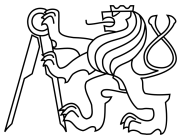
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Triangulation of planar objects and its implementation into the AToM package
Student:	Martin Štrambach
Supervisor:	Ing. Miloslav Čapek, Ph.D.
Study Programme:	Informatics
Study Branch:	Computer Science
Department:	Department of Theoretical Computer Science
Validity:	Until the end of summer semester 2017/18

Instructions

Study the problem of planar triangulation, summarize available tools, their pros and cons and select an algorithm based on your study for implementation in the frame of the AToM project. Describe all important parameters influencing quality of the resulting grid, rate of the algorithm's convergence and specify proper termination criteria. Discuss how these parameters can be improved by selecting some advanced meshing strategy, focus on the speed of the algorithm and its robustness. Final implementation should support features like user-defined density function, mesh junctions and controllability of mesh density. As the second topic, propose and isolate (at least in naive version) algorithm for finding all "loops" and "dipoles" for a given mesh grid that will be later utilized for topological and structural optimization of planar antennas. Project should be accompanied with documentation and basic examples. The algorithm will be tested on canonical examples (polygons).

References

Will be provided by the supervisor.

doc. Ing. Jan Janoušek, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague February 8, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer science

Bachelor's thesis

Triangulation of Planar Objects and Its Implementation into AToM Package

Martin Štrambach

Supervisor: doc. Ing. Miloslav Čapek, Ph.D.

15th May 2017

Acknowledgements

I am grateful to many within the Department of Electromagnetic Field for suggestions and contributions made throughout. Foremost, I would like to thank my supervisor, Miloslav Čapek, for the continuous support of my bachelor study, for his patience, motivation, immense knowledge, and encouragement necessary to pursue such work from the outset. To the many friends, family and grandmother Věra, who have both cheerfully supported and ultimately tolerated my studies – I couldn't have done it without you.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. I further declare that I have concluded an agreement with the Czech Technical University in Prague, on the basis of which the Czech Technical University in Prague has waived its right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act. This fact shall not affect the provisions of Article 47b of the Act No. 111/1998 Coll., the Higher Education Act, as amended.

In Prague on 15th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Štrambach, Martin. *Triangulation of Planar Objects and Its Implementation into AToM Package.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Tato práce je zaměřena na tvorbu trojúhelníkových sítí planárních objektů ve dvou- a třírozměrných prostorech. Takovéto struktury a jejich efektivní tvorba jsou obzvláště důležité pro numerické výpočty. Jsou shrnuty nejběžnější techniky generování trojúhelníkových sítí jako je Delaunyho triangulace. V dalších sekcích jsou popsány algoritmy, které vytvářejí trojúhelníkové sítě v rovině. Zdůrazněna je problematika implementace vybraného algoritmu do anténního toolboxu AToM vyvíjeného v prostředí MATLAB. Společně s algoritmem pro tvorbu trojúhelníkových sítí jsou představeny utility pracující nad vytvořenou sítí pro optimizaci topologie antén.

Klíčová slova trojúhelníkové sítě, Delaunayho triangulace, implementace, AToM toolbox, fyzika, antény, MATLAB

Abstract

This thesis focuses on triangular mesh generation of planar objects in two- and three-dimensional spaces. It is especially important to create such structures for numerical computations effectively. Triangular mesh generation techniques like Delaunay triangulation are summarized together with selected algorithms for effective mesh generation. We emphasize problems which arose during implementation of the selected algorithm into antenna toolbox AToM which is entirely develeped in MATLAB. In the last chapter, we discuss implementation of numerous utilities which are used for mesh topology optimization.

Keywords triangulation, Delaunay triangulation, implementation, AToM toolbox, physics, antennas, MATLAB

Contents

List of Figures

List of Tables

List of Abbreviations

AToM	Antenna Toolbox for Matlab
BEM	Boundary Element Method
CDT	Constrained Delaunay Triangulation
MATLAB	Matrix Laboratory
MoM	Method of Moments
NSGA	Nondominated Sorting Genetic Algorithm
RFID	Radio-Frequency Identification

Introduction

Small antennas are ubiquitous, from computers to wearable electronics like smart watches, RFID chips and so on. It is necessary to make them as electrically efficient as possible usually with the desired shape. It puts high demands on the design of such antennas.

Antenna design, like other fields (*e.g.,* design of cars, fluid dynamics) of human work, underwent information revolution and people started using virtual prototyping in computer simulators to find the best possible shapes and ways how to power all kinds of them. This led to, for many, surprising results which made this technique state of the art. Many new numerical simulations of physical and mathematical phenomena were discovered over the years, but it is either impossible or infeasible to solve these problems analytically in acceptable time, thus none of them is applicable without a proper discretization of the antenna surface, in which geometry is tessellated into meshes of given type of elements.

Antenna toolbox for MATLAB – AToM [1] is software developed primarily for antenna design and optimization. Its core functionality contains the design of antenna geometry together with discretization. This model is then passed into electromagnetic solvers, namely, method of moments [2] and characteristic mode decomposition [3]. These algorithms solve the problem of radiation reformulated into its weak-form [4]. This cannot be done without a proper discretization of the radiating bodies, which establishes important, complicated and numerically challenging task, thus motivating the beginning of this work.

Tessellations must conform to a number of requirements like underlying geometry conformity, optimal element geometry and high resolution in areas of interest. Generation of optimal mesh is, as other covering and partitioning tasks, NP-hard optimization problem [5]. Because of the problem complexity, it is convenient to focus optimization only on areas of interest (*e.g.,* areas around feeding ports). Nowadays, the standard for generation of triangular meshes is Delaunay Triangulation [6], which can be used for other purposes

with its dual Voronoi diagram (both discussed in detail in Chapter 2).

AToM is being developed in collaboration with Department of Electromagnetic Field at Czech Technical University, Department of Radio Electronics at Brno University of Technology and industry partner ESI Group [7], which is one of the leading innovators in virtual prototyping industry.

MATLAB was chosen as the main tool for the AToM development. It offers many useful methods which are already implemented in the software. MATLAB was mainly known for its capability of fast code prototyping, but the language is being constantly developed and over the years, it has become speed-wise comparable with compiled low-level languages like C and Fortran [8]. A set of built-in functions comes with the support of implicit parallelization, which allows us to write high-performance programs using a high-level language.

Goals of the Thesis

The thesis is mainly focused on theoretical development of triangular mesh generation techniques, finding a suitable method for antenna discretization in MATLAB and its implementation into AToM. For the successful conclusion of the thesis, it is necessary to fulfil the following points:

- Summarize available mesh generation tools (see Chapters 1 and 2).
- Review important parameters for mesh generation (see Section 2.4).
- Describe existing algorithms and choose a robust and fast technique for the implementation into AToM (see Section 2.5).
- Implement selected algorithm into AToM (see Sections 3.1 and 3.2).
- Add support for user-defined density functions, mesh junctions, controllability of mesh density (see Section 3.2).
- Propose algorithm for antenna optimization above an arbitrary mesh by finding “loops”, “dipoles” and implement other mesh utilities (see Section 3.4).
- Create examples of basic functionality in AToM.

Chapter 1

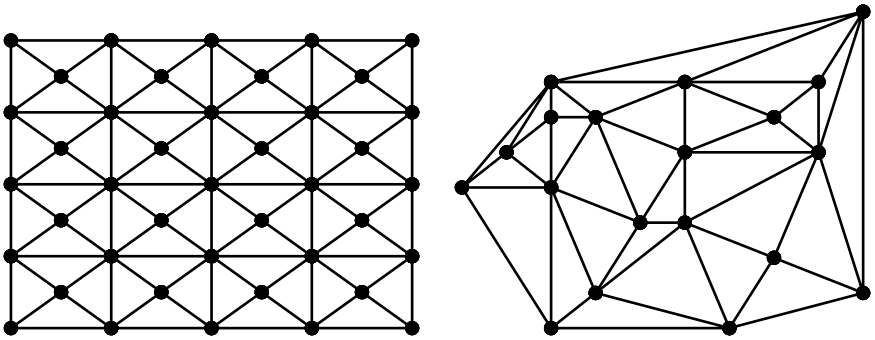
State of the Art

Physical solvers use domain discretization for numerical computations. Mesh is a collection of simple elements which connect points in Euclidean space. Even though we can think of many element shapes (*e.g.,* squares) which would be suitable for discretization, triangular meshes are mostly used for planar objects, since they are conformal with respect to the radiator bodies. This technique uses Delaunay triangulation (introduced in Section 2.2) in order to construct meshes which consist of elements, that are as similar as possible. In the case of Delaunay triangulation, it is an equilateral triangle.

Why do we need equilateral triangles? One would say it is enough to create arbitrary triangulation and then use mathematical solvers. Discretization of underlying integro-differential operators are encumbered with several approximations. The numerical errors coming from this approximation are minimized in case that the triangles are small enough and equilateral [10]. This requirement is in contradiction with the speed of the solver as its numerical complexity is $\mathcal{O}(n^3)$, in which n is a number of unknowns. It is, therefore, obvious that there is some optimal ratio between a number of triangles, their size and quality.

Implemented mesh generation technique must have following properties which are necessary for use in optimization tasks:

- *Speed* – mesh is generated in a feasible time. For use in AToM, it is necessary to generate triangular meshes which consist of up to 10,000 triangles. Larger meshes would be infeasible for numerical solvers.
- *Robustness* – it is possible to generate meshes above arbitrary planar objects with good results, which means mesh is generated with good quality of triangles and all constraints are kept.
- *Scalability* – mesh size can be controlled through numerous parameters. It is also possible to control element density in areas of interest.
- *Domain conformity* – mesh must capture underlying geometry in the best possible way.



(a) Structured mesh. (b) Unstructured mesh.

Figure 1.1: The difference between structured and unstructured meshes.

Meshes can be also divided into two types [9] according to the system of connections between points:

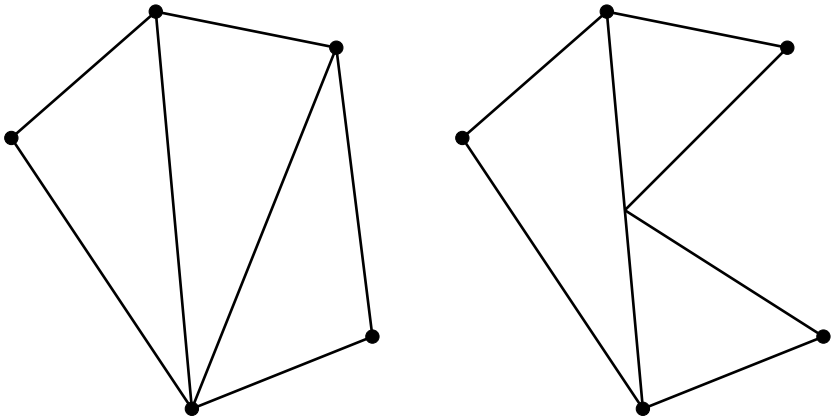
- *Structured mesh* – mesh nodes are numbered in a way that triangle connections can be determined by simple arithmetic.
- *Unstructured mesh* – mesh connections do not follow any mathematical pattern and must be stored in a separate matrix. This implies algorithms using unstructured meshes are more computationally expensive, but also offer greater geometrical flexibility. This type of meshes is discussed in the following text.

Another type of division is by inter-element connectivity:

- *Conforming* – elements connect along common edge or face.
- *Non-conforming* – elements can only partially intersect along common edge or face. This offers flexibility in mesh creation, but it is at the same time computationally expensive.

As stated in [11] three types of mesh generation algorithms dominate nowadays.

- *Grid/quadtree/octree overlay* – algorithms based on an underlying structured mesh. For planar meshes quadtree (recursive decomposition of a plane into square regions) is composed as recursive decomposition of the bounding box into rectangular elements. It is simple and efficient to construct initial mesh using this method. The main disadvantage is that lower quality elements are introduced near geometrical constraints. This is very dependent on alignment with underlying mesh.
- *Advancing front methods* – elements are created one-by-one starting from the boundary. The algorithm continues filling the domain towards the centre. Well shaped elements are created near the boundary, on the



(a) Conforming triangulation. (b) Non-conforming triangulation.

Figure 1.2: Triangulation type based on the inter-element connectivity.

other side elements with poor quality are created in areas where multiple fronts meet.

- *Delaunay refinement* – method starts with initial Delaunay triangulation satisfying all constraints. Mesh is then iteratively refined by removing the worst element. The new point is inserted and the whole mesh is updated. The algorithm terminates when all quality criteria for resulting mesh are met.

Multiple hybridizations of above-mentioned classes of algorithms were created over the years, *e.g.,* Frontal-Delaunay schemes which add new points in a manner of the advancing front methods, but the constraints are maintained with Delaunay triangulation.

AToM is targeted on users which work in other antenna simulation suites like FEKO [12] and CEM One [13]. These programs are usually very expensive and not always very easy to use. AToM offers solvers (*e.g.,* MoM) which are not fully implemented in MATLAB toolboxes and commercial software. In addition, it offers in-house mesh generation with options which are usually hidden from users. This will be utilized in symmetries, optimization, antenna arrays, etc.

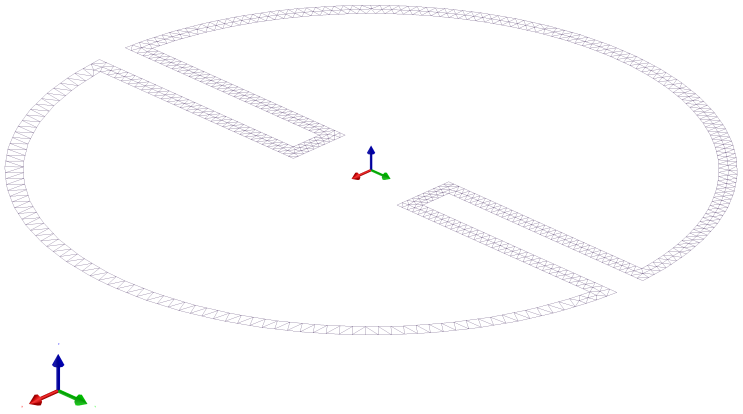


Figure 1.3: A discretized structure in the simulation suite for computational electromagnetics CEM One.

Figure 1.4: Detail of computed electrical current above the discretized structure.

Chapter 2

Triangulation

In general, we can subdivide space into k-dimensional simplices (k-simplex), the simplest convex full-dimensional polyhedron constructed from a minimal set of points in $\mathbb{R}^d$. It can be seen that the boundary of general k-simplex can be decomposed into lower dimensional simplexes.

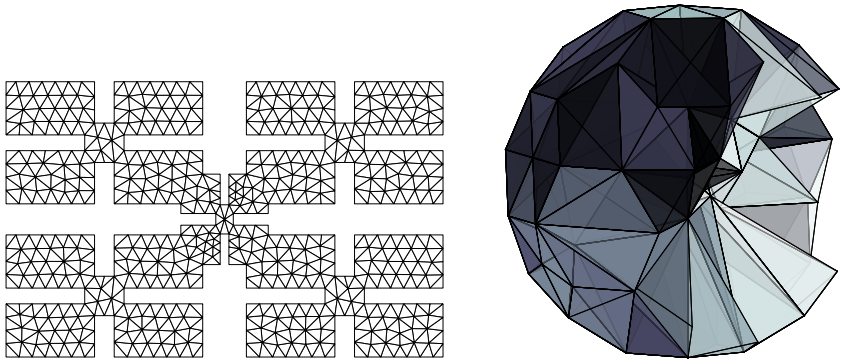
Most common shapes used for mesh generation are triangles for planar objects and tetrahedrons ("pyramids") for spatial objects. This thesis is entirely focusing on triangular (2-simplex) meshes of planar objects in $\mathbb{R}^2$ and $\mathbb{R}^3$.

The term triangulation is generally understood as a subdivision of a planar polygon into a planar graph consisting only of triangles (triangular mesh). In a basic form, we just create triangles by connecting points from a given set. The only valid kind of triangulation for us is when each triangle follows rules that neighbouring triangles only meet node-to-node and edge-to-edge (conforming triangulation, see Chapter 1).

The state of the art technique is Delaunay triangulation, which stands among other triangulations, due to some nice properties, which will be discussed later. Delaunay triangulation can also be extended into more dimensions, *e.g.,* to connect tetrahedrons into Delaunay tetrahedrization. We can go even further into higher dimensions, but such triangulations do not have practical use in industry and especially in the antenna theory. For more information and practical, examples you can visit documentation of MATLAB function `delaunayn` [14].

The more advanced problem is when one has an arbitrary polygon defined as a set of edges (set of points and connections between them). It is impossible to subdivide majority of shapes into a triangulation formed entirely of equilateral triangles, which we consider as an ideal triangulation. We need to utilize some kind of technique which finds the best position for newly inserted internal points of the polygon. These points are then connected using Delaunay triangulation respecting existing connectivity constraints. Some of the basic algorithms for polygon discretization are described in Section 2.5.

For better understanding of what Delaunay triangulation is and which



(a) Triangulation. (b) Tetrahedrization.

Figure 2.1: Example of 2D and 3D meshes.

properties it has, we first need to introduce two terms: Voronoi diagram and Voronoi region.

2.1 Voronoi Diagram

Having an arbitrary set of points $S \subset \mathbb{R}^2$, we can subdivide space into regions of their influence, called *Voronoi regions*, which together form *Voronoi diagram* (sometimes called Dirichlet tessellation). See Figure 2.2.

Voronoi region of a point $\mathbf{P} \in S$ is a convex polygon of a set of points $\mathbf{X} \in \mathbb{R}^2$, which have distance from all other points $\mathbf{Q} \in S$ smaller or equal to distance to $\mathbf{P}$.

$$V_{\mathbf{P}} = \{\mathbf{X} \in \mathbb{R}^2 : \ \|\mathbf{X} - \mathbf{P}\| \leq \|\mathbf{X} - \mathbf{Q}\|, \ \forall \mathbf{Q} \in S\}\,. \tag{2.1}$$

Let us consider half plane of points which are at least as close to $\mathbf{P}$ as to $\mathbf{Q}$.

$$H_{\mathbf{PQ}} = \{\mathbf{X} \in \mathbb{R}^2 : \ \|\mathbf{X} - \mathbf{P}\| \leq \|\mathbf{X} - \mathbf{Q}\|\}\,. \tag{2.2}$$

Then Voronoi region of $\mathbf{P}$ is an intersection of half planes $H_{\mathbf{PQ}}$, $\forall \mathbf{Q} \in S \setminus \{\mathbf{P}\}$.

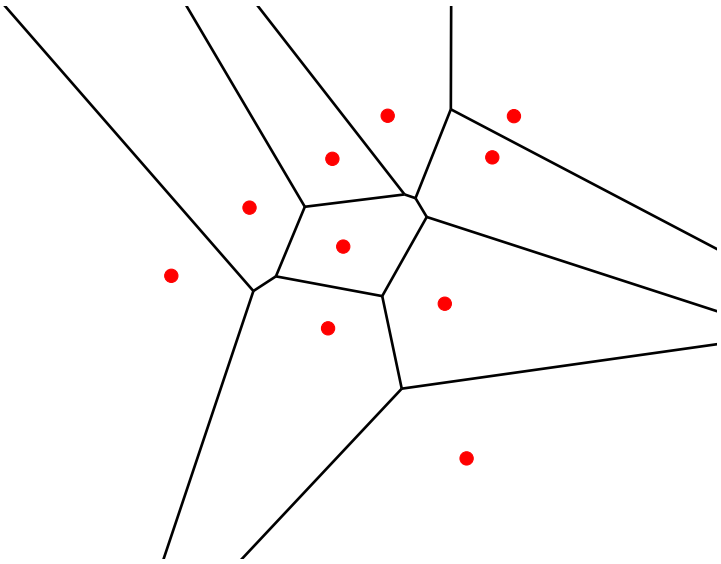


Figure 2.2: Voronoi diagram.

2.2 Delaunay Triangulation

Delaunay triangulation was first introduced in 1934 [6]. It is dual to Voronoi diagram. Delaunay edge exists between two points $\mathbf{P}_1, \mathbf{P}_2 \in S$ if and only if two Voronoi regions intersect along the common edge. Proof of this claim can be found in [5].

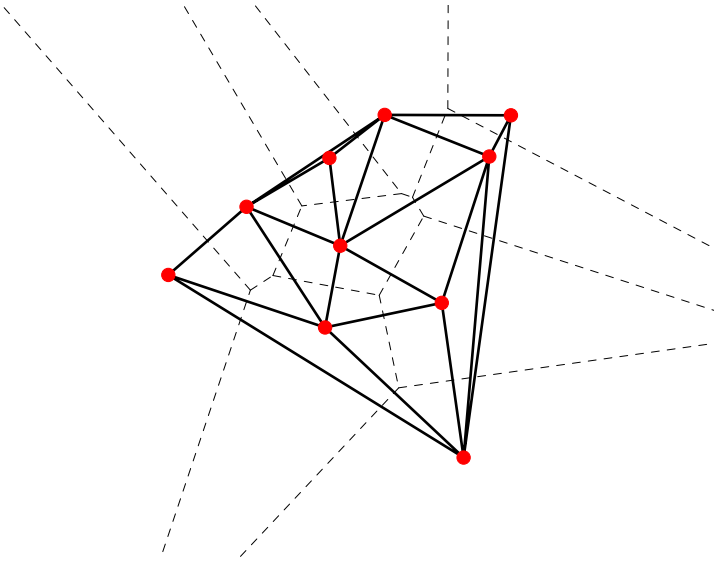
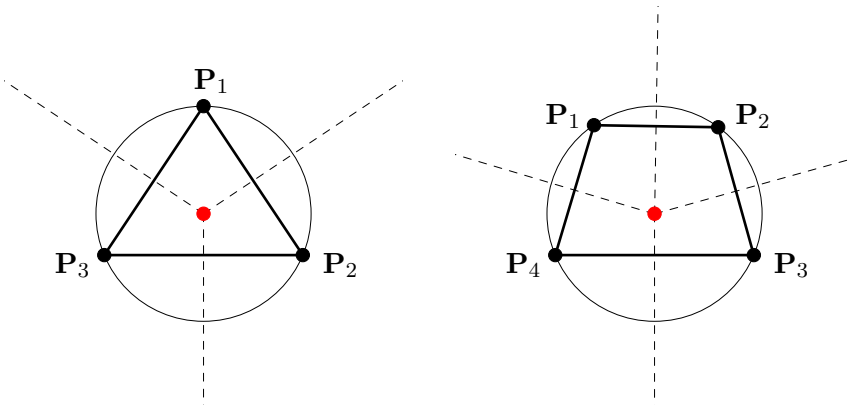


Figure 2.3: Delaunay triangulation created from Voronoi diagram.



(a) General position. (b) Degenerate position.

Figure 2.4: Voronoi vertex positions in a plane.

The general position of Voronoi regions is when three of them meet at the same point. On the other hand, some degenerate cases may occur when four or more Voronoi regions meet at the same point – *Voronoi vertex*. This implies those four points would connect into a quadrilateral. If we consider this case deeply it would mean four points lie on the same circle, which is very unlikely, because even a small perturbation in the point position would result in the general case. If no four points of S lie on the same circle, then Delaunay triangulation is unique [15].

Circumcircle claim. Let $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3 \in S$, then triangle $T_1 = \{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3\}$ is a triangle of Delaunay triangulation if and only if its circumcircle does not contain any other point $\mathbf{P}_i \in S$. We can use the previous property to define *locally Delaunay edges*. Edge $\mathbf{P}_1\mathbf{P}_2$ is locally Delaunay edge of a triangulation K, when it fulfils one of the following criteria:

1. Edge $\mathbf{P}_1\mathbf{P}_2$ is shared only by one triangle, thus it is a boundary edge of the triangulation.
2. Edge $\mathbf{P}_1\mathbf{P}_2$ is shared by triangles T_1 and $T_2 = \{\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_4\}$. When we construct circumcircle of T_1, then $\mathbf{P}_4$ lies outside of the circle.

Delaunay lemma. When every edge of a triangulation is locally Delaunay, then we call it Delaunay triangulation.

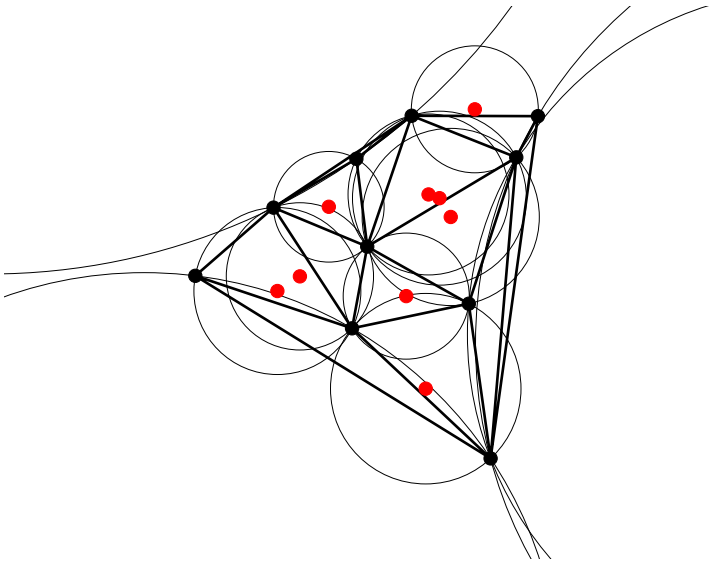


Figure 2.5: Illustration of circumcircle property.

It has been proven in [16] that every Delaunay triangulation maximizes minimal angle in the triangulation, which enables us to construct triangulations with triangles as close to equilateral as possible.

We must also note that every set of points in a plane can be triangulated using Delaunay triangulation [5].

2.3 Constrained Delaunay Triangulation

Constrained Delaunay triangulation [15], [5] (CDT) is a triangulation with following properties:

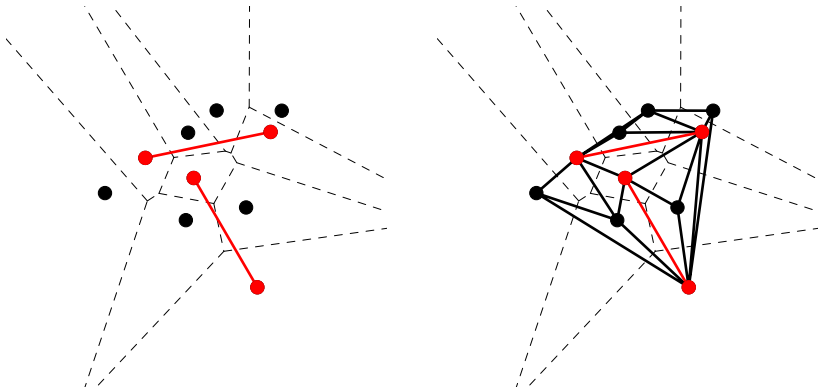
1. Specified edges are included in the triangulation.
2. It is as close as possible to Delaunay triangulation.

Let the G be a planar graph which contains fix edges e of a triangulation K. Two points $\mathbf{P}_1, \mathbf{P}_2 \in \mathbb{R}^2$ are visible from each other when $\mathbf{P}_1\mathbf{P}_2 \cap \mathbf{Q} = \emptyset$, $\forall \mathbf{Q} \in K \wedge \mathbf{P}_1\mathbf{P}_2 \cap e = \emptyset, \ \forall e \in G$.

Then the triangulation K is called *constrained Delaunay triangulation*, when each edge of G is an edge of K and for all remaining edges $e \in K$ with endpoints $\mathbf{P}_1\mathbf{P}_2$ there exists circle c with the following properties:

1. Points $\mathbf{P}_1, \mathbf{P}_2 \in S$ lie on c.
2. $\mathbf{P}_1$ and $\mathbf{P}_2$ are visible from each other and every other point $\mathbf{Q} \in G$ in the interior of c is invisible from all interior points $\mathbf{X} \in (\mathbf{P}_1, \mathbf{P}_2)$.

When we take G without edges, we get the definition of unconstrained Delaunay triangulation.



(a) Graph G. (b) CDT constructed from G.

Figure 2.6: Example of a graph G and resulting CDT K.

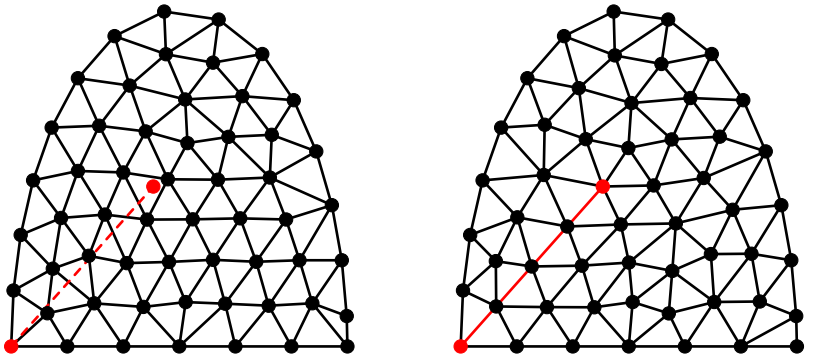
2.4 Triangulation Terminology

Fix Point

Some points on the domain need to be fixed. It means that they can not be shifted during the triangulation process. Typically boundary nodes must not be moved otherwise, shape of the polygon would change.

Fix Edge

Definition of fix points can be extended into fix edges. A fix edge is a mesh constraint defined as a connection by a straight line between two fix points in the resulting mesh. This does not necessarily mean no other points can lie on the edge. The edge is usually divided into more smaller elements according to density function which preserves the geometrical property of the original edge.



(a) Unsatisfied fix edge constraint. (b) Satisfied fix edge constraint.

Figure 2.7: Example of fix edge constraint.

Holes

A hole is a closed polygonal region inside our meshed domain, which does not contain any nodes, therefore, no triangles in its interior. Its boundary is defined in the same way as the boundary of the polygonal region itself. Some algorithms require the definition of the polygon boundary in clockwise order and hole regions counter-clockwise.

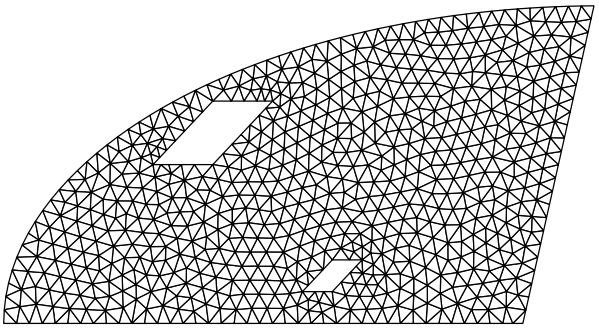


Figure 2.8: A polygonal region with holes.

Density Function

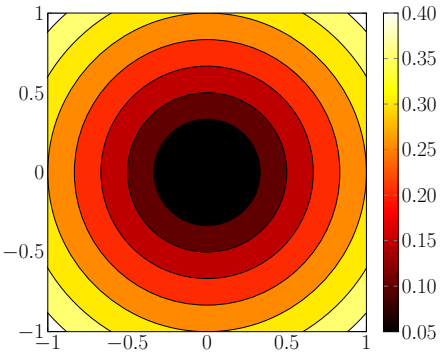
It is desired to control element size over the whole polygonal domain. Function $h(x, y)$ in two-dimensional space returning the size of the elements on the given position is used for this purpose. Most commonly used is a uniform function which returns elements of the same size for all points in the domain.

Listing 2.1: Example of an uniform density function in MATLAB.

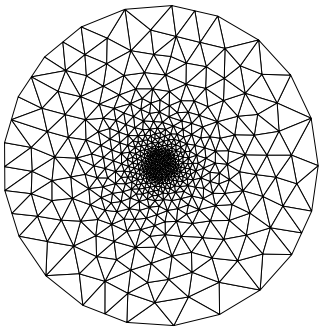
```
1          h = @(x, y) 1*ones(size(x, 1),1)
```

Listing 2.2: Example of a non-uniform density function in MATLAB.

```
1          h = @(x, y) max(0.3*sqrt(x.*x + y.*y), 0.05)
```



(a) Function heat map.



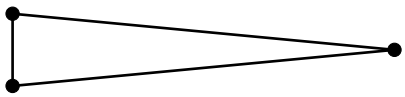
(b) Mesh with 1015 triangles.

Figure 2.9: Example of density function from the Listing 2.2 and its impact on the mesh generation.

Triangle Quality

Only very small set of problems can be discretized using perfectly equilateral triangles. In real world applications, domains have various dimensions and multiple geometrical constraints. Triangles usually differ by edge size and inner angle size. It is, therefore, necessary to measure the quality of individual elements.

In order to quantify the quality of elements, it is necessary to assign score q to every triangle in a triangulation based on its geometrical configuration which is scale-invariant. It is important to take in consideration that undesirable properties are overly small and large angles, therefore, skewed and distorted triangles.



(a) A needle with $q = 0.1419$.

(b) A cap with $q = 0.1050$.

Figure 2.10: Triangles with poor quality.

While many strategies were developed to get quality score for triangles [17], many of them are impractical due to high computational cost. Our selected measure for triangle quality was introduced in [18]

$$q = \frac{4\sqrt{3}a}{h_1^2 + h_2^2 + h_3^2}, \tag{2.3}$$

where a is triangle area and h_i is edge length. We get values $q \in [0, 1]$, where value 0 is equal to three points on the same line and 1 is an equilateral triangle.

2.5 Description of Basic Algorithms

2.5.1 DistMesh

DistMesh [19] is a simple refinement algorithm for creating high quality unstructured meshes in 2D. It is iteratively changing point positions to find triangulation force equilibria as an analogy to physical truss structure or equivalently a structure of springs. Nodes are then connected using Delaunay triangulation. Authors emphasized code readability and simplicity, which led to simple MATLAB code that can be used as a foundation for more robust solutions.

To compute force equilibrium for a set of points $\mathbf{p}$: (Nx2), in which N is number of points, we are looking for the solution with force vector $\mathbf{F}(\mathbf{p}) = 0$ of the following equation

$$\mathbf{F}(\mathbf{p}) = [\mathbf{F}_{\text{int},x}(\mathbf{p}), \mathbf{F}_{\text{int},y}(\mathbf{p})] + [\mathbf{F}_{\text{ext},x}(\mathbf{p}), \mathbf{F}_{\text{ext},y}(\mathbf{p})]\,, \tag{2.4}$$

where $\mathbf{F}_{\text{int}}$ are internal forces transferred through edges and $\mathbf{F}_{\text{ext}}$ are forces pushing to the domain boundary, which are equal to force coming from inside. This secures the fixed position of boundary nodes. Force $\mathbf{F}(\mathbf{p})$ is in each iteration dependent on mesh topology which is created by Delaunay triangulation. Since points move in each iteration of the algorithm, every node can have new neighbours, which means $\mathbf{F}(\mathbf{p})$ is a discontinuous function.

The system can be solved using forward Euler method where at discretized time t_n the solution is $\mathbf{p}_n \approx \mathbf{p}(t_n)$. Points are then updated by

$$\mathbf{p}_{n+1} = \mathbf{p}_n + \triangle t \mathbf{F}(\mathbf{p}_n)\,. \tag{2.5}$$

DistMesh supports user defined element size function $h(x, y)$, which determines the relative distribution, not the actual size of the elements, due to the artificial repulsive force which is applied to the most of the edges. Good value of this repulsive force was set by authors to $\mathbf{F}_{\text{scale}} = 1.2$.

The algorithm takes as input a set of points already placed inside the polygonal domain. This is significant part of more complex algorithms, which find initial position of internal nodes. We can start with random distribution of points. It is recommended to use points placed on equidistant grid for uniform distribution $h(x, y)$ or for non-uniform distribution $h(x, y)$ create initial points distribution weighted by probabilities proportional to $1/h(x, y)^2$.

Performance of this algorithm is very dependent on the initial distribution of points. Custom mesh generator was implemented in AToM, but it had slow converge (see 3.2).

2.5.2 Point Insertion Using Bowyer-Watson Algorithm

Bowyer-Watson Algorithm [21]

This method is based on Circumcircle property (Section 2.2) that no point of Delaunay triangulation can lie within circumscribed circle of any triangle. It removes all triangles which violate previous property and reconnects newly inserted point with the rest of the triangulation. It can be shown [20] that such set of triangles is contiguous polygon forming a cavity. When connecting boundary nodes with the newly inserted point we always get valid Delaunay triangulation.

Algorithm 1 Bowyer-Watson pseudocode above given triangulation

Add the new point P to the set of points
Create a set `badTriangles` of triangles whose circumcircles contain point **P**
Find all edges (`boundaryEdges`) from `badTriangles` which are reference exactly one triangle
Remove all `badTriangles` from the triangulation
Connect every edge from `boundaryEdges` with **P**
Add all new triangles to the triangulation

Since all triangles removed in each iteration are contiguous, we can store triangles in a way that we have information about neighbours of each triangle and simple tree search algorithm can be used to find a triangle to be deleted, thus insertion of a new point in the existing triangulation with n points can be done in $\mathcal{O}(\log n)$ operations. Then Delaunay triangulation of n points has complexity $\mathcal{O}(n \log n)$. However when all triangles must be removed the algorithm degrades to $\mathcal{O}(n^2)$ operations.

Initial Boundary Triangulation

Following two methods use Bowyer-Watson algorithm as reconnection method of already existing triangulation, therefore we need to provide an initial triangulation, which is at the same time body conforming, *e.g.,* initial triangulation can be constructed as CDT of a bounding box or CDT of the convex hull, with marked internal and external triangles.

Voronoi-vertex Insertion

The algorithm is a representative of Delaunay refinement algorithms. The first position for the point insertion is the centre of circumscribed circle [22], where the mesh is by the means of circumcircle property coarsest. Triangles with poor quality must be chosen for elimination. Usually, such triangles have large

areas or bad aspect ratios. In this way, bad triangles are eliminated and the resulting mesh tends to cause clustering according to boundary discretization.

Let us define function $f(\mathbf{X})$ of a characteristic property (*e.g.,* radii of circumcircle or ratio between inscribed and circumscribed circle) of a triangle as a function of its position based on coordinates $\mathbf{X}$ of its circumcircle. We can then describe all triangles by ratio

$$\alpha_k = \frac{\rho_k}{f(\mathbf{X}_k)}\,, \tag{2.6}$$

where ρ_k is the actual value of a characteristic property and $\mathbf{X}_k$ is the centre of the circumcircle. Then we continue inserting new points unless $\alpha_k < 1$, $\forall k \in [1, n]$, where n is a number of triangles in the triangulation K.

Algorithm 2 Voronoi-vertex insertion pseudocode

```
Create body conforming initial triangulation
while maximal ratio α ≥ 1 do
    Find the worst triangle with the maximum ratio α
    Insert the new point at the circumcircle centre of the worst triangle
    Reconnect the triangulation by the means of Bowyer-Watson algorithm
    Add new triangles to the existing triangulation
    Compute α for newly created triangles
end while
```

One special case must be handled when inserting new points. Voronoi vertex can be outside of the bounded domain, which means we must either skip this insertion or introduce some refinement technique.

Voronoi-segment Insertion

The second method inserts the new point into Voronoi region, rather than on Voronoi vertex. In addition it does not eliminate the worst triangle, but uses triangle type division, which allows to create ideal triangles according to the density function.

Like the previous method, it uses initial triangulation along with *external* and *internal* triangles. Internal triangles are then subdivided into *accepted*, *active* and *waiting*. Most of the triangles in the initial triangulation are non-accepted. Active triangles are neighbours with at least one accepted or one external triangle. The rest of non-accepted triangles is considered waiting.

Let us assume an active triangle neighbouring with an arbitrary accepted/external triangle. Figure 2.11 depicts this situation with an active triangle on the right and an accepted/external triangle on the left. These two triangles share an edge **PQ**. New point **X** is placed on the line connecting point $\mathbf{C}_A$, which is active triangle circumcentre, and **M** which is the midpoint of **PQ**.

Prescribed local feature size for the point $\mathbf{M}$ is

$$\rho_{\mathbf{M}} = f(\mathbf{M})\,. \tag{2.7}$$

We would like to have the new point inserted on the intersection of $\mathbf{MC}_A$ with the circle passing through $\mathbf{PQ}$ with radius $\rho_{\mathbf{M}}$.

It might happen that desired value of $\rho_{\mathbf{M}}$ is smaller than $p = \|\mathbf{P} - \mathbf{Q}\|/2$ which is the smallest radius of the circle passing through $\mathbf{PQ}$. Radius $\rho_{\mathbf{M}}$ has also an upper boundary. It is limited by the size of the circle passing through points $\mathbf{P}, \mathbf{Q}$ and $\mathbf{C}_A$

$$\rho = \frac{p^2 + q^2}{2q}\,, \tag{2.8}$$

where $q = \|\mathbf{C}_A - \mathbf{M}\|$. Limited value of $\rho_{\mathbf{M}}$ is then defined as

$$\hat{\rho}_{\mathbf{M}} = \min\{\max\{\rho_{\mathbf{M}}, p\}, \rho\}\,. \tag{2.9}$$

Position of point $\mathbf{X}$ is defined as

$$\mathbf{X} = \mathbf{M} + d\hat{\mathbf{e}}\,, \tag{2.10}$$

where $d = \|\mathbf{X} - \mathbf{M}\|$ and unit vector $\mathbf{e}$ are

$$d = \hat{\rho}_{\mathbf{M}} + \sqrt{\hat{\rho}_{\mathbf{M}}^2 - p^2}\,, \tag{2.11}$$

$$\hat{\mathbf{e}} = \frac{\mathbf{C}_A - \mathbf{M}}{\|\mathbf{C}_A - \mathbf{M}\|}\,. \tag{2.12}$$

When $\|\mathbf{P} - \mathbf{Q}\|$ is too short with respect to q, then the new point is inserted at the Voronoi vertex of the active triangle which is exactly what Voronoi vertex insertion method does (It is a special case of Voronoi-segment insertion.).

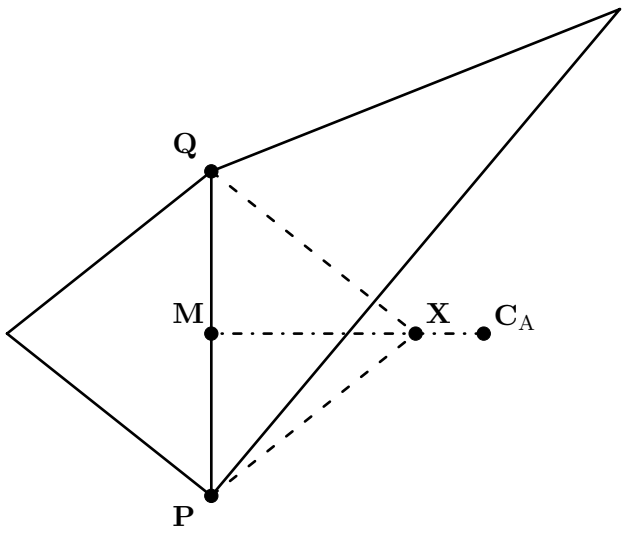


Figure 2.11: Example of the new node position using Voronoi-segment insertion

New triangles need to be reassigned with types described above. The algorithm is moving from the boundary to the centre of the domain similar to advancing front methods, but at the same uses Delaunay triangulation to maintain constraints, therefore it is Frontal-Delaunay hybrid (see Chapter 1). All new triangles are always internal since no external one can be removed. Midpoint $\mathbf{M}_i$ of every new triangle is considered accepted when

$$\frac{\rho_{\mathbf{M}i}}{\rho_i} < \delta\,, \tag{2.13}$$

where $\rho_{\mathbf{M}i} = f(\mathbf{M}_i)$, ρ_i is circumcircle radii and δ is empirical value which can be, *e.g.,* $\delta = 1.5$. Triangles adjacent to the new ones must be subdivided into active and waiting. The algorithm stops when there are no active triangles left.

Algorithm 3 Voronoi-segment insertion pseudocode

```
Create body conforming initial triangulation
Divide triangles into internal and external
Divide internal triangles into accepted, active and waiting.
Order triangles by their circumcircle radii
while numberOfActive > 0 do
    Take the triangle with the biggest circumcircle radii
    Insert the new point according to the Voronoi Segment criterion
    Reconnect the triangulation by the means of Bowyer-Watson algorithm
    Add new triangles to the existing triangulation
    Divide triangles next to the new ones into active and waiting
end while
```

2.5.3 Mesh2D

Mesh2D [23] is an algorithm for creating planar triangular meshes of arbitrary polygons in $\mathbb{R}^2$. Its core functionality is based on the quadtree decomposition algorithm together with CDT, Laplacian-like smoothing based on the DistMesh algorithm and elimination of bad triangles in a manner of Frontal-Delaunay algorithms. The whole algorithm was re-worked in the beginning of 2017 [24] and the latest released version is 3.0.0. The following description is for functionality present in the version 2.4, which was the newest version at the time when the work on the new mesh generation core for AToM began and first changes were deployed.

The algorithm offers an interface for single polygonal faces and multiple connected faces. For connected faces, it is assured that points on the boundary will be unique and element size will smoothly transition between them.

Input for the algorithm is a set of points in $\mathbb{R}^2$ together with a list of connections (connections can be omitted when points are sorted clockwise). Other options for mesh quality are maximal length of edges and a user defined

density function in the form $f(x, y)$. Therefore special treatment must be performed to use for 3D planar objects, described in detail in Chapter 3. Firstly set of nodes is checked for duplicate nodes, edges and non-referenced nodes which are eventually removed. To assure the same mesh for the same polygons only differently rotated in the plane XY, the domain is rotated that longer edge of the bounding box is parallel to axis y. Then the whole domain is decomposed according to density function and maximal edge length into an unbalanced quadtree (internal nodes have different depth of decomposition). Boxes of the quadtree decomposition are split diagonally to form an initial triangulation. At the end, the whole mesh is rotated back with respect to the initial rotation.

When the domain is decomposed all boundary edges are split to respect constraints given by density function, maximal edge length and values interpolated from the background mesh.

The main optimisation routine is based on the spring smoothing using force equilibria introduced in DistMesh. The initial set of points is a set of boundary points combined with points from the quadtree which are inside the polygonal domain. The mesh is being iteratively smoothed until the maximum edge length change between two iterations is less than given value (implicit value is 2 %). Mesh is repaired in each iteration (no duplicate edges and nodes). New nodes are inserted to the circumcircle of bad triangles with small angles or very long edges (Voronoi-vertex insertion).

Before the algorithm terminates the mesh is again fixed and nodes in all triangles are reordered counter-clockwise.

2.5.4 Matlab Triangulations

`triangulation` class [25]

`triangulation` class is an in-memory representation of 2D and 3D data. This means triangulation is constructed using existing set of *nodes* and *connectivityList*. This class offers a group of methods on top of the triangulation. Following are important for computations in AToM:

`circumcenter`	centre of triangle circumcircle,
`edgeAttachments`	neighbouring triangles connected to a given edge,
`freeBoundary`	edges referenced by one triangle in the triangulation,
`incenter`	incenter of a triangle,
`neighbors`	neighbours to a given triangle,
`vertexAttachments`	triangles attached to a given vertex.

`delaunayTriangulation` class [26]

`delaunayTriangulation` is a subclass of the `triangulation` class. It constructs valid Delaunay triangulation from a set of points in 2D or 3D. Such

triangulation is created over the polygonal domain of the convex hull. 2D triangulation can take advantage of constraints, which allow adding a fix edge between two points, thus creating CDT. It has all above-mentioned methods from triangulation class and adds three more:

`convexHull`	convex hull of a set of nodes,
`isInterior`	checks whether a triangle is inside of the CDT,
`voronoiDiagram`	returns Voronoi vertices and regions from a set of nodes.

Visualization

- `triplot` [27] – triangular 2D mesh plot, takes as input a triangulation or set of nodes and connectivity list,

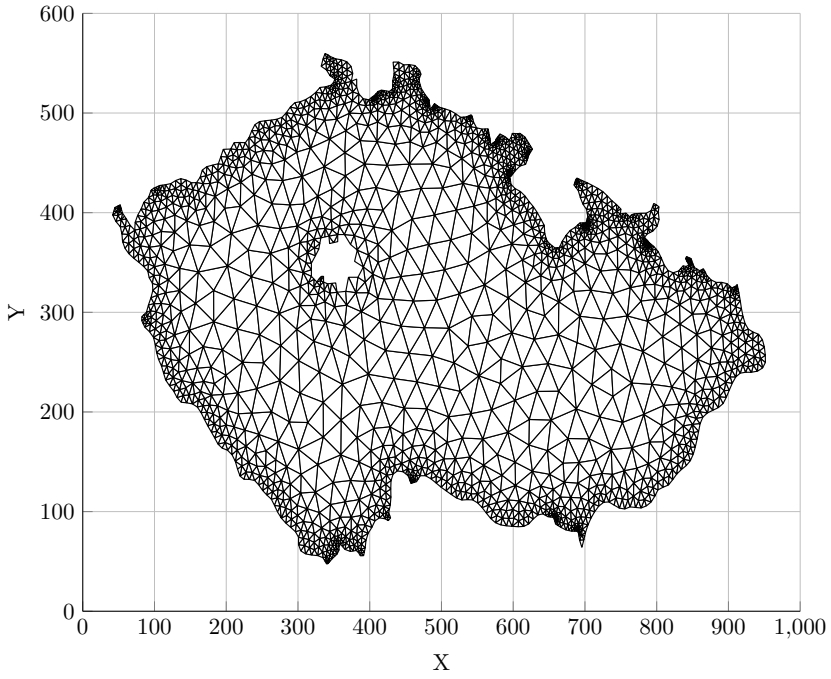


Figure 2.12: Example of a `triplot` plot.

- `trimesh` [28] – triangular mesh plot, displays 2D and 3D triangulations, enables setting of colour proportional to the surface height,
- `trisurf` [29] – triangular 3D surface mesh plot, only 3D triangulation can be displayed using this function with surface colour proportional to the surface height.

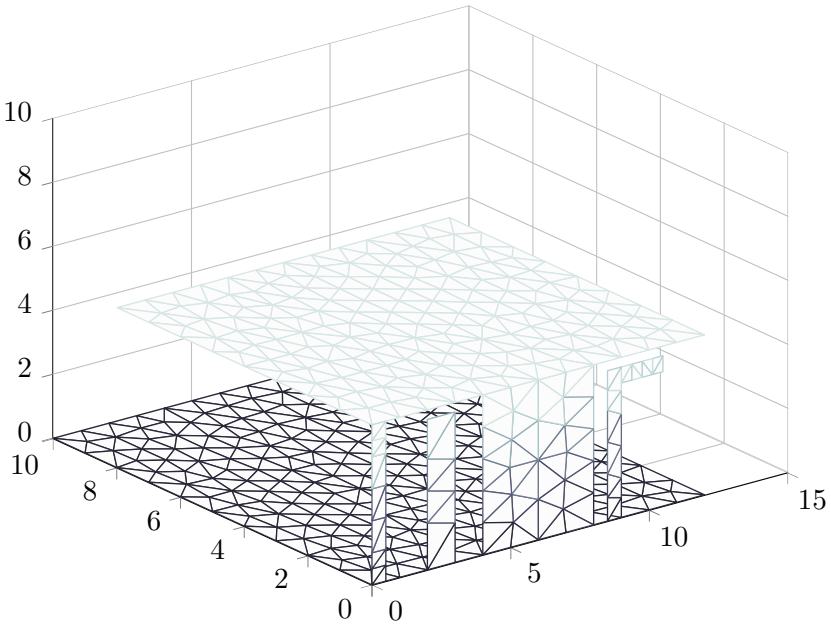


Figure 2.13: Example of a `trimesh` plot.

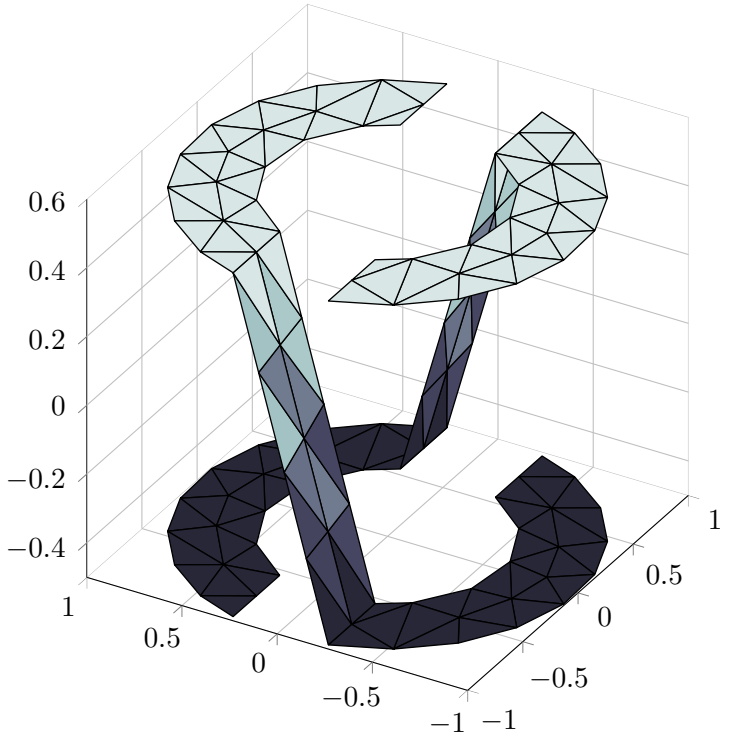


Figure 2.14: Example of a `trisurf` plot.

Chapter 3

MATLAB Implementation

3.1 AToM and Models Connection

Geometry Representation

Geometry model (`Geom`) represents different geometrical objects and defines their methods. Each 2D object is composed of simple primitives – `Line`, `EllipseArc` and `EquationCurve`. Following 2D objects can be represented in Geom:

`Ellipse`	object from ellipse arcs.
`Parallelogram`	object from four parallel lines.
`Polygon`	arbitrary shape constructed from lines.
`PolyLoop`	arbitrary closed object composed of lines, ellipse arcs and equation curves.

Mesh Representation

All discretized curves and planes are represented by `MeshObject`, which saves data of a given object – nodes, connectivity and local metadata. `Mesh` exists only in one instance per project and collects data from all `MeshObject` objects – nodes, elements, which are split into 1D edges for curves and 2D edges and connectivity for planar objects.

Metadata for the `Mesh` object are maximal element size, user defined density function and mesh density, which defines the number of elements per wavelength (described in Section 3.2). Element size itself is dependent property [1]based on the value of mesh density. Other properties, even though it is possible to represent them in matrix form, are dependent because it is much easier to compute them dynamically then ensure data validity after each modification of the mesh.

[1]A special type of property in MATLAB, whose value depends on other values and properties and it is computed when the property is queried.

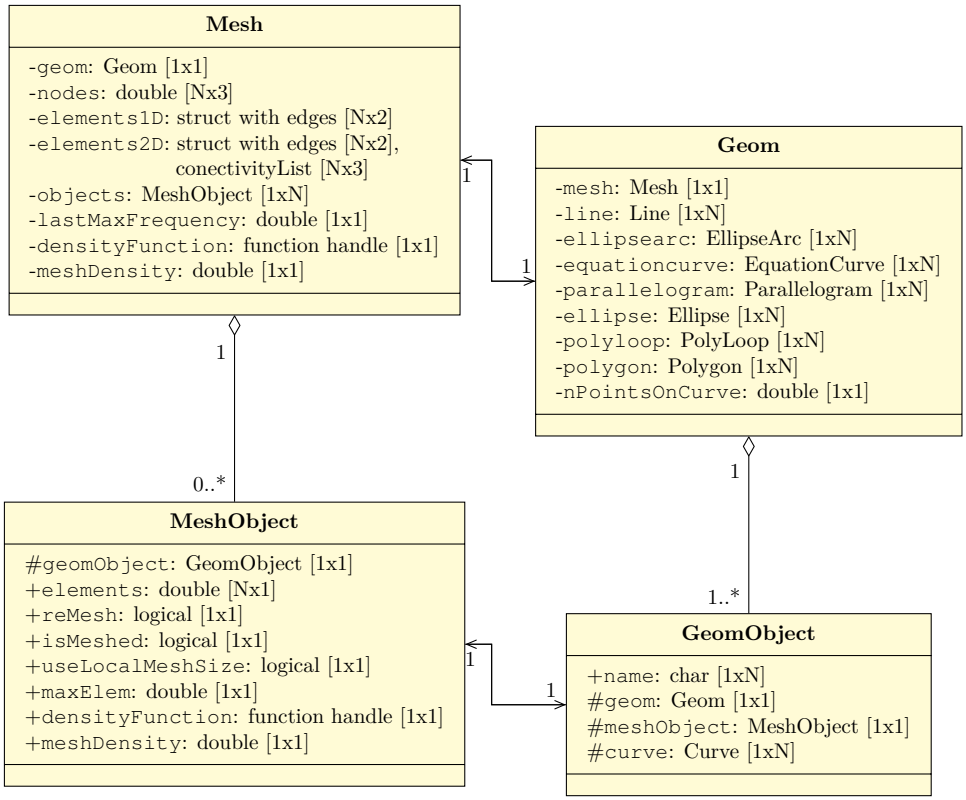


Figure 3.1: Class diagram of mesh and geometry.

Solver

Method of moments solver takes generated mesh as an input along with various metadata, which is important to construct proper basis functions [30]. It is necessary to pass following information:

`Nodes – [Nx3]`	coordinates in $\mathbb{R}^3$.
`ConnectivityList – [Nx3]`	indices into nodes.
`Edges – [Nx2]`	start and end nodes.
`EdgeCentroids – [Nx1]`	centre nodes of each edge.
`EdgeLength – [Nx1]`	edge lengths.
`TriangleArea – [Nx1]`	area of each triangle.
`TriangleCentroids – [Nx1]`	triangles incenters.
`TriangleEdges – [Nx3]`	indices to edges for each triangle.

Together with the metadata about the mesh it is a must to pass data about the geometry, *e.g.,* the geometry of feeding ports. This is the main reason why we need to preserve geometrical constraints in the mesh. AToM allows port definition above existing geometry using endpoints, therefore this constraint must be kept in the generated mesh using CDT.

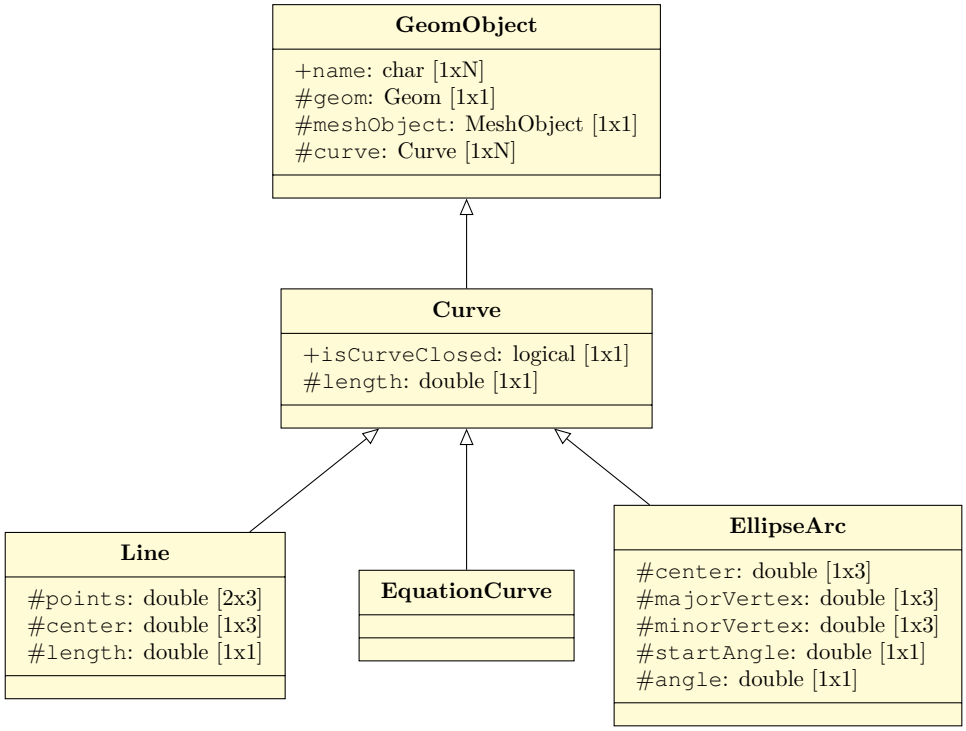


Figure 3.2: Class diagram of 1D curves.

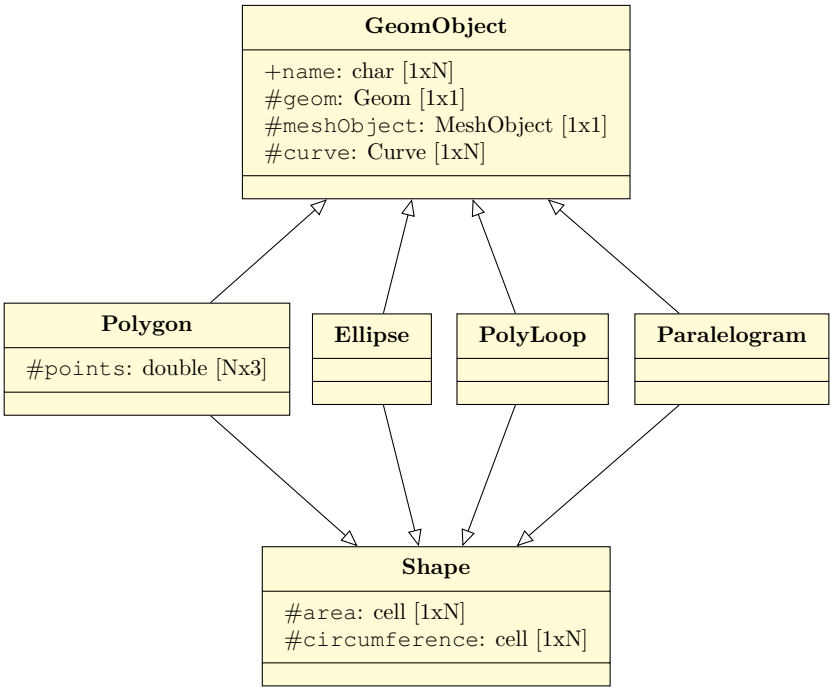


Figure 3.3: Class diagram of 2D objects.

3.2 Mesh Model

Global and Local Coordinate Systems

Before we can start algorithm for mesh generation in 2D, we need to transform 1D mesh coordinates into two-dimensional space. This means we need to introduce a local coordinate system for geometrical objects (every geometrical object in AToM is a planar object but in the three-dimensional space). This transformation changes all points coordinates to $z = 0$ for all points but keeps object proportions.

It is necessary to find point $\boldsymbol{O}$ which is the origin of our local coordinate system and perpendicular vectors $\hat{\mathbf{x}}', \hat{\mathbf{y}}'$ and $\hat{\mathbf{z}}'$ which will form the orthonormal basis of the local coordinate system.

For polygons, we take arbitrary point as the origin $\boldsymbol{O}$. Then two vectors $\hat{\mathbf{x}}, \hat{\mathbf{y}}$ pointing from $\boldsymbol{O}$ that $\hat{\mathbf{x}}' \times \hat{\mathbf{y}}' \neq 0$. When $\hat{\mathbf{x}}' \cdot \hat{\mathbf{y}}' \neq 0$ vectors $\hat{\mathbf{x}}'$ and $\hat{\mathbf{y}}'$ must be forced to be orthonormal. Vector $\hat{\mathbf{x}}'$ is kept in the position and $\hat{\mathbf{y}}'$ is rotated by angle

$$\alpha = \frac{\pi}{2} - \operatorname{acos}\left(\hat{\mathbf{x}}' \cdot \hat{\mathbf{y}}'\right) \tag{3.1}$$

around axis $\hat{\mathbf{z}}' = \hat{\mathbf{x}}' \times \hat{\mathbf{y}}'$. This operation is well-defined for some geometrical objects, *e.g.,* ellipse – $\boldsymbol{O}$ is the centre of the ellipse, $\hat{\mathbf{x}}'$ is the major axis, $\hat{\mathbf{y}}'$ is the minor axis.

When we have axes of the local coordinate system, each node $\mathbf{N}_i$ of K is transformed to the local coordinates as following

$$\alpha = \operatorname{atan}(\hat{\mathbf{z}}'_x, \hat{\mathbf{z}}'_z)\,, \tag{3.2}$$

$$\beta = \operatorname{asin}\left(\hat{\mathbf{z}}'_y\right)\,, \tag{3.3}$$

$$\mathbf{A} = \mathbf{R}_{\mathbf{y}}(-\alpha)\begin{bmatrix}\hat{\mathbf{z}}' \\ \hat{\mathbf{x}}'\end{bmatrix}\,, \tag{3.4}$$

$$\mathbf{B} = \mathbf{R}_{\mathbf{x}}(\beta)\mathbf{A}\,, \tag{3.5}$$

$$\gamma = \operatorname{atan}(\mathbf{B}_{22}, \mathbf{B}_{21})\,, \tag{3.6}$$

$$\mathbf{N'}_i = \mathbf{R}_{\mathbf{z}}(-\gamma)\mathbf{R}_{\mathbf{x}}(\beta)\mathbf{R}_{\mathbf{y}}(-\alpha)(\mathbf{N}_i - \boldsymbol{O})\,. \tag{3.7}$$

Note that $\mathbf{R}_{\mathbf{x}}, \mathbf{R}_{\mathbf{y}}$ and $\mathbf{R}_{\mathbf{z}}$ are elemental rotation matrices (for more information see [31]) about axes $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ respectively.

After the transformation all points satisfy condition $z' = 0$. At this point algorithms for 2D triangular mesh generation can be applied. When a triangular mesh is computed, all nodes are transformed back to the global coordinate system

$$\mathbf{N}_i = \mathbf{R}_{\mathbf{y}}(\alpha)\mathbf{R}_{\mathbf{x}}(-\beta)\mathbf{R}_{\mathbf{z}}(\gamma)\mathbf{N'}_i - \boldsymbol{O}\,. \tag{3.8}$$

1D Mesh

It is necessary to create a 1D mesh of the domain boundary. This 1D mesh must also respect density function. It would be very computationally inefficient to divide curves into elements from their analytical prescription, therefore, the 1D mesh is created numerically.

Because every geometrical curve in AToM can by described by parameter $t \in [0, 1]$, it is reasonable to divide it into equidistant parts, which are later connected to elements of the desired size. This interval is subdivided into small pieces using function `linspace`.

First of all element size ρ from the density function is computed at the initial point $\mathbf{P}_1$ of the 1D element (this is a simplification of our problem, because it would be much more precise to compute size at the element centre, but this centre is unknown before we have actual 1D element). Then we continue adding up parameter t until our element length is $\|\mathbf{P}_1 - \mathbf{P}_2\| < \rho$, where $\mathbf{P}_2$ is a point on the curve with the parameter t.

Algorithm 4 Algorithm for line discretization

```
Discretize parameter t into selected number of equidistant parts
Compute length of the line (remainingLength)
Set initial parameter t_0 to 0
Compute local element size (localMeshSize) at t_0
while remainingLength > localMeshSize do
    Find part of t that length of segment t_{i-1}t_i is equal to localMeshSize
    remainingLength = remainingLength − localMeshSize
    Compute new localMeshSize at t_i
    if remainingLength < localMeshSize then
        Distribute remaining part of the line evenly between all edges
    end if
    Add new point to the set of points
end while
```

It is necessary to do smoothing of the line which is performed in the if statement, otherwise triangle patches are very likely to be created at the end of the curves. For dense meshes (A mesh with 10 and more elements on the boundary curve can be already considered as a dense because error created with this method is less than ten percent.) the deviation from ideal local element size is negligible.

2D Mesh

We started with a summary of necessary features for mesh generation in AToM. Since antenna can have an arbitrary shape, which it must take in note other constraints introduced in geometry, following list is a comprehensive

overview of all features which must be supported by the new mesh generation core:

- Create mesh above an arbitrary polygon in 2D.
- Support holes in polygons.
- Provide an interface for custom element size.
- Controllable mesh density with a user defined density function.
- Enable definition of fix nodes and fix edges for 1D and 2D feeding ports.
- Control correct distribution of elements on the intersection of connected planes.

Before mesh generator was reimplemented, it was using custom solution based on the DistMesh algorithm (Section 2.5.1). It was already using elements size but without density functions. It supported insertion of fix edges and fix points. Its main disadvantage was very slow convergence for larger meshes together with poor quality triangles.

We started with implementation of our own solution based on point insertion algorithms (Section 2.5.2). Insertion algorithms are able to generate meshes with exceptional quality [22] and adaptability. Unfortunately, it uses multiple tree structures which are very hard to implement efficiently in MATLAB. Since everything is represented as matrix in MATLAB, it is much more beneficial to use vectorization which is implicit parallelization of matrix operations.

We decided to use a modified Mesh2D algorithm (Section 2.5.3). This algorithm can generate meshes with user defined density functions together with arbitrary element size. It also supports connected polygons that triangles on the borderline are always conforming. It does not support fix edges and fix nodes. This functionality was necessary to add.

	Time (s)	q_{avg}	q_{min}
Original AToM solution	4.4063	0.8892	0.5551
Voronoi-vertex insertion	4,0839	0.9157	0.6523
Mesh2D	0.8906	0.9660	0.7569

Table 3.1: Comparison of mesh generation algorithms run in MATLAB R2015b. Benchmark mesh was a circle with approx. 2,000 triangles.

Element size for antenna discretization is directly connected with maximal frequency for numerical computation. Minimal number of triangles per wavelength is eight [32]. Therefore, it is possible to set number of elements per wavelength in AToM and element size is then computed according to the

following expression

$$\rho = n\frac{c_0}{f_{\max}}\,, \tag{3.9}$$

where n is a number of elements per wavelength, $f_{\max}$ is the maximal frequency for physical computation and c_0 is the speed of light.

After changing the frequency or density function, mesh must be invalidated and recomputed starting from the geometry discretization.

3.3 Mesh Utilities

It is assumed that not every user will be willing to use mesh generation core from AToM, yet he might want to use some basic computations above imported meshes. Such utilities are described in this section altogether with MATLAB code snippets.

3.3.1 Mesh Transformations

Not only we want to create mesh itself, but we want to use other geometrical transformations above existing mesh. Transformations are especially useful for generation of antenna arrays, *i.e.,* repetition of the same elements. It is computationally more efficient to copy antennas rather than run mesh generation process on the same geometry.

All following transformation are done in $\mathbb{R}^3$ since input for mesh generator in AToM are planar objects in three-dimensional space.

Translation

The first transformation is translation, which shifts every node $\mathbf{N}$ of a triangulation K by vector $\mathbf{t}$.

$$\mathbf{N}_{\text{new}} = \mathbf{N} + \mathbf{t}\,. \tag{3.10}$$

```
1 %% translate nodes
2 newNodes = bsxfun( @plus, nodes, shift );
```

Rotation

Another basic transformation is a rotation. It rotates mesh along standard basis vectors $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ in the right-handed Cartesian coordinate system. Rotation matrices are constructed and applied to coordinates of each node $\mathbf{N}$.

$$\mathbf{N}_{\text{new}} = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha)\mathbf{N}\,. \tag{3.11}$$

```
%% create rotation matrix
rotationMat = makehgtform('zrotate', zAngle) ...
    * makehgtform('yrotate', yAngle)  * ...
        makehgtform('xrotate', xAngle);

rotationMat = rotationMat( 1:3, 1:3 ); % homegenous ...
    coordinates are redundant
%% rotate every point
newNodes = nodes * rotationMat';
```

Scaling

Ratios and dimension of an object can be changed by an operation called scaling. It is done by three scalars s_1, s_2 and s_3 in directions of axes $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$. Special case when $s_1 = s_2 = s_3$ is called uniform scaling. Each node $\mathbf{N}$ is transformed as following

$$\mathbf{N}_{\text{new}} = \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{bmatrix} \mathbf{N}\,. \tag{3.12}$$

```
%% create scale matrix
scaleMat = makehgtform('scale', ratio);

scaleMat = scaleMat( 1:3, 1:3 ); % homegenous coordinates are ...
    redundant
%% scale every point
newNodes = nodes * scaleMat';
```

Reflection

Reflection is the only transformation which is not done as sum of vectors or by matrix multiplication. Input for this function is a triangulation (a set of *nodes* and *connectivityList*) altogether with the trinity of points defining a plane. All nodes are then replicated using the following transformation.

$$\hat{\mathbf{n}} = \frac{(\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1)}{\|(\mathbf{P}_2 - \mathbf{P}_1) \times (\mathbf{P}_3 - \mathbf{P}_1)\|}\,, \tag{3.13}$$

$$h = \hat{\mathbf{n}} \cdot (\mathbf{N} - \mathbf{P}_1)\,, \tag{3.14}$$

$$\mathbf{N}_{\text{ref}} = -2h\hat{\mathbf{n}} + \mathbf{N}\,, \tag{3.15}$$

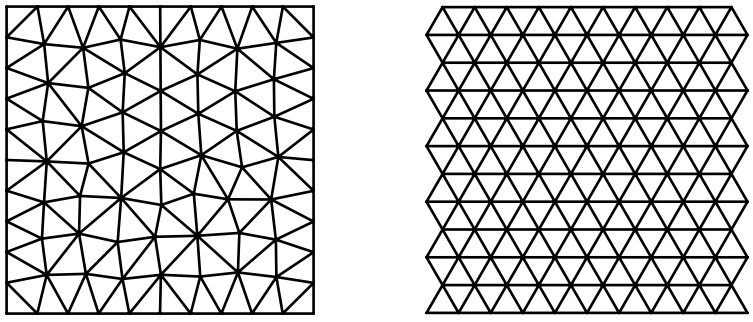
where $\hat{\mathbf{n}}$ is a normal of the plane passing through points $\mathbf{P}_1, \mathbf{P}_2$ and $\mathbf{P}_3$. $\mathbf{N}$ is a node and h is a distance of $\mathbf{N}$ to the reflection plane. The previous algorithm can be transformed to MATLAB code as following:

```
1 %% compute new point positions
2 u = points(2,:) - points(1,:);
3 v = points(3,:) - points(1,:);
4 normal = cross(u, v); % normal to plane given by vectors
5 normal = normal/norm(normal,2); % normalize normal
6 dist = normal*(bsxfun(@minus, nodes, points(1,:)))'; ...
      %distance to plane
7 newNodes = -2.*bsxfun(@times, dist', normal) + nodes; % shift ...
      points by -2*dist using normal vector
```

We have nodes with new positions but we are still missing connectivity for the new nodes and in addition we need to deal with a case when reflected nodes lie in the reflection plane. Since duplicate nodes (points which lie in the reflection plane) do not form valid triangulation, we firstly take those which are unique using the function `uniquetol`, which handles possible rounding errors, and then shift original connectivity by the number of nodes before the reflection and substitute positions of removed nodes with pointers into the original node set.

```
1 %% compute mesh connectivity
2 [newNodes, ~, ic] = uniquetol([nodes; newNodes], eps, ...
      'ByRows', true);
3 newConnectivityList = [connectivityList; ...
      connectivityList+length(nodes)];
4 newConnectivityList = ic(newConnectivityList);
```

3.3.2 Regular Uniform Triangulation



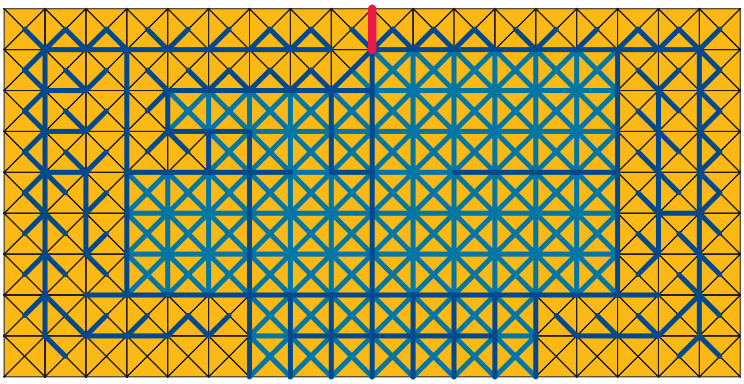
(a) A non-uniform mesh with $q_{\text{avg}} = 0.9373$.

(b) A uniform mesh from equilateral triangles and $q_{\text{avg}} = 1$.

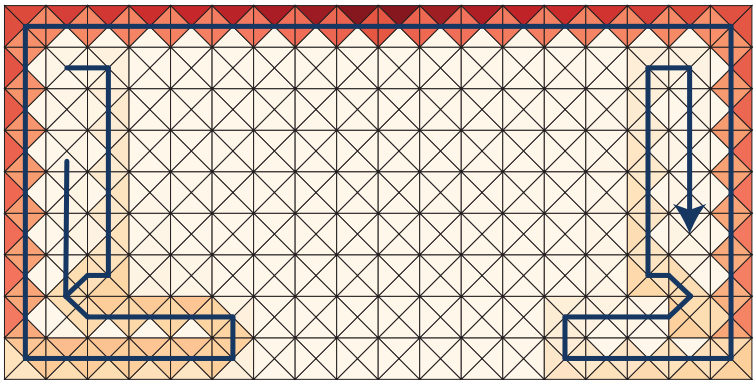
Figure 3.4: A discretized square.

Some applications require meshes of the same size constructed from regular elements laid in a chosen pattern. This means boundary conformity is sacri-

ficed for purposes of an optimisation task (Figure 3.4b). The structure is then optimized using either deterministic method, where the result is relatively regular, or heuristic algorithms like genetic algorithm NSGA-II [33], which uses flood filling to speed up computation.



(a) A structure with removed edges after optimization.



(b) Current flow in the optimized structure.

Figure 3.5: Antenna optimization with deterministic method using uniform triangulation.

These tasks are usually run above meshes consist of equilateral or rectangular triangles of the same size.

For general polygons, domain conformity is not usually kept. The regular underlying mesh is constructed from selected type of elements. Only those triangles which have all three vertices inside the domain are kept in the final triangulation.

3.4 Finding Longest "Loops" and "Dipoles"

Antenna theory distinguishes among many types of antennas. Two of them are dipole and loop antennas. While a dipole effectively separates electrical charge [34], it is topologically a path, a loop is capable of forming loop current, it is topologically a solenoid.

It is necessary for antenna optimisation to find such structures in triangular meshes because every discretized antenna can be decomposed into dipoles and loops. An undirected graph of RWG basis functions can be constructed above an existing mesh in order to find the optimal current in the structure [35]. This graph is created by connecting incenters of two neighbouring triangles. It is then possible to formalize loops and dipoles into fundamental graph structures - *a simple path* and *a simple cycle*.

Definition 1. *A **path** of length k from a vertex u to a vertex u' in an undirected graph $G = (V, E)$ is a sequence $\{v_0, v_1, v_2, \ldots, v_k\}$ of vertices such that $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in E$, $\forall i,\ i \in \{1, 2, \ldots, k\}$. The length of a path is a number of edges in the path. A path is **simple** if all vertices in the path are distinct.*

Definition 2. *A path $\{v_0, v_1, v_2, \ldots,\ v_k\}$ forms a **simple cycle** if $k \geq 3$, $v_0 = v_k$ and $v_0, v_1, v_2, \ldots, v_k$ are distinct.*

The main goal is to find all paths and cycles in a given graph and then find a structure with the best physical properties. This task is unfortunately computationally infeasible because it was proven that counting paths and cycles is #P-hard problem [36].

Instead of counting all paths by enumeration, a number of s-t paths (paths leading from the node s to the node t) can be estimated. This was experimentally tested with good results in [37]. This experiment also shows that number of s-t paths is extremely high even for very small sets of nodes. From the physical point of view only the longest (or a bit shorter than the longest) paths and cycles are interesting. It has been proven that finding the longest path in a graph is NP-hard problem [38].

Definition 3. *A **hamiltonian cycle** of an undirected graph $G = (V, E)$ is a simple cycle that contains each vertex in V.*

The longest simple cycle problem NP-completeness can be shown using Hamiltonian cycle for reduction.

Proof. Hamiltonian cycle problem is a special case of the longest simple cycle problem and it is NP-complete [38]. If and only if the longest simple cycle of the graph is as large as the number of nodes of the graph, then the graph has a Hamiltonian cycle. □

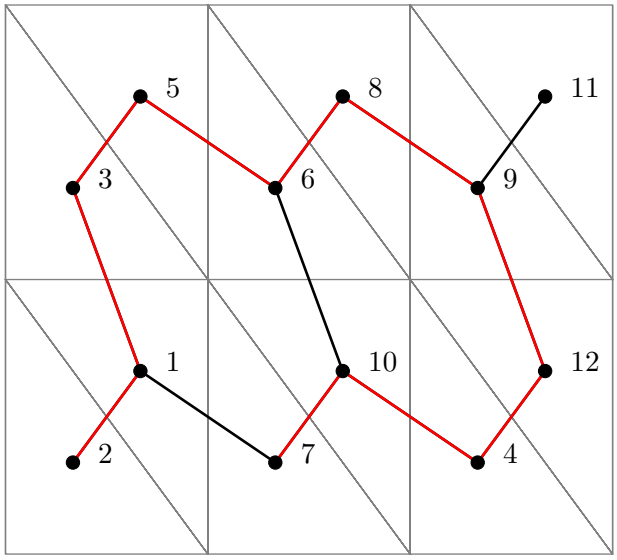


Figure 3.6: A path of the length 10 on a discretized dipole.

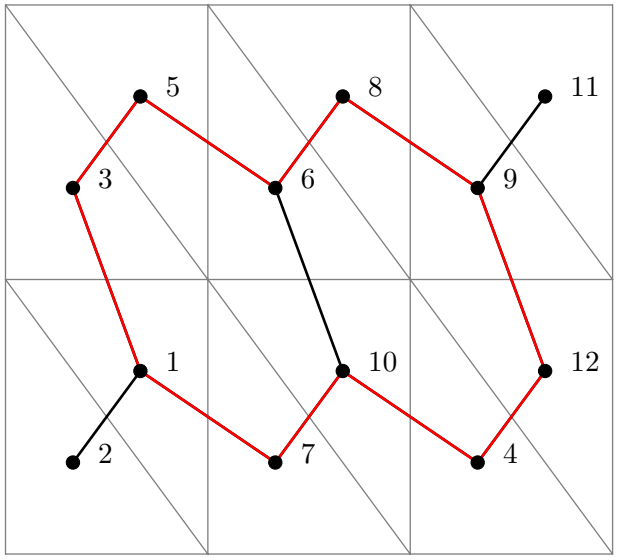


Figure 3.7: A cycle of the length 10 on a discretized dipole.

Since we cannot find path and cycles in a feasible time, we wrote our own naive algorithm for finding all paths and cycles in a graph G which can be used as proof of concept for physical methods.

Algorithm 5 Naive algorithm pseudocode for finding all paths and cycles

```
function findPathsAndCycles(G)
    while numEdges in G > 0 do
        Find arbitrary edge e
        Add e to path P
        Save P
        Remove e from G
        extendPath(v_0, v_k, P)
        Remove e from P
    end while
end function

function extendPath(v_0, v_k, P)
    Find all neighbouring nodes N of v_0
    for i = 1 to size(N) do
        if (v_0, N(i)) ∈ P then
            continue
        end if
        Add (v_0, N(i)) to P
        if isValid(P) then
            if isCycle(P) then
                Save P as cycle
            else if isPath(P) then
                Save P as path
                extendPath(N(i), v_k, P)
            end if
        end if
        Remove (v_0, N(i)) from P
    end for
end function
```

Conclusion

In this thesis, we summarized principles and basic algorithms for unstructured triangular mesh generation in two-dimensional spaces, which is typically encountered in numerical simulations and physical optimizations.

After thorough summary of mesh generation techniques like Delaunay triangulation and its constrained version, we focused on available algorithms. Firstly, we started with the implementation of our own solution, but it was soon clear that robust solution is not easy to implement from scratch.

The main goal of the thesis was to replace the mesh generation core in AToM to enhance performance and quality of generated meshes. We decided to use an existing solution called Mesh2D which, as stated in Chapter 3, outperforms our own and old mesh generation core in AToM by speed and overall triangle quality. It was necessary to consider all physical constraints like mesh size based on maximal frequency, boolean operations, the existence of symmetry planes, feeding positions, etc.

Implemented solution is able to discretize an arbitrary planar polygon in three-dimensional spaces, which can have any number of empty regions – holes. The user can set up his own density functions and change mesh size. It is also possible to generate symmetric meshes along planes created from standard basis vectors.

In addition to the changed mesh generation core, multiple mesh utilities, like the problem of finding the longest path and cycle in a graph were theoretically described and implemented.

Future Work

Development of the project AToM will continue until the end of 2017. It will be released into alpha and distributed to partners at the beginning of July 2017. It is expected that many bugs, which accompany this kind of big projects, will be discovered. It is also necessary to tweak communication between AToM GUI and themesh generator. An ordinary user will be presumably controlling

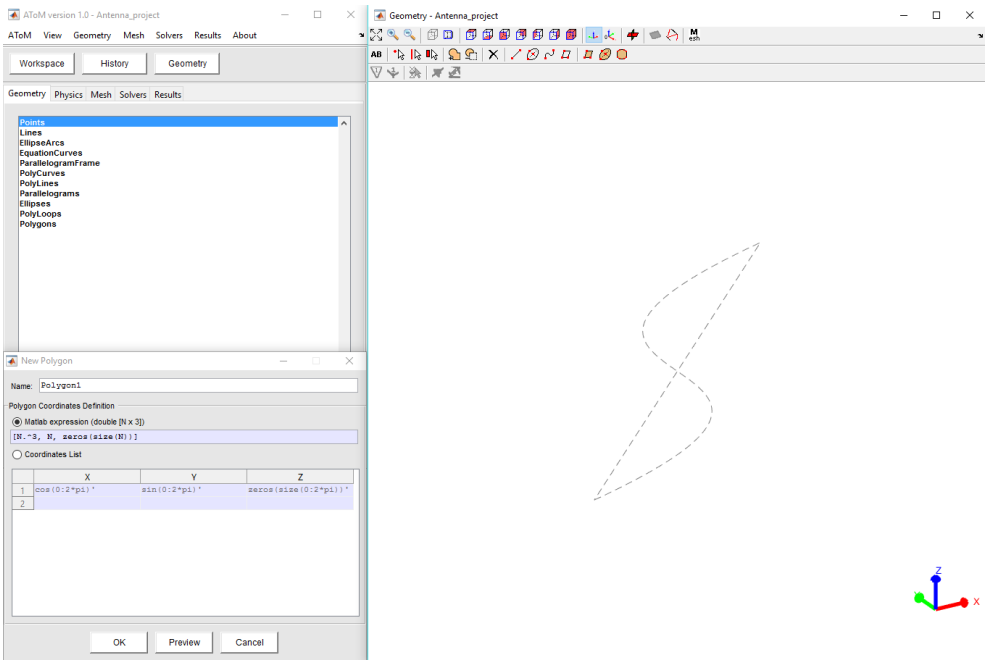


Figure 3.8: Geometry design in AToM.

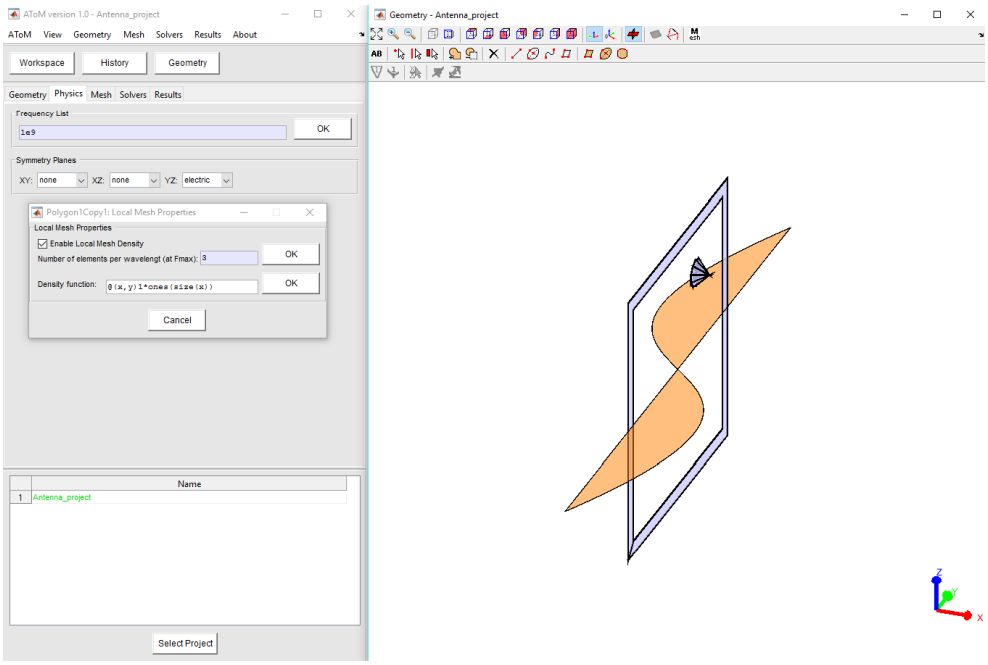


Figure 3.9: Mesh parameters set-up in AToM.

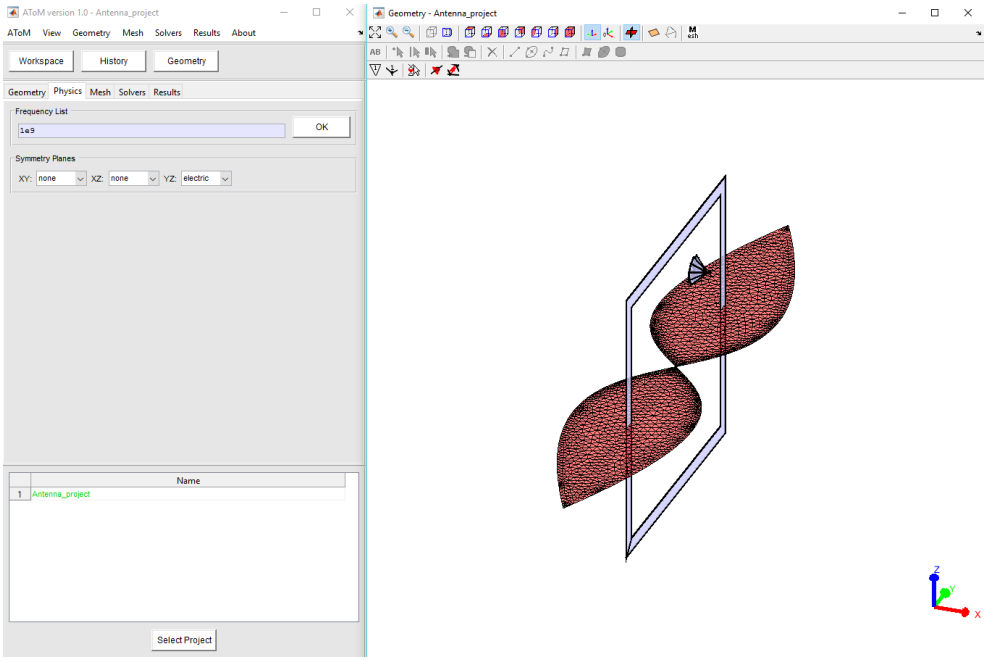


Figure 3.10: Generated mesh in AToM.

AToM through GUI, therefore it will be necessary to test dozens of use cases which can arise during the use of AToM. Other parts of AToM like Geom, GUI and MoM are also undergoing intensive development towards this date.

There are multiple things which remain to be done in mesh generator itself. A major feature which must be developed until the alpha release is *an intersection of planes*. Information about intersections must be delivered from Geom (this part is not ready yet) and then processed by mesh generator using Mesh2D functionality for connected planes and fix edges.

Another part which remains to be done is *a connection with BEM module*. It is necessary to provide a mesh together with metadata about types of edges (inner, outer), positions and dimensions of feeding ports, etc.

The last point is a feature called *local density refinement*. The main idea is that density function together with mesh size does not provide enough fine mesh in the areas of interest (*e.g.,* very small area around feeding ports). It is desired to change density locally in the selected area. It can be done using Voronoi insertion along with refining function provided in Mesh2D. The tricky part is to respect all existing constraints in the selected area. This must be implemented on GUI side (currently it is not possible to use drag-select to pick triangles) and in the mesh generator (insertion of new points and refinement).

Bibliography

[1] Czech Technical University in Prague, Department of Electromagnetic Field. *AToM - Antenna Toolbox for MATLAB [online]*. [cit. 2017-04-18]. Available from: `http://www.antennatoolbox.com/`

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

[3] Harrington, R. F.; Mautz, J. R. Theory of Characteristic Modes for Conducting Bodies. *IEEE Trans. Antennas Propag.*, volume 19, no. 5, September 1971: pp. 622–628, doi:10.1109/TAP.1971.1139999.

[4] Morse, P. M.; Feshbach, H. *Methods of Theoretical Physics*. McGraw-Hill, 1953.

[5] Edelsbrunner, H. *Geometry and Topology for Mesh Generation*. Cambridge University Press, 2001, ISBN 978-0-521-68207-7.

[6] Delaunay, B. Sur la sphère vide. A la mémoire de Georges Voronoı. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et na*, volume 6, 1934.

[7] ESI Group. *ESI [online]*. [cit. 2017-04-24]. Available from: `https://www.esi-group.com/`

[8] Aruoba, S. B.; Fernández-Villaverde, J. A Comparison of Programming Languages in Economics. *The National Bureau of Economic Research*, June 2014.

[9] Engwirda, D. *Locally Optimal Delaunay-refinement and Optimisation-based Mesh Generation*. Dissertation thesis, The University of Sydney, September 2014.

[10] Eichler, J.; Hazdra, P.; et al. Aspects of Mesh Generation for Characteristic-Mode Analysis. *IEEE Antennas and Propagation Magazine*, June 2014.

[11] Shewchuk, J. R. Lecture Notes on Delaunay Mesh Generation. [online], 2012. Available from: `https://people.eecs.berkeley.edu/~jrs/meshpapers/delnotes.pdf`

[12] HyperWorks. *FEKO [online].* [cit. 2017-05-06]. Available from: `https://www.feko.info/`

[13] ESI Group. *CEM One [online].* [cit. 2017-05-06]. Available from: `https://www.esi-group.com/software-solutions/virtual-environment/electromagnetics/cem-one-solution`

[14] MathWorks. *N-D Delaunay triangulation [online].* [cit. 2017-04-18]. Available from: `https://www.mathworks.com/help/matlab/ref/delaunayn.html`

[15] Chew, L. P. Constrained Delaunay Triangulations. *Algorithmica*, volume 4, June 1989.

[16] Sibson, R. Locally equiangular triangulations. *The Computer Journal*, volume 21, 1978.

[17] Shewchuk, J. R. What Is a Good Linear Finite Element? - Interpolation, Conditioning, Anisotropy, and Quality Measures. Technical report, In Proc. of the 11th International Meshing Roundtable, 2002.

[18] Bhatia, R. P.; Lawrence, K. L. Two-Dimensional Finite Element Mesh Generation Based on Stripwise Automatic Triangulation. *Computers and Structures*, volume 36, 1990.

[19] Persson, P.-O.; Strang, G. A Simple Mesh Generator in MATLAB. *SIAM Review*, volume 46, June 2004.

[20] Barker, T. J. Three dimensional mesh generation by triangulation of arbitary point sets. Technical report, 8th Computational Fluid Dynamics Conference, 1987.

[21] Bowyer, A. Computing Dirichlet Tessellations. *The Computer Journal*, volume 24, February 1981.

[22] Rebay, S. Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm. *Journal of Computational Physics*, volume 106, 1993.

[23] Darren Engwirda. *MESH2D [online].* [cit. 2017-04-29]. Available from: https://www.mathworks.com/matlabcentral/fileexchange/25555-mesh2d-delaunay-based-unstructured-mesh-generation

[24] Engwirda, D. *MESH2D 3.0.0 [online].* [cit. 2017-05-02]. Available from: https://github.com/dengwirda/mesh2d

[25] MathWorks. *Triangulation in 2-D or 3-D [online].* [cit. 2017-04-18]. Available from: https://www.mathworks.com/help/matlab/ref/triangulation-class.html

[26] MathWorks. *Delaunay triangulation in 2-D and 3-D [online].* [cit. 2017-04-18]. Available from: https://www.mathworks.com/help/matlab/ref/delaunaytriangulation-class.html

[27] MathWorks. *2-D triangular plot [online].* [cit. 2017-04-18]. Available from: https://www.mathworks.com/help/matlab/ref/triplot.html

[28] MathWorks. *Triangular mesh plot [online].* [cit. 2017-04-18]. Available from: https://www.mathworks.com/help/matlab/ref/trimesh.html

[29] MathWorks. *Triangular surface plot [online].* [cit. 2017-04-18]. Available from: https://www.mathworks.com/help/matlab/ref/trisurf.html

[30] Rao, S. M.; Wilton, D. R.; et al. Electromagnetic Scattering by Surfaces of Arbitrary Shape. *IEEE Trans. Antennas Propag.*, volume 30, no. 3, May 1982: pp. 409–418, doi:10.1109/TAP.1982.1142818.

[31] Lay, D. C. *Linear Algebra and Its Applications 4th ed.* Addison-Wesley, 2012.

[32] Peterson, A. F.; Ray, S. L.; et al. *Computational Methods for Electromagnetics.* Wiley – IEEE Press, 1998.

[33] Deb, K. *Multi-Objective Optimization using Evolutionary Algorithms.* Wiley, 2001.

[34] Balanis, C. A. *Antenna Theory Analysis and Design.* Wiley, third edition, 2005.

[35] Capek, M.; Jelinek, L.; Kadlec, P.; Strambach M. *Excitation of Optimal and Suboptimal Currents.* EuCAP 2017, Paris, France, 2017.

[36] Yamamoto, M. Approximately counting paths and cycles in a graph. *Discrete Applied Mathematics*, volume 217, 2017.

[37] Roberts, B.; Kroese, D. P. Estimating the Number of s-t Paths in a Graph. *Journal of Graph Algorithms and Applications*, volume 11, 2007.

[38] Schrijver, A. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer-Verlag Berlin Heidelberg New York, 2003, ISBN 3-540-44389-4.

Appendix A

Content of the Enclosed CD

- `README.pdf` ..... CD content description
- `AToM` ..... directory with implementation
  - `+models`
    - `+mesh`
      - `@Mesh` ..... class `Mesh` implementation
      - `@Mesh_2D` ..... class `Mesh2D` implementation
      - `@MeshObject` ..... class `MeshObject` implementation
    - `+utilities`
      - `+meshPublic` ..... mesh utility implementation
  - `examples`
    - `mesh` ..... directory with mesh examples
- `topology` ..... implementation of topology optimization
- `insertionAlgorithms` .. implementation of point insertion algorithms
- `text`
  - `BP_Strambach_Martin_2017.pdf` ..... text of the thesis in PDF
  - `thesis` ..... GIT repository of the thesis