figure 4.12: Different viewing angles of three triangles (to run the animation, open the PDF file in Adobe Acrobat Reader).

The algorithm can be divided into few steps.

1. Data are converted back into `connectivityList` and `points` matrices.
2. Structure is rotated (see Section 4.1) and translated so the required viewing angle lies in x-y plane.

3. Triangles are then sorted with respect to distance from z component of triangle centroid to x-y plane.

4. The core of the algorithm, see Figure 4.13, is then repeated for each triangle.

```
faces_v(1,:) = faces_b(1,:); k = 1; p =[1 4 2 5 1 4; 2 5 3 6 3 6];
for j = 2:faces_n
    tri = triangulation(faces_v(1:k,:),vert(:,1),vert(:,2));
    nv1 = (0.1*cvert(j,1:2) + 0.9*vert(faces_b(j,:),1:2));
    nv2 = (0.5*cvert(j,1:2) + 0.5*vert(faces_b(j,:),1:2));
    nv3 = reshape(mean([nv1(p);nv2(p)]),[2 3])';
    nvert = [nv1; nv2; nv3; cvert(j,1:2)];
    ivert = tri.pointLocation(nvert);
    if ( sum(isnan(ivert)) > 0 )
        k = k + 1; faces_v(k,:) = faces_b(j,:);
    end
end
faces_v = faces_v(k:-1:1,:);
```

Figure 4.13: Direct visibility algorithm.

Determination of the direct triangle visibility is demonstrated in a series of figures 4.14-4.17. If the camera lies below the structure (left outermost picture in animation), only the yellow and the red triangles are visible. The closest triangle is set before start of `for` loop.

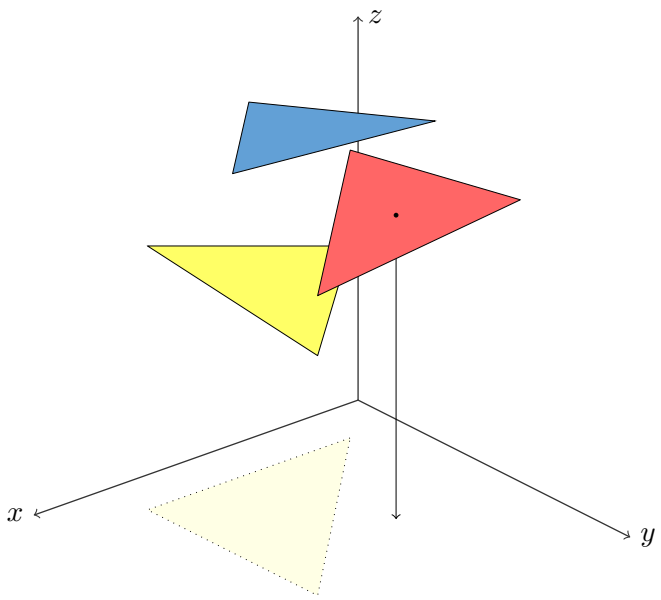


Figure 4.14: The first iteration of algorithm determining direct visibility of triangles.

The yellow triangle is projected onto x-y plane. The algorithm evaluates

if any inner point from the red triangle projected into x-y plane is outside of area bounded with already projected triangles. In outlined case, the red triangle is evaluated as visible.

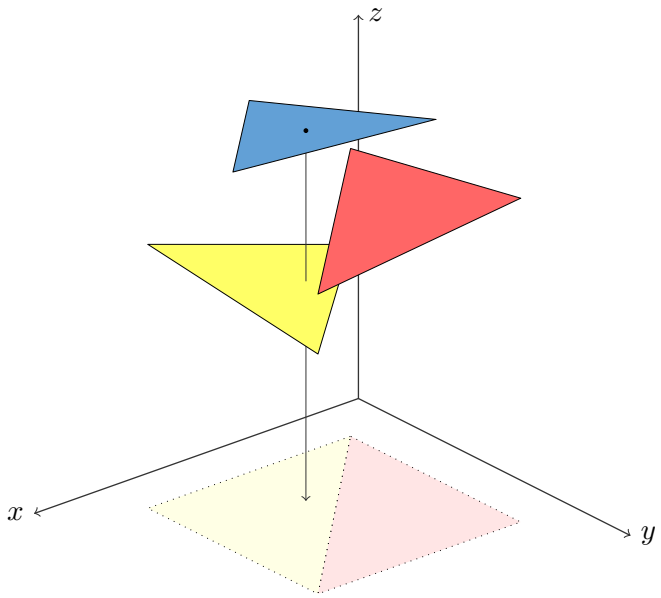


Figure 4.15: The second iteration of algorithm determining direct visibility of triangles.

The same process is repeated for the blue triangle. Every inner point of the blue triangle falls into area already bounded with previous triangles. The blue triangle is evaluated as not visible. In different scenario, where the camera position is above structure (right outermost position in animation), all triangles are visible. This situation is evaluated as follows.

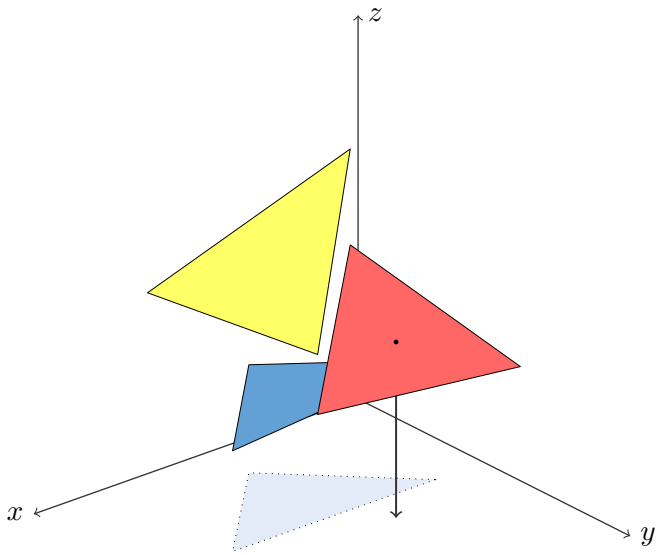


Figure 4.16: The first iteration of algorithm determining direct visibility of triangles in different situation.

Now the triangle at the first position is the blue one. Some inner points of the red triangle are projected into area bounded by projected the blue triangle however there are also points, which falls outside of this area. The red triangle is evaluated as visible one.

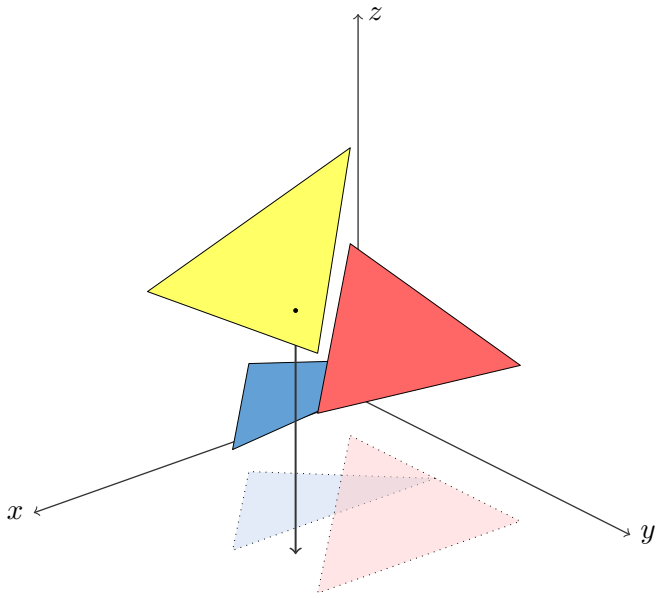


Figure 4.17: The second iteration of algorithm determining direct visibility of triangles in different situation.

The same situation occurs for the yellow triangle, which is also evaluated as visible.

The following plot shows time dependence of the outlined algorithm.

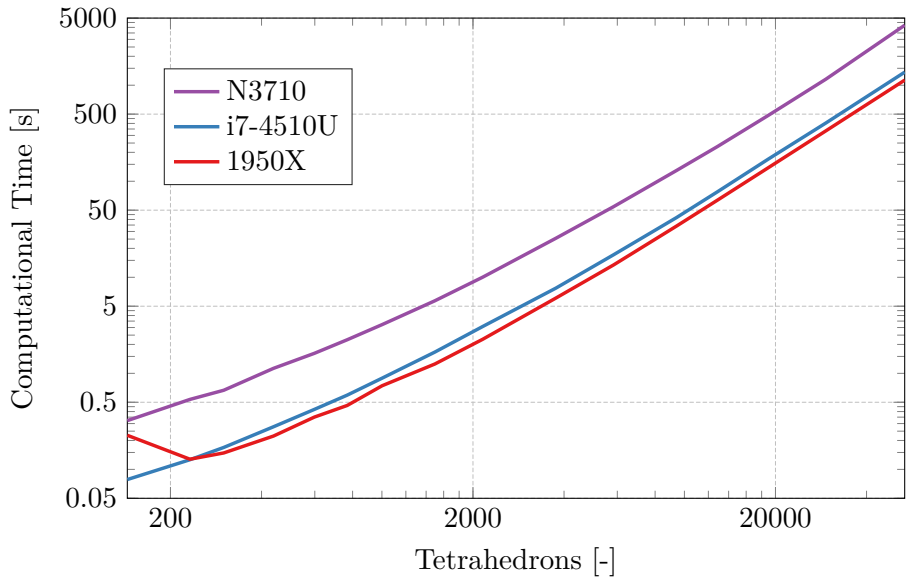
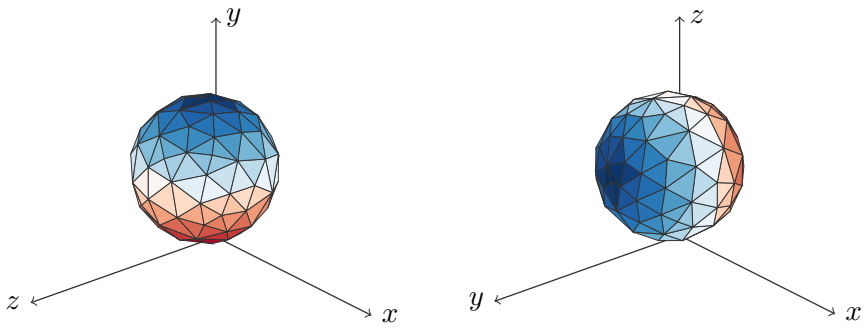


Figure 4.18: Time complexity of `reduceTriangle()` function. Algorithmic complexity was estimated as $O(n^2)$. Three different computer processing units were used.

MATLAB and Ti*k*Z do not use the same coordinate system, see Figure 4.19, which has to be respected in a final rendering.



(a) : MATLAB default coordinates. **(b) :** Ti*k*Z default coordinates.

Figure 4.19: MATLAB and Ti*k*Z coordinate systems compared to each other.

This is particularly done with inserting `\tdplotsetmaincoords{90}{0}` command on the beginning of `.tex` file.

The implemented method has a problem in situations in which is not clear which triangles are closer to x-y plane. In Figure 4.20 is showed this phenomenon.

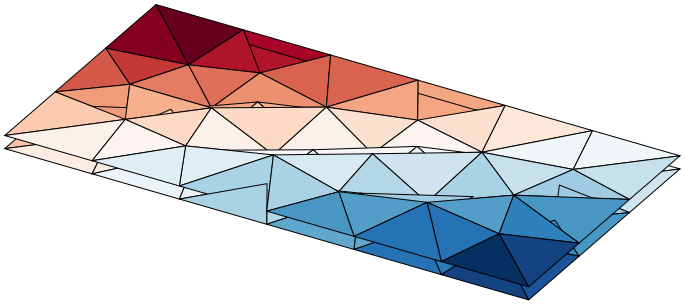


Figure 4.20: Example of problematic mesh.

However, a mesh refinement (if possible) cures this issue effectively. More examples are depicted in Appendix A.

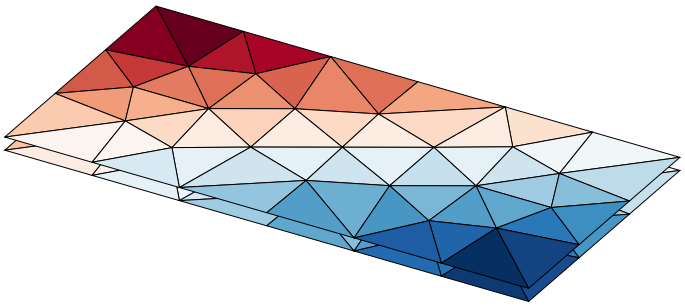


Figure 4.21: Example of correct output.

Chapter 5

Conclusion

Elementary theory of electromagnetic field integral equations solving by Method of Moments was discussed in this thesis. Volumetric Electric Field Integral Equation solved for current density (VEFIE-J) is chosen for its little demands on basis functions and straightforward evaluation. Two software for obtaining tetrahedral discretization were suggested. The *gmsh* package offers both modelling and tetrahedral mesh generation, however its modelling possibilities are restricted by short range of tools. *TetGen* suite has only tetrahedral mesh generator, although it is possible to use common modelling software such as *AutoCAD*, *progeCAD* etc. After eight months of user experience with both, *gmsh* seems to be the better option.

The fourth chapter was devoted to manipulation and visualisation of volumetric method of moments results. The function `plotCurrent()` visualises current density on boundary triangles. It has to be slightly rewritten for vectorization, which would accelerate this method. The current density direction arrows will be added in next version. The second function `plotSlice()`, which plots field on particular plane, has reached excellent results. The execution time of the function in case of three hundred thousand of tetrahedral elements was about twenty seconds. As was said, `plotSlice()` was formerly designed for different type of basis functions. Piece-wise constant basis functions type does not use full potential of this method, however it is convenient for evaluation. The last implemented function `patches2TikZ()` returns correct results in most cases. However, there are still inputs which are not treated and correctly debugged. The algorithm for triangle direct visibility, which is also used in `patches2TikZ()`, was described.

The mandatory argument `BasisFunctions` in functions `plotCurrent()` and `plotSlice()` is not used. However, it is added for backwards compatibility, if the basis functions will be chosen differently in future. The next work will focus on implementation of another plotting functions (*i.e.* charge distribution visualisation, near field and far field plottings, ...) and on debugging of the current functions.

Bibliography

[1] R. Garg, *Analytical and Computational Methods in Electromagnetics*. Artech House, 2008.

[2] M. N. O. Sadiku, *Numerical Techniques in Electromagnetics with MATLAB*. CRC Press, 2009.

[3] A. Zangwill, *Modern Electrodynamics*. Cambridge University Press, 2013.

[4] A. G. Polimeridis, J. F. Villena, L. Daniel, and J. K. White, "Stable FFT-JVIE Solvers for Fast Analysis of Highly Inhomogeneous Dielectric Objects," *Journal of Computational Physics*, vol. 269, pp. 280–296, 2014.

[5] M. M. Botha, "Solving the Volume Integral Equations of Electromagnetic Scattering," *Journal of Computational Physics*, vol. 218, no. 1, pp. 141–158, 2006.

[6] G. H. Golub and C. F. V. Loan, *Matrix Computations*, fourth Edition ed. The Johns Hopkins University Press, 2013.

[7] R. F. Harrington, *Field Computation by Moment Methods*. IEEE Press, 1993.

[8] S. A. Carvalho and L. S. Mendes, "Scattering of EM Waves by Inhomogeneous Dielectrics with the Use of the Method of Moments and 3-D Solenoidal Basis Functions," *Microwave and Optical Technology Letters*, vol. 12, no. 6, pp. 327–331, 1999.

[9] J. Markkanen and P. Ylä-Oijala, "Discretization of Electric Current Volume Integral Equation with Piecewise Linear Basis Functions," *IEEE Transactions on Antennas and Propagation*, vol. 62, no. 9, pp. 4877–4880, 2014.

[10] A. G. Polimeridis, M. T. H. Reid, S. G. Johnson, J. K. White, and A. W. Rodriguez, "On the Computation of Power in Volume Integral Equation Formulations," *IEEE Transactions on Antennas and Propagation*, vol. 63, no. 2, pp. 611–620, Feb. 2015.

[11] J. Tuček, *Reaction Integrals in Volumetric Method of Moments and Their Evaluation.* Czech Technical University, 2019.

[12] R. Graglia, "Static and Dynamic Potential Integrals for Linearly Varying Source Distributions in Two- and Three-dimensional Problems," *IEEE Transactions on Antennas and Propagation*, vol. 35, no. 6, pp. 662–669, Jun. 1987.

[13] A. D. Dunavant, "High Degree Efficient Symmetrical Gaussian Quadrature Rules for the Triangle," *International Journal for Numerical Methods in Engineering*, vol. 21, no. 6, pp. 1129–1148, 1985.

[14] H. Edelsbrunner, *Geometry and Topology for Mesh Generation.* Cambridge University Press, 2001.

[15] P. L. George, F. Hecht, and E. Salte, "Automatic Mesh Generator with Specified Boundary," *Computer Methods in Applied Mechanics and Engineering*, vol. 92, no. 3, pp. 269–288, 1991.

[16] J. Ruppert and R. Seidel, "On the Difficulty of Triangulating Three-Dimensional Nonconvex Polyhedra," *Discrete & Computational Geometry*, vol. 7, pp. 227–253, Dec. 1992.

[17] V. Neuman, *Tetrahedral Discretization and Its Visualisation for Volumetric Method of Moments.* Czech Technical University, 2019.

[18] MathWorks. (2019, May) MATLAB. The MathWorks, Inc. [Online]. Available: https://www.mathworks.com/

[19] C. Geuzaine and J.-F. Remacle, "Gmsh: A Three-Dimensional Finite Element Mesh Generator with Built-In Pre- and Post-Processing Facilities," *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[20] H. Si, "Tetgen, a Delaunay-Based Quality Tetrahedral Mesh Generator," *ACM Transactions on Mathematical Software*, vol. 41, no. 2, pp. 1–36, Feb. 2015.

[21] *Autodesk® Nastran® User's Manual*, Autodesk, Inc., 2018. [Online]. Available: https://knowledge.autodesk.com/

[22] (2019, May) Antenna Toolbox for MATLAB. [Online]. Available: http://www.antennatoolbox.com/

[23] D. Chakravorty. (2019, May) STL File Format (3D Printing) – Simply Explained. [Online]. Available: https://all3dp.com/what-is-stl-file-format-extension-3d-printing/

[24] D. Schaubert, D. Wilton, and A. Glisson, "A Tetrahedral Modeling Method for Electromagnetic Scattering by Arbitrarily Shaped Inhomogeneous Dielectric Bodies," *IEEE Transactions on Antennas and Propagation*, vol. 32, no. 1, pp. 77–85, Jan. 1984.

Appendix A
Results Gallery

Results of implemented functions with different argument settings are shown in this appendix. All pictures were obtained through function `patches2TikZ()`. First two pictures are examples of direct visibility algorithm output for complex objects. Latter pictures serves as examples of output with different settings.

A.1 Direct Visibility Results

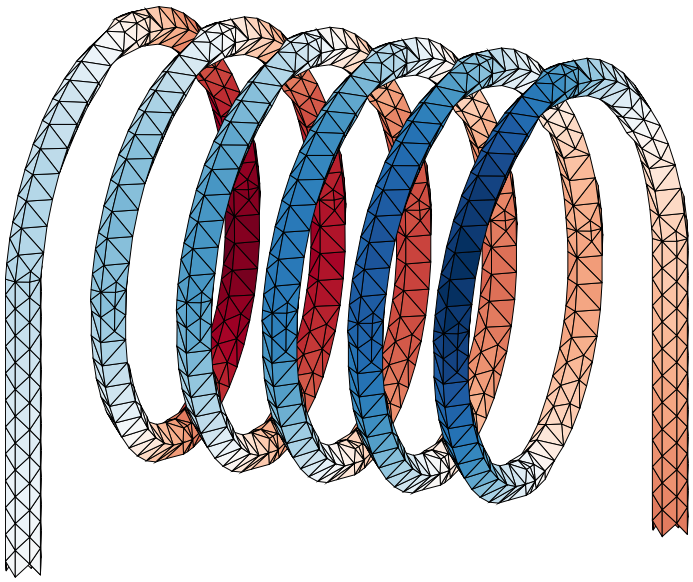


Figure A.1: Example of visualisation, solenoid object taken from *gmsh* demos.

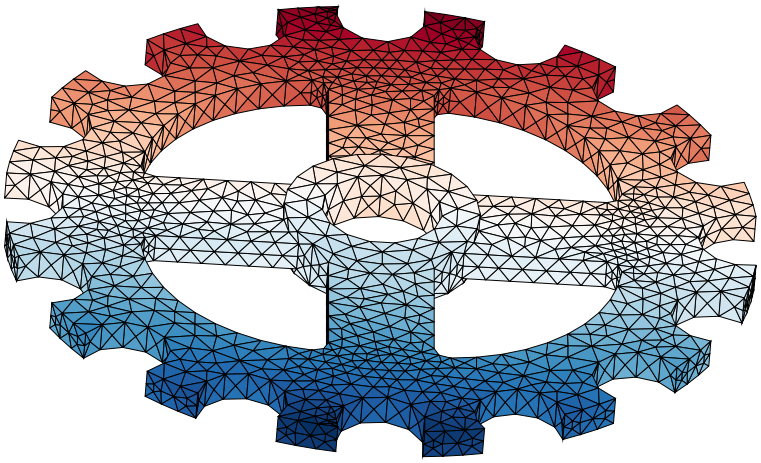


Figure A.2: Example of direct visibility algorithm result.

A.2 Gearbox Example

Codes cited below result in Figure A.3 and Figure A.4 respectively.

```
hndl = plotCurrent(Mesh,BF,J,'Part','Real',...
    'PlaneNormal',[0 -1 0 0],'Relation','>');
```

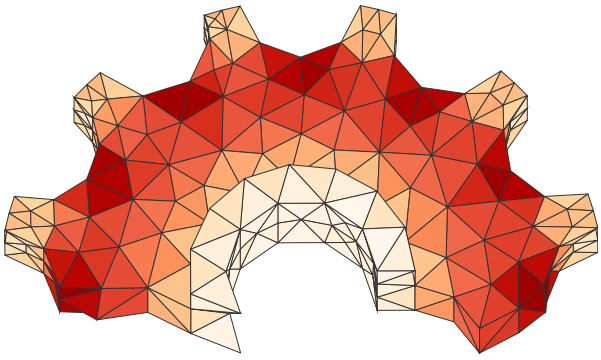


Figure A.3: Example of an output from function `plotCurrent()`.

```
hndl = plotSlice(Mesh,BF,J,'Part','Real',...
    'PlaneNormal',[0 0 1 0.1],'PatchesCeiling',72);
```

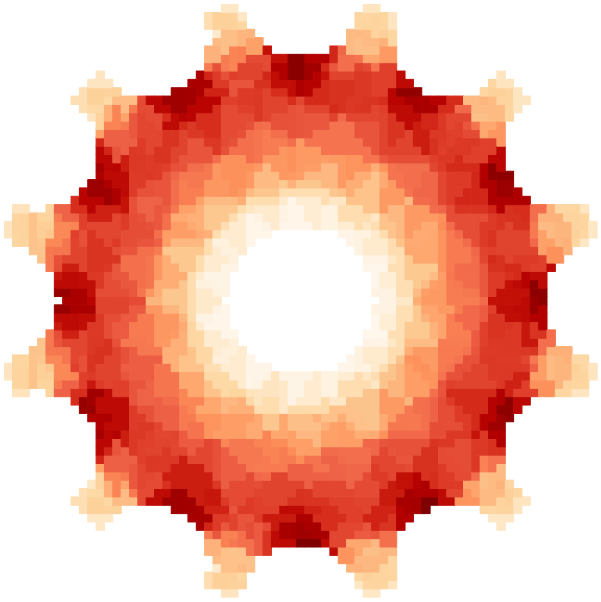


Figure A.4: Example of an output from function `plotSlice()`.

A.3 Heima Example

The two following snippets plot Figure A.5 and Figure A.6 respectively.

```
1 hndl = plotCurrent(Mesh,BF,J,...
2     'Part','Imag','PlaneNormal',[0 1 0 0]);
```

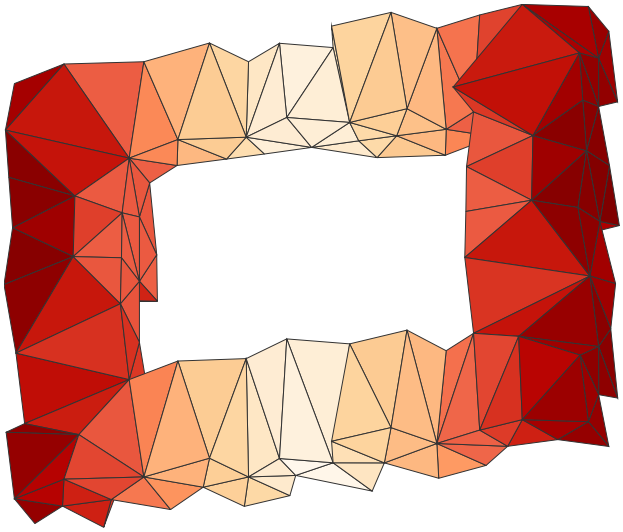


Figure A.5: Example of an output from function `plotCurrent()`.

```
1 hndl = plotSlice(Mesh,BF,J,'Part','Imag',...
```

```
2      'PlaneNormal','y','PatchesCeiling',97);
```

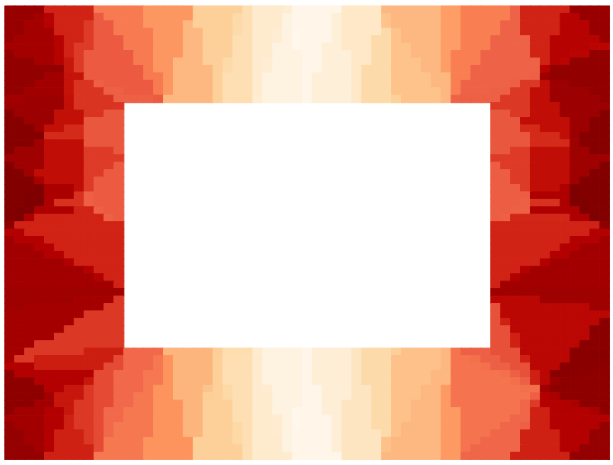


Figure A.6: Example of an output from function `plotSlice()`.

A.4 James Example

The cited scripts draws Figure A.7 and Figure A.8 respectively.

```
1  hndl = plotCurrent(Mesh,BF,J,'Part','Abs');
```

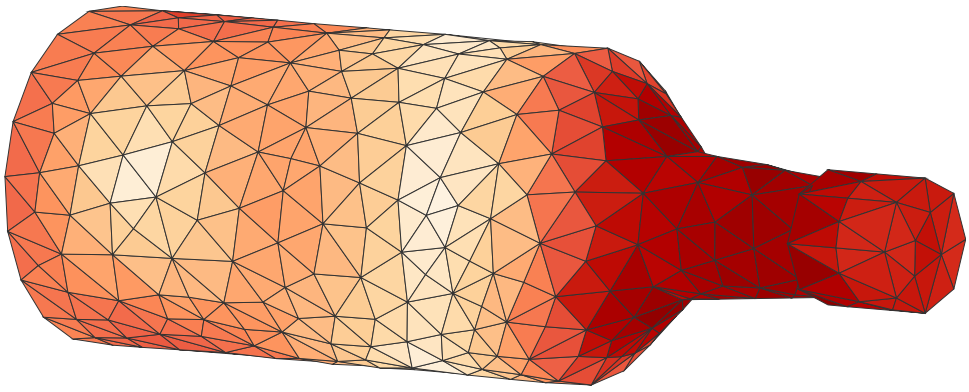


Figure A.7: Example of an output from function `plotCurrent()`.

```
1  hndl = plotSlice(Mesh,BF,J,'Part','Abs',...
2      'PlaneNormal',[0 0.1 1],'PatchesCeiling',86);
```

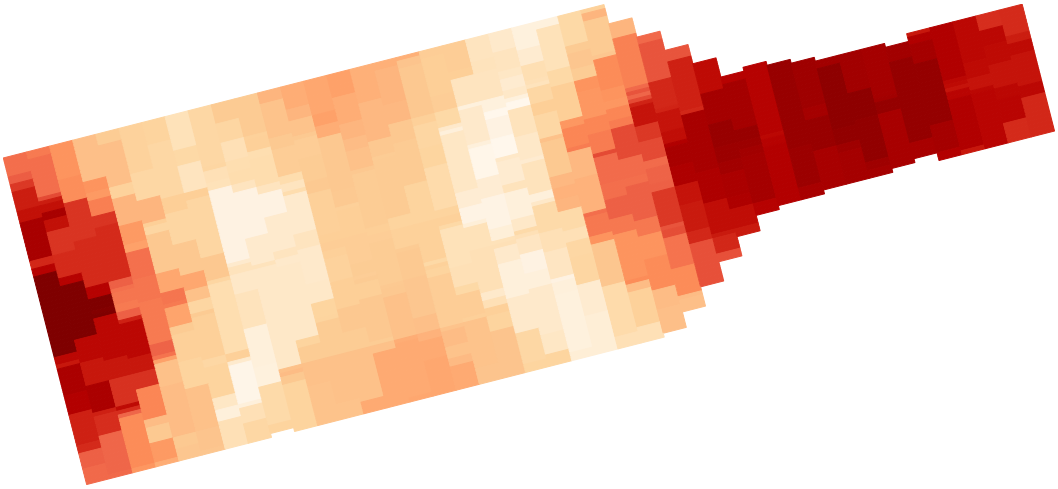


Figure A.8: Example of an output from function `plotSlice()`.