Universidade de São Paulo
Instituto de Física

Otimização da distribuição de fluidos em meios porosos usando padrões de venações de folhas

Caio Martins Ramos de Oliveira

Orientador: Prof. Dr. Adriano Mesquita Alencar

Dissertação de mestrado apresentada ao Instituto de Física para a obtenção do título de Mestre em Ciências

Banca Examinadora:

Prof. Dr. Adriano Mesquita Alencar (IF-USP)
Prof. Dr. Claudimir Lucio do Lago (IQ-USP)
Prof. Dr. Caetano Rodrigues Miranda (IF-USP)

São Paulo
2017

FICHA CATALOGRÁFICA
Preparada pelo Serviço de Biblioteca e Informação
do Instituto de Física da Universidade de São Paulo

Oliveira, Caio Martins Ramos de

Otimização da distribuição de fluídos em meios porosos usando padrões de venação de folhas. São Paulo, 2017.

Dissertação (Mestrado) – Universidade de São Paulo. Instituto de Física. Depto. de Física Geral

Orientador: Prof. Dr. Adriano Mesquita Alencar

Área de Concentração: Física

Unitermos: 1. Mecânica dos fluídos; 2. Dinâmica dos fluídos computacional; 3. Venação de folhas; 4. Rede de canais.

USP/IF/SBI-027/2017

Universidade de São Paulo
Instituto de Física

Fluid distribution optimization in porous media using leaf venation patterns

Caio Martins Ramos de Oliveira

Supervisor: Prof. Dr. Adriano Mesquita Alencar

Master's dissertation presented to Instituto de Física to obtain the Master of Science degree

Master's Committee:

Prof. Dr. Adriano Mesquita Alencar (IF-USP)
Prof. Dr. Claudimir Lucio do Lago (IQ-USP)
Prof. Dr. Caetano Rodrigues Miranda (IF-USP)

São Paulo
2017

"...our small planet, at this moment, here we face a critical branch-point in the history. What we do with our world, right now, will propagate down through the centuries and powerfully affect the destiny of our descendants. It is well within our power to destroy our civilization, and perhaps our species as well. If we capitulate to superstition, or greed, or stupidity we can plunge our world into a darkness deeper than time between the collapse of classical civilization and the Italian Renaissance. But, we are also capable of using our compassion and our intelligence, our technology and our wealth, to make an abundant and meaningful life for every inhabitant of this planet. To enhance enormously our understanding of the Universe, and to carry us to the stars."
-Carl Sagan

Acknowledgments

I would like to thank all lab members for creating a particularly stimulating work evironment. Their feedback and cooperation was very helpful and much appreciated.

I would like to thank Juan for giving me a hand with the experiments and for being such a helpful, generous and kind person.

I would like to thank prof. Murilo for the valuable OpenFOAM-related suggestions he gave me.

I would like thank my aunt, Mary Christine, and a dear friend, Pedro, for correcting some of my most awful English mistakes.

I would like to thank all my friends and everyone from Yoga ao ar livre group who supported throughout the writing of this dissertation. All of them greatly contributed to my emotional and spiritual devolopment.

I would like to thank Elliot Hulse for sharing his life experience in his awesome videos. They have certainly helped me become a stronger version of myself.

I would like to thank my supervisor, Adriano, for the patience and encouragement he demonstrated in all of our conversations, as well as the invaluable insights he always provided.

I would like thank my parents, Nilton and Eneida, for the help and love they have always offered me. Their encouragement allowed me to pursue my truth.

I also would like to thank my wife, Juliana, for supporting me and encouraging me in every way even during my darkest hours.

Last but not the least, I would like to thank me, for having the self-love, the self-respect and the courage to keep on fighting. I am grateful for believing that I could still succeed even when the odds were against me.

Done.

Resumo

Diversos exemplos de redes de transporte quase ótimas podem ser encontradas na natureza. Essas redes distribuem e coletam fluidos através de um meio. Evidências sugerem que os vasos sanguíneos do sistema circulatório, as vias respiratórias nos pulmões e as veias das venações em folhas são exemplares de redes que evoluiram para se tornarem efetivas em suas tarefas sendo, ao mesmo tempo, eficientes energeticamente. Dessa forma, não chega a ser surpreendente que recentes melhorias de performance em dispositivos de geração de energia modernos ocorrem devido ao uso de arquiteturas de canais inspiradas na natureza. Guiados por estas observações, nesse trabalho, investigamos a aplicação de padrões de venações verossímeis geradas por computador em um tipo de dispositivo fotovoltaico. Resolvemos o problema de escoamento através do dispositivo usando ferramentas de Dinâmica de Fluidos Computacional (CFD). Além disso, procuramos desenvolver modelos experimentais. Em última instância, estamos em busca das propriedades da rede que afetam sua performance.

Palavras-chave: Mecânica dos fluidos, Dinâmica dos fluidos computacional, venação de folhas, redes de canais

Abstract

Several examples of nearly optimal transport networks can be found in nature. These networks effectively distribute and drain fluids throughout a medium. Evidence suggests that blood vessels of the circulatory system, airways in the lungs and veins of leaf venations are examples of networks that have evolved to become effective in their tasks while simultaneously being energy efficient. Hence, it does not come as a surprise that recent performance improvements of modern power generating devices occur due to the use of nature-inspired channel architectures. Guided by this observations, in this work, we investigate the application of visually realistic computer-generated leaf venation patterns to a type of photovoltaic device. We solve the flow through the device problem using Computational Fluid Dynamics (CFD) tools. Moreover, we attempt to develop experimentals models. Ultimately, we seek to single out the network properties that affect their performance.

Keywords: Fluid mechanics, Computational fluid dynamics, leaf venations, channel networks

Nomenclature

ADI	Alternating direction implicit
BC	Boundary condition
BST	Binary search tree
CAD	Computer aided design
CFD	Computational Fluid Dynamics
CV	Control volume
DCEL	Doubly-connected edge list
DNS	Direct numerical simulation
DSSC	Dye-Sensitized Solar Cell
emf	electromotive force
FCS	Fluorescence correlation spectroscopy
FCV	Fuel cell vehicle
FV	Finite volume
GUI	Graphical user interface
SOR	Successive over-relaxation
IC	Initial condition
m-FGPV	Microfluidic gel photovoltaics
NMR	Nuclear magnetic ressonance
NN search	Nearest neighbor search
NS equations	Navier-Stokes equations
PDE	Partial differential equation

PEMFC	Proton-membrane exchange fuel cell
PIMPLE	merged PISO-SIMPLE
PISO	Pressure Implicit with Splitting of Operator
PLA	Polylactic acid
REV	Representative elementary volume
RNG	Relative neighborhood graph
SIMPLE	Semi-Implicit Method for Pressure Linked Equations
STL	STereoLithography
VLA	Vein length per area

Contents

1. Introduction

1.1. Outline

The search for efficient transport networks has been the subject of multiple studies across numerous fields [1–3]. The high degree of interest stems partially from the need to minimize the cost of transport in a large number of human-designed distribution networks while maintaining, or perhaps improving, distribution efficacy. Highways, roads, railroads, electric power supply, water distribution and drainage networks are all instances that could benefit from transport optimization [1, 4–6]. Nature provides abundant examples of irrigation and drainage systems ranging from the cardiovascular's arteriovenous and lymphatic systems to the respiratory system involving the lungs [1, 7–12]. These natural transport networks are similar to the transport networks designed and constructed by humans. In addition to serving an analogous purpose, they are also subject to energy saving requirements and topological constraints [1, 13]. The natural networks, however, are a product of evolution, which has been solving the optimization problem for millions of years. Therefore, it is not unreasonable to suppose that these networks, found in nature, are highly optimal in their tasks [2, 7, 8].

Pursuing the validity of this reasoning, in this work, we make an attempt to optimize the transport by employing channel networks with similar structure and properties to the ones found in nature, in a procedure known as *biomimetics* [8, 14, 15]. In particular, we generate networks inspired in leaf venation patterns using a computer algorithm and later estimate their performance. By doing this, we focus not only on optimizing the properties of each individual network, but also on identifying the venation-inpired networks with optimal *geometrical* traits. This approach is in contrast to the optimization problem of fixing a geometry and varying the properties of the network, a problem which has already been extensively treated in the literature and, in graph theory, goes by the name of Minimum Concave Cost Flows [1, 16, 17]. Therefore, observe that the problem we engage in is still quite open [6], as the configuration of the network itself is also at stake.

We have already listed some possible targets that would benefit from an improvement in transport performance. Although different optimal networks may be governed by similar underlying principles [1, 7], notice that their performance ultimately depends on the particular target. That is because each target has its own peculiarities. This may influence what is estimated to be a good performance. Hence, a reasonable approach for an empirical study aiming to identify optimal transport networks is to

have the networks be designed for the applications of interest. Generally, experience and intuition are used to design the networks [6]. In our case, we set out aiming to increase the performance of a particular application, finding inspiration in leaf venations, at least at a fundamental level. Ultimately, we seek attributes of the geometries which correlate strongly with their performance. Once our understanding grows, we hope to be able to optimize the target's performance [18].

The generated venation-inspired configurations are presented in Chapter 2, including a description of how we produced them. Subsequently, in Chapter 3, we detail how we estimated the geometry performance using *Computational Fluid Dynamics* (CFD), also offering a statistical analysis of the results. In Chapter 4, we showcase the efforts that went into producing an experimental model that can validate the CFD results. A flowchart illustrating the steps described in these chapters is presented in Fig. 1.1. Finally, in Chapter 5, we lay out some of our plans for future work, as later, with a more systematized knowledge about the properties that make a network efficient for the targets, we hope to gain insight on how to improve its performance. A thorough discussion on the implemented algorithms and data structures that played a supporting role in leaf venation generation is provided in Appendix A, while in Appendix B the CFD tools used to solve the governing equations are presented.

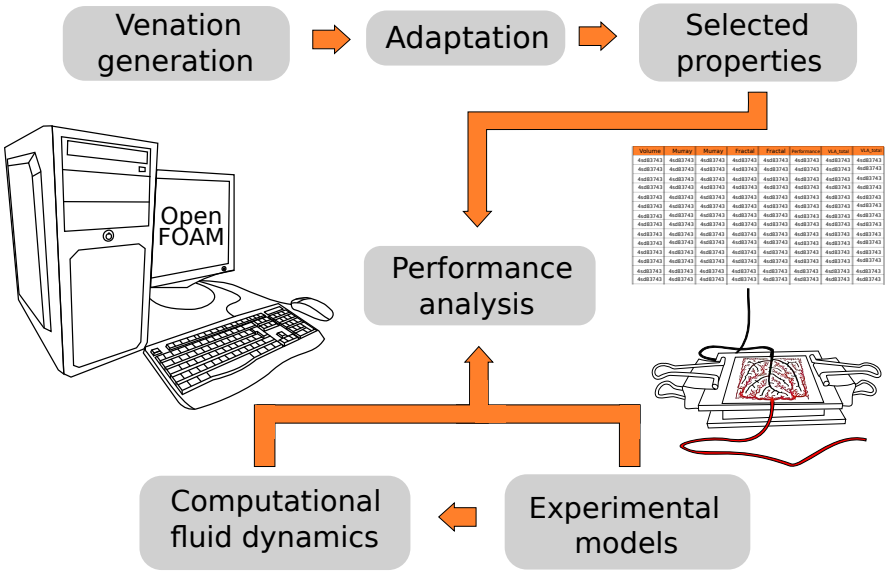

Figure 1.1.: Flowchart describing the methodology employed to reach the objectives of this work.

In the remainder of this chapter, we introduce topics which are relevant to the discussion in the upcoming chapters. Next, we present some possible applications of our work.

1.2. Target applications

As previously stated in the last section, optimal design of channel architectures is application-dependent. Hence, as we focused on improving geometries which are venation-inspired, we narrowed down the targets to the ones which have at least one component with properties that are similar to leaves. In particular, leaves, may be regarded, to some degree, as a bidimensional porous medium with a channel network embedded on it to distribute water and nutrients [19–21]. Taking this into account, we were able to select two targets for the venation-like geometries. Next, we introduce the target applications, present their features and discuss how including a venation-inspired channel network might be beneficial to them. For now, we only state that, for both applications, there is a need for the inclusion of an additional complementary geometry to drain the solvent. A discussion clarifying how we accomplished this as well as the reason why leaves do not require the additional geometry will be offered in Chapter 2.

1.2.1. Dye-sensitized solar cell variant

The first application, the one that in fact motivated this work, is a variant of dye-sensitized solar cells (DSSCs). DSSCs are a third generation type of solar cell that have the potential to be both cheap and efficient [22]. In DSSCs, photons are absorbed by dye molecules, as opposed to the band gap excitation of electrons mechanism in standard semiconductor solar cells. The excited dye molecules then transfer the charges to the semiconductor TiO_2 nanoparticles on which they are adsorbed. The charges are then transported to the anode. From there, they make their way along the circuit until they reach the cathode. The circuit is completed with the aid of an electrolyte solution which absorb the electrons and restore the dye molecules to their ground state [22, 23]. As long as the device is exposed to light, it will produce an electromotive force (emf).

In the variant we mentioned a method was devised to regenerate the cells after dye degradation, which can happen after long-term UV exposure [22], for instance. The degraded dye is then collected and replenished with new dye. An effective way to distribute the new dye across the device was introduced by inserting an agarose hydrogel slab containg the electrolyte solution between the electrodes [24], see Fig. 1.2a. In order to facilitate the dye molecule distribution, allowing for them reach and be adsorbed by the TiO_2 nanoparticles, a channel network was formed on one of the hydrogel slab faces. Therefore, in theory, optimizing the channel networks present in the hydrogel porous medium could improve the regeneration process of the device. This fact makes this variant of DSSCs, named μ-fluidic gel photovoltaics (μ-FGPVs) [24], the primary target of our study for reasons that will be discussed in Chapter 3.

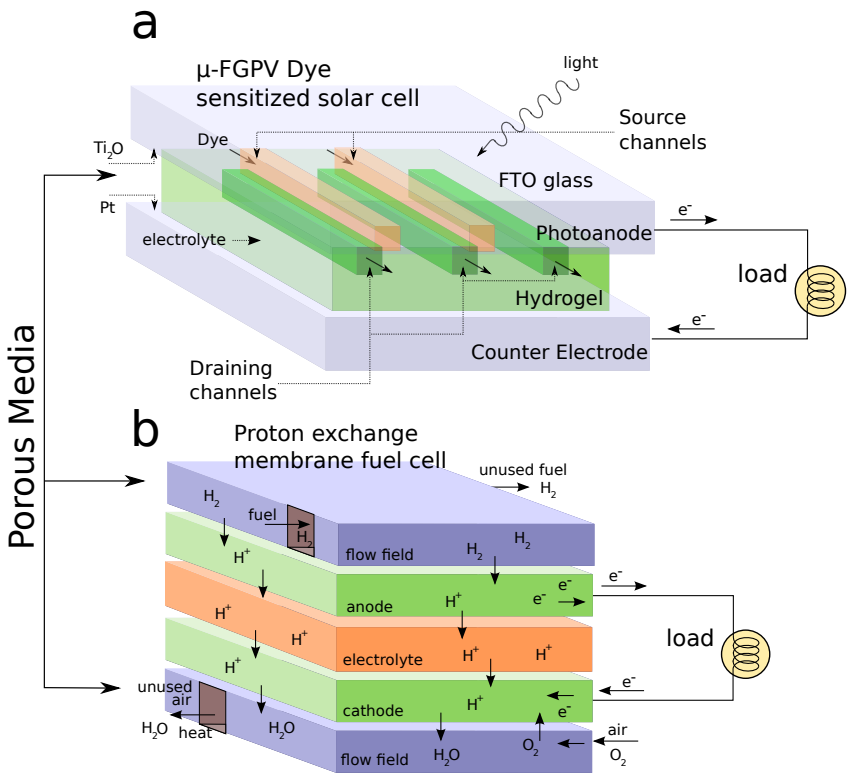

Figure 1.2.: Devices that employ channel networks embedded in a porous medium. Both devices are presented as solutions to the current need for greener power supply demand. (a) is a sketch of the DSSC variant, the μ-FGPVs [24]. This arrangement includes a hydrogel layer, which is used in the distribution of dye molecules. The molecules are adsorbed onto the surface of the Ti_2O nanoparticles. This design enables solar cell regeneration after the dye degradation. (b) is a diagram of a proton exchange membrane fuel cell (PEMFC) [25]. Fuel cell vehicles (FCVs), which make use of this technology, are already a reality [26].

1.2.2. Proton-Exchange Membrane Fuel Cells

A second even more promissing target of the bidimensional venation-inspired geometries was envisioned later, after a more systematic literature seach. Proton-exchange membrane fuel cells (PEMFCs) are devices that convert the chemical energy of a fuel into a *emf* [25]. The fuel cells are similar to batteries, but require a continuous supply of fuel, typically hidrogen, to produce the *emf.* PEMFCs, in particular, have a proton-conducting polymer membrane containing electrolyte confined between the electrodes. Finally, attached to both the anode and cathode are two porous flow field plates with bidimensional channel networks carved onto them, see Fig. 1.2b. It has been shown that improvements on the arquitecture of the channels may enhance distribution and collection of reactants across the electrodes ultimately increasing cell performance [18, 27–29]. This makes PEMFCs an excellent target of our study.

We will cover more details about PEMFCs in Chapter 2.

1.3. Partial differential equations classification

Throughout upcoming chapters, in particular Chapter 3, we will encounter many partial differential equations (PDEs). These equations govern the flow through the target systems. Therefore, understanding the behavior of the PDEs and knowing the boundary and initial conditions which are appropriate to each system will be of great importance during the forecoming discussion.

PDEs may be classified into three types: elliptic, parabolic or hyperbolic equations [30–33]. While parabolic and hyperbolic PDEs are associated with marching problems, that is, problems where the solution varies in time, elliptic PDEs describe equilibrium problems. The canonical examples for the elliptic, parabolic and hyperbolic PDEs are, respectively, the Laplace, the diffusion and the wave equations. The greatest difference between each of the PDE types are the boundary conditions (BCs) as well as the initial conditions (ICs) required for a unique and stable solution. The properties of each PDE category is summarized in Tab. 1.1 [30, 33]. As a last remark, we stress that hyperbolic equations possess an additional trait, which is the limited speed with which the solution propagates throughout the system. This has to do with both real characteristics defining the domain of dependence of the solutions [31, 33]. For further details on this topic, the reader may refer to Arfken or other CFD textbooks [30–33]. For now, we present a method used to classify the second-order PDEs.

Table 1.1.: Table classifying PDE types according to the problem type, boundary conditions and solution domain.

Equation type	Problem type	Conditions	Solution domain
Hyperbolic	Marching	Cauchy BCs	Open
Parabolic	Marching	Dirichlet/Neumann BCs	Open
Elliptic	Equilibrium	Dirichlet/Neumann BCs	Closed

In two dimensions, the most general second-oder PDE for a scalar field ϕ can be written as:

$$a\frac{\partial^2\phi}{\partial x^2} + b\frac{\partial^2\phi}{\partial x\partial y} + c\frac{\partial^2\phi}{\partial y^2} + d\frac{\partial\phi}{\partial x} + e\frac{\partial\phi}{\partial y} + f\phi + g = 0 \tag{1.1}$$

Classification PDE can be performed by evaluating the following discriminant D and by verifying Tab. 1.2:

$$D = b^2 - 4ac \tag{1.2}$$

Table 1.2.: Second-order PDE classification in 2D [33].

D	Equation type	Characteristics
> 0	Hyperbolic	Two real characteristics
$= 0$	Parabolic	One real characteristic
< 0	Elliptic	No characterics

Classification of the PDEs is possible by considering the value of D makes and its relation with the charcteristic equation:

$$a\left(\frac{dy}{dx}\right)^2 - b\left(\frac{dy}{dx}\right) + c = 0 \tag{1.3}$$

In case D is positive, the characteristic equation has two real roots and the solution ϕ exhibit a simple wave form, a typical hyperbolic trait [33]. Additional aspects are summarized in Tab. 1.2. Moreover, notice that it is possible to reduce the more complex PDEs to the respective canonical form through a change of variables, that is, one can reduce a complex hyperbolic equation to a wave equation. Finally, even if the coefficients a, b, c, etc. depend on the variables x and y or possibly on the solution ϕ itself, the PDEs may still be classified using this approach. In this complex case, however, the behavior of the PDE is local and may change throughout the solution domain, exhibiting a parabolic behavior in a certain region, while showcasing a hyperbolic behavior in another.

If the second-order PDE depends on N variables, then Eq. 1.1 becomes:

$$\sum_{j=1}^{N}\sum_{k=1}^{N} A_{jk}\frac{\partial^2\phi}{\partial x_j \partial x_k} + H = 0 \tag{1.4}$$

where $A_{jk} = A_{kj}$ and H accounts for the other terms [33, 34]. Classification of the PDEs using this matrix approach then requires the evaluation of the eigenvalues λ of the matrix with elements A_{jk}. When at least one eigenvalue λ is null, the PDE is parabolic. When all eigenvalues are of the same sign and non-zero, the PDE is elliptic. Finally, when all eigenvalues but one are of the same sign, with all of them being non-zero, the PDE is hyperbolic.

1.3.1. Transport equation

The transport equation or convective-diffusion equation is one of the PDEs we will encounter throughout this work. It can be written as:

$$\frac{\partial\phi}{\partial t} = D\boldsymbol{\nabla^2}\phi - \boldsymbol{u}\cdot\boldsymbol{\nabla}\phi \tag{1.5}$$

where ϕ is a scalar field representing the reactant concentration, D is the diffusion coefficient of the medium and $\boldsymbol{u}$ is the velocity field. We observe by examining Eq. 1.5 that putting aside the second term on the right side, we retrieve the diffusion equation, a parabolic PDE. The convective term, $\boldsymbol{v} \cdot \boldsymbol{\nabla}\phi$, on the other hand, has a hyperbolic nature. This can easily be verified, for intance, by factoring the second-order one dimensional wave equation, a hyperbolic equation, into two first-order equations [30, 35]. A more detailed discussion on the transport equation in the context of CFD can be found in Sec. B.3.

1.3.2. Navier-Stokes equations

The Navier-Stokes equations is a set of non-linear equations describing momentum conservation for a control volume. In this work, we will use the incompressible Navier-Stokes equations, which may be expressed in the following form:

$$\rho\frac{\partial \boldsymbol{u}}{\partial t} + \rho\boldsymbol{u} \cdot \boldsymbol{\nabla}\boldsymbol{u} = -\boldsymbol{\nabla}p + \mu\boldsymbol{\nabla^2}\boldsymbol{u} + \rho\boldsymbol{g} \tag{1.6}$$

where $\boldsymbol{u}$ is the velocity field, ρ is the fluid density, p is the pressure, μ is the dynamic viscosity of the fluid and $\boldsymbol{g}$ is the gravity acceleration. A thorough discussion on Eq. 1.6, how derivate it, and additional properties such as flow types may be found in Sec. B.2.

We realize, by examining Eq. 1.6, that classifying the Navier Stokes is not straighforward at all. Nevertheless, it can be done using the matrix approach described earlier in this section, see Eq. 1.4. Classification of the behavior of the Navier-Stokes equations has been made according to the type of flow and the mach number M, the ratio between the flow velocity $\boldsymbol{u}$ and the speed of sound of the medium, see Tab. 1.3. Further details on how the classification process is performed are out of the scope of this text, but can be encountered in the references [33, 34].

Table 1.3.: Outline of the classification of the Navier-Stokes equations based on the flow type and mach number M.

	Steady flow	Unsteady flow
Vicous flow	Elliptic	Parabolic
Inviscid flow	$M < 1$ Elliptic $M > 1$ Hyperbolic	Hyperbolic
Thin shear layers	Parabolic	Parabolic

1.4. Statistical concepts

In this section we briefly present some basic statistical concepts which will be useful when we discuss the results in Chapter 3.

1.4.1. Terminology and basic concepts

Here, we define the basic concepts and terminology which must be considered when evaluating the results later on. We start by making a distinction between the term population and sample. A population, in general, is a large collection of similar items we seek information about. A sample, on the other hand, is representative a group of items taken from the population. In this context, the information describing the entire population, usually provided in the form of summary numbers, e.g., the mean, variance, etc., are called parameters, while the summary numbers of the samples are called statistics. In practice, the parameters are desired, but not known, so one usually seeks to infer information about the parameters using statitics. There are at least two forms of learning about the parameters in this fashion: hypothesis testing and confidence intervals [36, 37]. We discuss more about them later in this section. For now, we present the relations for some of the fundamental summary numbers, which are useful when describing the data attributes later on.

Commonly used summary numbers when one is dealing with the a single attribute are the mean $\overline{x}$ and the standad deviation σ. The mean of a random variable x representing an attribute can be calculated by:

$$\overline{x} = \frac{1}{N}\sum_{i}^{N} x_i \tag{1.7}$$

where x_i are the sampled values of the attribute, that is, the measurements and N is the number of measurements. The standard deviation of x, in turn, is given by:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i}^{N}\left(x_i - \overline{x}\right)^2} \tag{1.8}$$

Other important summary numbers when more than one random variable is being considered are the covariance and correlation coefficients. These are important because they reveal whether two attributes exhibit a linar relationship or not. The covariance between the random variables x and y is of the form:

$$\mathrm{cov}(x, y) = \frac{1}{N}\sum_{i}^{N}\left(x_i - \overline{x}\right)\left(y_i - \overline{y}\right) \tag{1.9}$$

The closer the covariance is to being null, the weaker is the linear relationship between the two attributes. However, if the attributes have two different scales, it is possible that a weak linear relationship may yield a high covariance. This problem is solved by dividing Eq. 1.9 by the standard deviation of the of the random variables x and y, which yields:

$$\mathrm{corr}(x, y) = \frac{\mathrm{cov}(x, y)}{\sigma_x \sigma_y} \tag{1.10}$$

where $\mathrm{corr}(x, y)$ is known as the Pearson correlation coefficient and σ_x and σ_y are the standard deviation of the random variables x and y respectively. This correlation coefficient varies from -1 to 1 [36–38]. When $\mathrm{corr}(x, y)$ is close to 1, there is a positive linear relationship between the attributes, or a positive correlation. When it is close to -1, there is a negative linear relationship, or anticorrelation. At last, when it is close to 0, there is not a linear relationship between the variables, and, thus, no correlation. In Chapter 3, the Pearson correlation coefficient will be extensively employed when we present some correlograms of the constructed data base.

Finally, when one is seeking to display the data without making any assumptions about the population parameters, such as the summary numbers above, it is possible to use boxplots, see Fig. 3.16. The median, the second quartile Q_2, is represented by the band inside the box. The lower and upper region of the box are defined by the first Q_1 and third quartile Q_3 of the data set. Quartiles are the three points that divide the data set in four equal parts. The standard definition for the boxplot whiskers cut-offs is given by the following relation:

$$\text{upper whisker}: \quad Q_3 + 1.5IQR$$

$$\text{lower whisker}: \quad Q_1 - 1.5IQR$$

where,

$$IQR = Q_3 - Q_1$$

Boxplots defined this way are also known as Tuckey boxplots [39]. Outliers are depicted as dots in the region above or below the whiskers.

1.4.2. Hypothesis testing

Hypothesis testing is one of the most common forms of learning about the parameters of a populations from the statistics of a sample. It consists in making an initial assumption, the null hypothesis H_0, collecting data to test the hypothesis and then rejecting H_0 or not based on the evidence in favor of an alternative hypothesis. In practice, the method used is usually the p-value approach [36–38]. This hypothesis testing method is coupled with a test statistic, for instance, the t-value, used in t-tests.

The t-value, the test statistic used in this work, is a scalar function t of the observations. Its calculation depends on the type of test being made. In a one-sample t-test we test whether a population mean has a value defined in H_0. In a two-sample t-test, we test whether the means of two populations are equal, H_0, or not, the alternative

hypothesis. The t-value relation for the latter case, which is frequently used to test different model types have similar attributes later in this work, is:

$$t = \frac{\overline{x}_1 - \overline{x}_2}{\sqrt{\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}}} \tag{1.11}$$

where $\overline{x}_1$ and $\overline{x}_2$ are the sample means, s_1 and s_2 are the sample standard deviations and N_1 and N_2 are the sample sizes. The t-test defined with the t-value given by Eq. 1.11 is known as the Welch's t-test [40]. This t-test's only assumption is that the two populations from which the samples were withdrawn have normal distributions. It is possible to notice that the greater the value of t, the more unlikely H_0 is. That is because the greater the difference between the sample means, the lesser their standard deviations and the greater the sample sizes, the more evident any difference between the two population means become. Next, after the t-value is determined, we must calculate the degrees of freedom ν. This statistic is used later to determine the t-distribution we need to employ in the hypothesis test. The value ν can be approximated by the following relation:

$$\nu \approx \frac{\left(\frac{s_1^2}{N_1} + \frac{s_2^2}{N_2}\right)^2}{\frac{s_1^4}{N_1^2\nu_1} + \frac{s_2^4}{N_2^2\nu_2}} \tag{1.12}$$

where $\nu_1 = N_1 - 1$ and $\nu_2 = N_2 - 1$ are the degrees of freedom associated with the first and second variance estimates. Once ν has been computed, we can calculate the t-distribution, a distribution associated with t-values. If t-values for different samples of same size were taken from the same populations, the values would follow a t-distribution, see Fig. 1.3. The t-distribution probability density function $f(t)$ for a particular ν is given by:

$$f(t) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)}\left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}} \tag{1.13}$$

where Γ is the gamma function. It is then possible to determine the probability, the p-value, of a t-value with Eq. 1.13. Naturally, the lower the p-value, the more unlikely H_0. Generally, the null hypothesis is rejected when the p-value is under a threshold value, see Fig. 1.3. The most common thresholds are 0.01,0.05 and 0.1. In this work, we employ the 0.05 threshold.

T-tests for other statistics such as the correlation coefficient also exist, but are out of the scope of this text [38].

1.4.3. Confidence intervals

Confidence intervals can be calculated using different methods. The one we use is based on the t-distribution, given by Eq. 1.13. Hence, the assumption that the

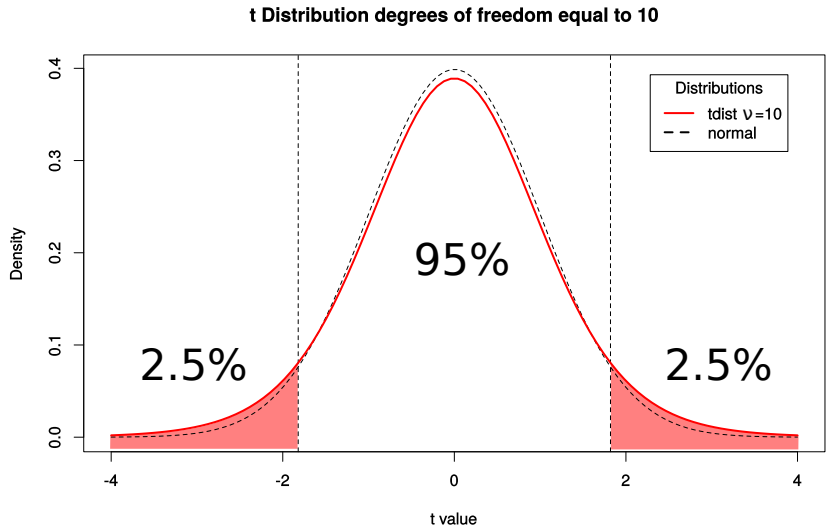

Figure 1.3.: Comparison between a t-distribution with $\nu = 10$ and a normal distribution. The area under the two tails of the t-distribution are highlighted in red. The total area under the tails corresponds to a probability of 5%.

population have normal distributions is also made. The confidence interval is defined by the two t-values delimiting the area symmetrically centered at the t-distribution mean, which accumulates most of the probability. The two tails of the distribution, depicted in red, see Fig. 1.3, are outside the confidence interval range, due to their unlikelyhood. The cumulative probability of each tail is 0.025, considering a 0.05 threshold. Hence, the confidence interval is defined by the t-values that separate the center from the tail regions of the distribution. In the case where we seek to determine the confidence intervals for population mean from a sample, for instance, we first compute the t-distribution using Eq. 1.13 using a simple relation for the degrees of freedom v, $v = N-1$ [38]. Finally, the confidence intervals t' are expressed beside the sample mean, $\overline{x} \pm t'$. This means the population mean is inside the interval with 95% certainty.

At last, we emphasize that, in this work, all t-tests and confidence intervals were computed using the R programming language [41].

2. Generating interdigitated leaf-like channel network patterns

2.1. Overview

Several examples of channel networks are present in nature as well as in modern devices. In many cases, the functions performed by naturally occuring channel networks are analogous to the ones performed by transport systems in man-made gadgets. We came across two power generating devices that make use of channel networks embbeded in porous medium to effectively distribute key substances to the electrodes: PEMFCs and μ-FGPVs, see Fig. 1.2 [24, 25]. In particular, in the case of PEMFCs, it has been demonstrated in previous studies that the design of flow field patterns, the channel networks embedded in the device, has a considerable influence on the performance of these cells [18, 29]. Consequently, it seems natural to investigate the impact of nature inspired channel arquitectures on these devices.

As a matter of fact, fractal-like, biomimetic desings are currently under investigation for PEMFCs, a more established technology, with recent results showing a substantial increase in performance [27, 42] when compared to the other conventionally employed flow field patterns, the pin-type, parallel, serpentine and interdigitated [43], see Fig. 2.1. In fact, in one of these studies, which showcased a leaf venation inspired design, a 20-25% improvement was observed [18]. The improved efficiencies are justified by the more uniform distribution of the reactants.

Motivated by these results, we set out to employ a myriad of channel network designs with even more realistic patterns. In order to do that, we first implement an algorithm capable of generating visually realistic venations [44]. Later, to generate some of the final designs, we add another step to the algorithm, which generates a complementary interdigitated venation pattern, see Fig. 2.2. In later chapters, we present the results of computational fluid dynamics (CFD) simulations and experiments performed on the generated geometries. Ultimately, we seek correlations between key venation properties and the uniformity of reactant distribution in a slab-like system.

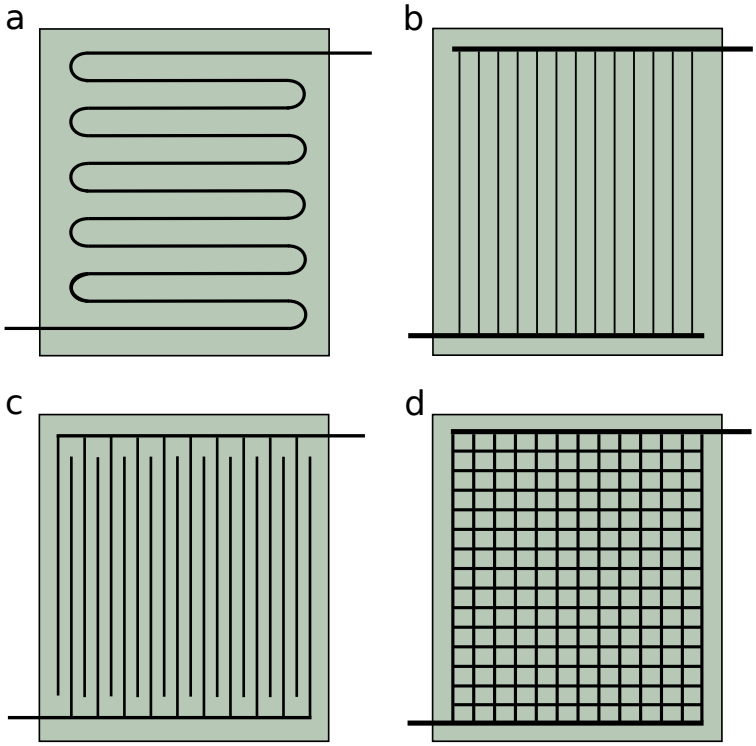

Figure 2.1.: The most commonly encountered flow fields in PEMFCs. Each one has their own advantadges and drawbacks. The designs are, in order, known as (a) serpentine, (b) parallel, (c) interdigitated and (d) pin-type.

2.1.1. Leaf venation descriptions

2.1.1.1. Systematic description

The complexity of leaf venations have marveled and intrigued people since antiquity. That curiosity has driven many researchers to catalogue the different venations encountered in nature [19, 45]. Among the classification systems there is a widely employed one proposed by Hickey [46], which is based on the venations of dicotyledonous plants. Hickey's description stands out due to it considering several venation properties including aspects other than the leaf venation itself, such as leaf shape and structure of the leaf margin [19, 46], see Fig. 2.3. The disposition of the venation relative to the margin is also considered. Due to these and other factors, Hickey's systematic description plays an important role in classifying and identifying taxa [19]. His classification may even help determine acient climate from known correlations. For instance, it has been observed that leaves with brochidodromous venation are predominant in tropical floras while non-brochidodromous patterns prevail in northern temperate floras [47].

Vein hierarchy is another feature accounted for in Hickey's description [19,46,48,49].

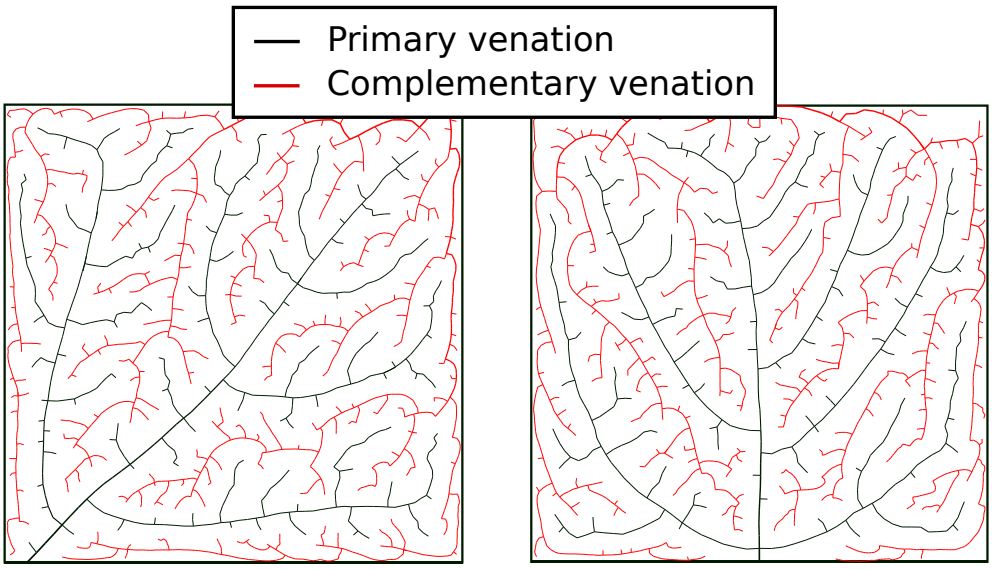

Figure 2.2.: Results from the algorithm we implemented: venation inspired design generated computationally. The primary pattern, in black, is similar to open venations found in leaves. The complementary pattern, in red, is constructed in a subsequent step using the primary venation as a mold. For further information on these results, see Sec. 2.6.

As is the case with other classifications, that is accomplished by introducing the concept of vein orders. In most angiosperm leaves, the venation starts with the primary order vein, or midvein, which starts at the leaf base from the petiole and crosses the leaf blade towards the apex, see Fig. 2.4a. Primary veins are the ones with largest diameter. Secondary veins, on the other hand, ramify directly from the midvein towards the leaf margin and possess a smaller diameter compared to the midvein, Fig. 2.4b. Veins of first and second orders usually form at the first stage of leaf development, a slow expasion phase which occurs mainly due to cell division. Third order veins, in turn, ramify from secondary veins and have an even smaller diameter. These three categories comprise the lower order veins [48, 50]. Minor veins, veins of fourth order onwards, are also present in most angiosperms forming a reticulate which covers the whole lamina, Fig. 2.4b. Minor vein formation takes place during the second stage of leaf development, a 'rapid' growth phase that occurs mostly due to cell expansion. As a result, classification of veins based on orders is also indicative of different stages of leaf development [48, 50, 51].

The usefulness of vein order classification is much broader though. In a recent study, for instance, a correlation between the vein densities (vein length per area - VLA) of lower order veins and total leaf size has been demonstrated [50]. The study not only singled out these correlations by employing a large and diverse database, but proposed a method to estimate total leaf size from the venation found in leaf fragments, making the prediction of leaf sizes from fossil fragments possible. Fundamentally,

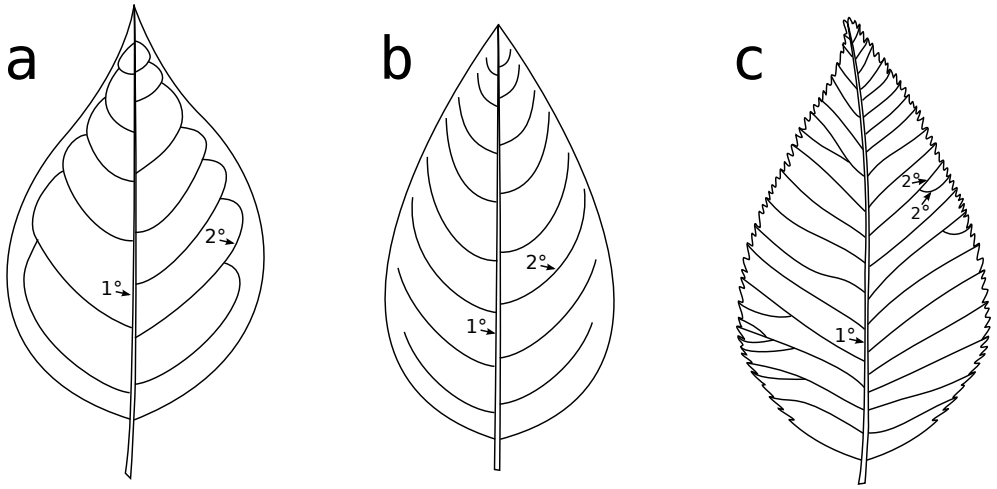

Figure 2.3.: Some of the different categories of leaf venation as proposed and depicted by Hickey. In these sketches, redrawn from Hickey's article [46], only the first and second order veins are portrayed. (a) represents a brochidodromous venation: notice that second order veins are joined close to the margin. (b) illustrates an Eucamptodromous leaf venation, with second order veins ending before they reach the leaf margin. The last venation pattern (c) is an example of Craspedodromous venation: second order veins end only at the leaf margin and may bifurcate as they approach it.

that allows the prediction of the acient climate, since leaf size correlates with the rainfall patterns of a given location. Interestingly enough, when all vein orders are considered no correlation between VLA and leaf size is observed.

2.1.1.2. Topological and geometrical description

Hickey's systematic description of venation patterns provides an exceptional method of identifying taxa. Unfortunately, it is limited to dicotyledonous plants [46]. On the other hand, when determining the link between form of venations and their function, the topological-geometric description has the upper hand. One of the reasons behind this is that it can be applied to any type of venation, including, for example, venations of monocotyledons, such as grass leaves [19].

The topological description resorts to graph theory to represent venation patterns. Venations are then considered as graphs [19, 52]. The nodes of the graph designate points where ramification occurs while the edges portray the veins between two ramification points. It is important to notice that when describing a system in a solely topological manner, the angles and distances between edges and nodes are neglected. Hence, when using the topological framework, the features that receive

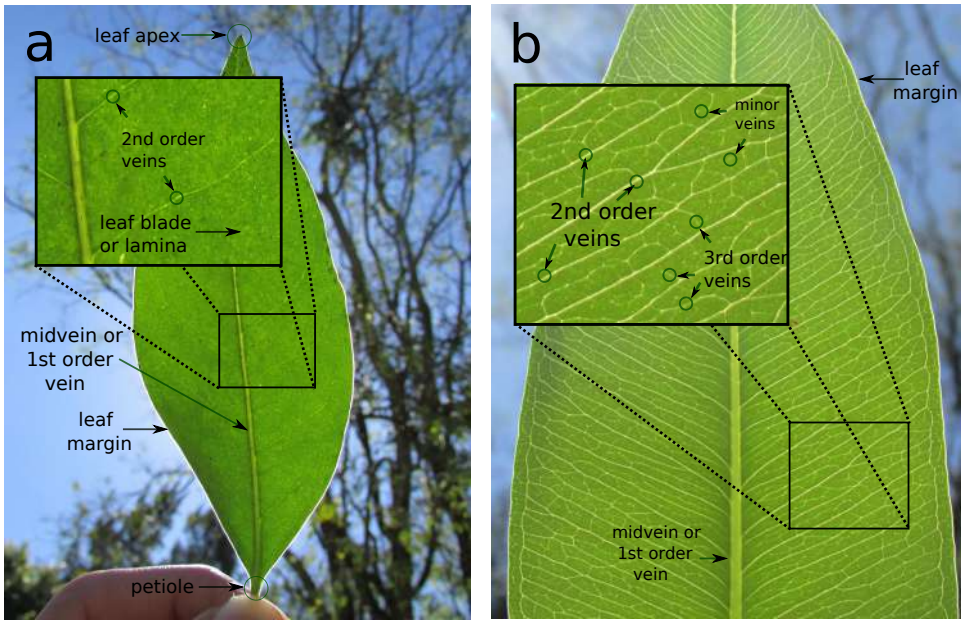

Figure 2.4.: Illustration identifying vein orders in real leaves. The leaf apex, lamina, petiole and margin are also singled out. (a) a leaf with an Eucamptodromous venation type. The second order veins stretch from the midvein towards the leaf margin without reaching it. (b) an example of Acrodromous venation, with two veins running parallel to the margin in convergent arcs towards the leaf apex.

emphasis are the interrelationships between edges and nodes. For instance, the graph representation of venation patterns enables the study of the reticulation of a pattern by observing the redundancy of the graph [19, 53], a graph parameter which allows for the distinction between open and closed graphs [19], see Fig. 2.5. In case the redundancy of the graph is zero, the graph is labeled as open or tree-like, see Fig. 2.5a. In case the redundancy is greater, the graph is denominated closed, see Fig. 2.5b.

There are, however, subtleties when distinguishing between open and closed venations. When considering all of the vein orders of dicotyledonous plants, for example, venations tend to belong to the closed category, although they are not completely closed, since minor veins may sometimes end freely in the lamina. On the other hand, if only first and second vein orders are considered, the scenario becomes more diverse. Brochidodromous venations, for instance, are designated as closed whereas craspedodromous and eucamptodromous venations are open, see Fig. 2.3. Both venation types have different adaptive capacities, which are throughly discussed in the literature [19, 48].

Finally, geometric traits of the venation graphs may be accounted for by embedding them in the Euclidian plane. Nodes of the graph become points in the plane while edges become segments. Distances and angles become relevant, in contrast with the

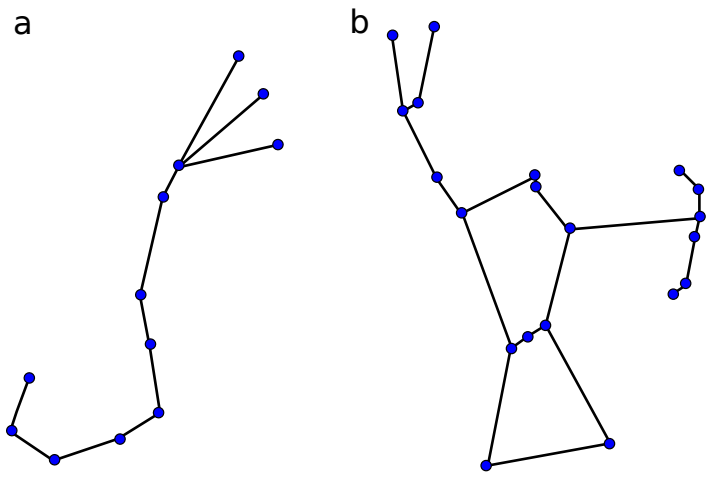

Figure 2.5.: Graphs illustrating the different graph types and the redundancy parameter: (a) open graph, since the graph redundancy is zero and (b) closed graph, since the redundancy is greater, meaning there may be more than one path to reach a node n_1 from a node n_2.

topological description. This resulting graphs are called Euclidian graphs [54]. Using the Euclidian metric is fundamental to maintain the visual resemblance between venation patterns and their respective graph representation, as discussed in Sec. 2.3. Hence, we use this decription when implementing the algorithms for venation construction [44].

2.1.1.3. Fractal dimension description

Statistical self-similiarity is a property of many natural processes and objects, such as coastlines, skylines, wall cracks, etc. In particular, ramifying structures, e.g., venations, also present this trait. Self-similar objects under varying degrees of magnification are known as fractals, a term coined by Benoit Mandelbrot, referred to by many as the father of fractal geometry [55, 56].

Among the tools fractal geometry leaves at our disposal is the concept of fractal dimension. The most usual definitions of dimension, such as the Euclidian and topological dimensions, lead to integer values. Fractal dimensions, on the other hand, allow for the possibility of objects having non-integer dimensions. For instance, a line has the topological dimension D_T of 1, while the coastline of Britain has a fractional dimension D of 1.25 [56]. In a sense, the fractal dimension of an object may be seen as a measure of how well it fills the space containing it: the greater its value the better it fills up the space. Interestingly, from this remark, it follows that the coastline of Norway ($D = 1.52$) [57], for example, is better at covering up the two dimensional plane than the coastline of Australia ($D = 1.13$) [56].

The fractal dimensions of venations patterns of different species have been determined in many studies. For instance, *Acer trautvetteri* presents a fractal dimension of 1.55 [58], while *Macropeplus ligustrinus* possesses a fractal dimension of 1.4 [59]. Not only these results confirm that venations do a good job of covering up a plane, but they hint at a method to identify different plant species based on the fractal dimension parameter. Indeed, a study aiming to identify different plant species of the Brazilian Atlantic forest and Cerrado did just that. It used the fractal dimensions of venations and leaf shape, in conjunction with pattern recognition techniques, to successfully identify the different taxa [60].

The numerical definition of fractal dimension, however, is not unique. In fact, there are quite a few definitions, each with its own set of advantages and drawbacks. In this work, we employed the box counting method to determine the fractal dimension of the venation patterns. In order to determine the box counting fractal dimension D_B of a 2 dimensional object, for instance, the plane may be covered with a square grid with cells of side length δ. Next, the number $N(\delta)$ of boxes which capture a part of the curve is counted. The following definition may then be used to compute D_B:

$$D_B \sim -\frac{\ln N(\delta)}{\ln \delta} \tag{2.1}$$

In practice, $N(\delta)$ changes depending on the side length of the grid cells chosen. In particular, if δ is large the estimate for D_B might be innacurate as well as if δ is too small, since self-similar objects that appear in nature are not true fractals, meaning they are not self-similar under all scales. Hence, to circumvent this problem, the fractal is covered with a sequence of grids with decreasing δ. The best fit for the data may be subsequently determined by applying the method of least squares for a more reliable estimate of D_B [55].

2.1.2. Vein development: canalization hypothesis

In this work, a set of algorithms able to produce visually accurate patterns serve as the steppingstone to channel network generation. The algorithms, proposed by Runions et al. [44], find their inspiration on previous models of vein formation, some of which are based on a hypothesis of leaf vascular development, the *canalization hypothesis* [44,61–65]. Consequently, the model itself is consistent with the biological knowledge of venation development.

The *canalization hypothesis* states that vein differentiation occurs in response to a signal, and that at least part of the signal is composed of auxin, a hormone inherent in plants [44,61–64]. Auxin produced throughout the leaf blade must be removed for leaf development to resume. Although diffusion takes place, auxin removal from the leaf also occurs due to a directional transport of the hormone. The PIN family of integral membrane proteins, in particular, the PIN1 protein play a major role in that

directionality [48, 51, 66]. As auxin is driven out from regions of high concentration, it induces new veins in their direction, which aim to eliminate the signal surplus by carrying it towards the leaf base. As a rule, vein differentiation is oriented towards existing vein tissues with excess draining capacity. As a result, a feedback process is established: cell differentiation increases their capacity to transport the signal which induces the differentiation [44, 61–64].

A formal analogy between rain water forming channels as it moves down a sandy slope and the induced veins can facilitate the understanding of the process of gradual vein development [62]. Indeed, the carved channels *canalize* liquid from its surroundings, discharging it into larger channels. The leaf veins, in turn, *canalize* the excess auxin into narrow strands, discharging it into larger veins that ultimately conduct it to the leaf base.

Finally, the last remark relevant to the algorithm functioning involves observations of auxin sources shape in the leaf. It turns out they tend to be localized in space, to the extent that they may be assumed as discrete [44, 67]. These observations may have in fact shed light on some aspects of vascular formation formerly unexplained by the *canalization hypothesis*. Nevertheless, many questions concerning vein pattern formation remain unanswered [68].

2.1.3. Venation functions

The leaf venation has two primary functions. It plays a part in the mechanical stabilization of the leaf and it is also responsible for the transport of substances throughout the lamina [19, 48, 49]. Concerning the first, it has been investigated how veins, in conjunction with other tissues, combine to display the observed mechanical properties of leaves. For intance, it has been demonstrated that the angle formed between the second order veins and the midvein has an impact on the overall mechanical stability of the leaf. Moreover, aspects such as leaf size, the E-modulus of the leaf tissue or structures along the leaf margin are also significant [19].

Regarding the second role of venations, it is clear that they are the main structure responsible for providing water and the nutrients necessary for normal leaf functioning and are also responsible for removing the products of photosynthesis [19]. Venations, as well as the rest of the vascular system of plants, are comprised by two types of tissue, each type responsible for addressing one of these tasks. These are the xylem, which performs the water and other nutrients transport from the root to the leaves and the phloem, which carries the organic products (sap) from the leaves to other parts of the plant that require it [69]. Water enters the leaf via the venations, accounting for the major portion of material transported in the xylem. In addition, it leaves the leaf through evaporation at the stomata. Indeed, evaportation is the driving force that drags water from the root upward until it reaches the leaf. This is in contrast with the mechanical pumping of blood in animals performed by the

heart. This difference has cast many doubts on the validity of Murray's law to the vascular system of plants [19, 70–72], as discussed in Sec. 2.3.

A last remark worth discussing is related to the differences between water transport in closed and open venations. The redundancy associated with closed venations, for example, protects leaves from a possible injury which a eliminates a vein path [2, 18, 19]. While in an open pattern, this elimination could potentially result in leaf death, the damage in a closed pattern is compensated by other available paths. Moreover, studies suggest that the pressure profiles of open and closed venations differ [19]. Flow in leaves can be simulated treating leaf parenchyma as a porous medium [19–21]. The simulations show that closed systems tend to display a more uniform pressure distribution when compared to open venations with the same VLA.

2.2. Remarks on the use of venation designs on possible targets

In light of the discussed in Sec. 2.1, we consider potential issues to the direct application of venation patterns as the channel network designs in PEMFCs and μ-FGPVs. Moreover, we present some modifications to the venation pattern designs, which are expected to solve the issues. In Sec. 2.5, we briefly examine how these adjustments were implemented.

First off, leaf venations comprise both source and collecting channels, which are, correspondingly, the xylem and the phloem tissues of the plant vascular system. Thus, flux occurs simultaneously towards and away from the lamina in different sections of the same veins. The water and nutrients are carried via the xylem while the phloem transports the products [69]. When only the upward transport of water is considered, the vein segment at the petiole can be considered as the single inlet of the leaf system. In contrast, multiple outlets spread throughout the blade exist, the stomata where evaporation occurs. In the target of the venation pattern designs, however, there is a single flux direction, which carries both the reactants and products in the same ducts. Moreover, the inlet and outlet are located at different parts of the system [18]. In order to solve this issue, a second channel network was introduced. With the second network, the complementary venation, see Sec. 2.5, the listed requirements are met: flow is established from one network to the other in a single direction, reactants and product are tranported by both channel networks and the inlet and outlet are placed at opposing locations of the system.

Secondly, as pointed out in Sec. 2.1.3, the pressure distribution in closed venations is more homogeneous than in open venations due to path redundancy [19]. Furthermore, the presence of loops make areas of stagnant flow more likely to occur [18]. In plants, the drawbacks of closed arquitechtures are largely offset by their many adaptive advantadges over open patterns, such as the reduced impact of vein injury [2,19].

On the other hand, these adaptive capabilities are irrelevant when considering applications such as PEMFCs, for instance. Thus, we focus on the generation of open interdigitaded patterns, as open patterns have already been shown to be optimal when efficiency is considered [2, 19]. In future work, we plan to consider patterns which are closed from the third vein order onwards, in order to study the effects of redundancy on reactant distribution.

Another aspect to be discussed concerns the boundary shape of the target applications, see Fig. 2.1 . Generated venation architectures can be affected by the border and growth type chosen [44]. Hence, while a myriad of leaf shapes exist in nature, we are constrained by the boundary type of the target applications. Therefore, we developed venation patterns embedded exclusively in a rectangular shaped system, since this was by far the most common configuration found in the literature [18, 24, 27, 42, 43].

Lastly, we again highlight that it is the sun that promotes the transport in the plant vascular system through evaporation, i. e., an external source does the work, meaning that plants themselves do not spend energy to induce transport [19]. In the vascular system of humans, however, the picture changes, since it is the heart that pumps the blood and generates the flow [73]. This fundamental difference has cast a doubt on the applicability of Murray's law to venations in order to describe vein diameter relations at ramifications [70]. Nonetheless, the target applications resemble the our circulatory system in regard to flow induction. In PEMFCs, for instance, air compressors are generally used to pump reactant gases into the cell, playing a role analogous to the heart. As a result, we employed Murray's law to define vein diameters despite the uncertainty of its validity to venations. Moreover, application of Murray's law seemed to play a significant role in the performance increase observed in a recent study about venation inspired-geometries [18]. The exponent n employed in equation Eq. 2.2 was set as a parameter of the model.

2.3. Algorithm for open venation pattern generation

The algorithm we implemented for open venation patterns generation proposed by Runions et al. [51] is grounded on a set of key assumptions. The first is that vein induction and insertion tends to occur between the auxin sources and nearby veins. That is in agreement with the model proposed by Mitchison for vein induction [44, 74]. The second key aspect is that the algorithm not only computes vein induction, but actual vein insertion takes place. Moreover, to accurately simulate the vein development, leaf blade growth is also considered [44]. Auxin sources are deemed as discrete, as that assumption is supported, to some extent, by experimental data [67]. Furthermore, the model proposed by Runions et. al. is set up on the continuous space instead of a grid, as occcured in a previous model proposed by Gottlieb [44,75]. This modification allows for the generation of venation patterns strinkingly similar to the ones found in actual leaves, see Fig. 2.2.

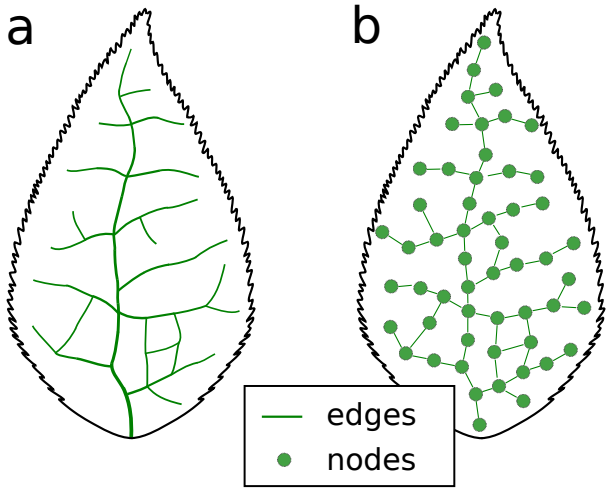

Figure 2.6.: A venation pattern example (a) and its corresponding embedded graph representation (b). Edges represent the veins, while the nodes are placed at ramification points or locations where veins change direction.

Graph theory's framework grant us all of the features which are essential for the implementation of the algorithm in the continuous space. Venation patterns may be viewed as a graph $G = (V, E)$ embedded in the Euclidian space, see Sec. 2.1.1.2, where E, the set of edges, correspond to the veins of the pattern and V, the set of vertices, denote the channel bifurcations or points where change of orientation occurs, see Fig. 2.6. It is clear from this comparison that the vertices, or vein nodes, are the building blocks of the simulated venation patterns and, as consequence, it follows that vein node placement has a great impact on the overall generated venation architecture. Therefore, in a nutshell, what the algorithm does is essentially handle vein node placement as simulated leaf growth occurs. Observe that the rules regulating vein node positioning are generally in accordance with the discussed in Sec. 2.1.2, meaning that veins are induced by discrete auxin sources embedded in the plane [44]. Finally, the complete tree-like graph G, as well as the auxin sources, which can be assumed as edgeless nodes, are confined to the leaf perimeter.

The graph G representing the venation is directed and all of its edges are oriented away from the tree data structure's root, which corresponds to the node at the leaf base. In fact, that is the definition of an out-tree or arborescence, Fig. 2.7 [76]. The corresponding data structure is similar to binary trees, which are discussed in Sec. A.2.2.1, except for the fact that each internal node may have more than 2 children, see Fig. 2.7. As such, many of the recursive algorithms utilized in the binary tree context (e.g., tree search, traversal, insertion, etc.) [77] can be readily adapted to treat the data structure that stems from G.

Before proceeding to the nuts and bolts of the algorithm, we consider a final key point: the initial state of the leaf. In general, each simulation starts by placing a

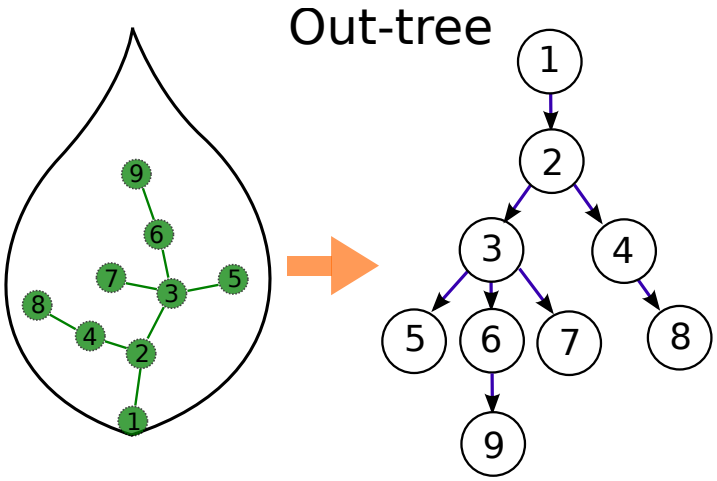

Figure 2.7.: Graph represenation of the venation pattern data structure: the arborescence or out-tree.

single vein node, the seed, within the leaf contour. Customarily the seed is inserted in the vicinities of the leaf boundary, although that is not a requirement. The region the seed is placed, referred to as the leaf base, gives rise to the primary vein of the leaf. Nevertheless, more than one seed can be utilized to produce different leaf varieties (e.g., grass leaves) [44]. Consequently the initial configuration of seeds within the simulated leaf blade may be compared, to a certain degree, to the leaf primordiums of different species [62].

Finally, we discuss practical details of the algorithm. One of its main aspects is that it is structured upon a main loop, see Fig. 2.8 [44]. We stress that the first loop iteration begins with no auxin nodes and some specified seed arrengement, but aside from that, there is nothing differentiating it from the other iterations. Hence, we set out by examining the group of instructions contained in a single loop iteration, as that suffices for the comprehension of the overall geometry evolution.

The first step within each iteration is the insertion of new auxin nodes throughout the leaf blade. Auxin nodes are, thus, placed randomly in space, see Fig. 2.8b. We highlight that even though individual nodes are placed randomly, the resulting density of the auxin node distribution, however, must obey a density parameter ρ_{auxin}, as discussed in Sec. 2.5. In addition, there can be neither vein nor auxin nodes within a distance threshold value k_d, the *kill distance*, see Fig. 2.8b, another input parameter provided by the user. As a result, the auxin node candidates which do not fit the last criterion, i.e., auxin nodes candidates which possess other nodes within the specified limit are readily discarded, see Fig. 2.8b,c.

The next steps consist in determining the effect of the current auxin node distribution on new vein segments induction. Each auxin source, in the open venation pattern algorithm, influences only the vein which is closest to it. The effect of this choice is in accordance with the canalization hypothesis. Computationally that is

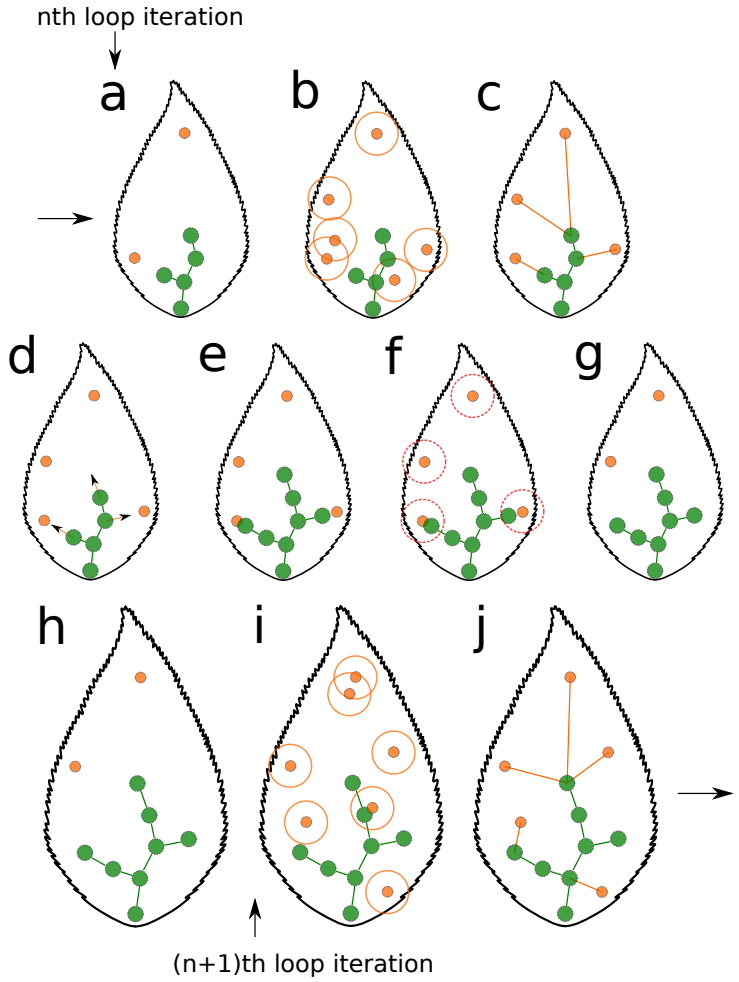

Figure 2.8.: Diagram representing the steps of a loop iteration of the open venation algorithm.

translated into auxin nodes influencing only their nearest vein node, see Fig. 2.8c. Notice that many vein nodes may not be influenced by an auxin node and that, meanwhile, some vein nodes may be influenced by one or more auxin nodes. In the latter case, the program determines the direction of the resultant of the vectors defined from the vein node to each auxin influencing it, see Fig. 2.8c,d. Subsequently, new vein nodes are inserted onto the plane along the directions computed in the previous step, see Fig. 2.8e. The distance from the inserted node to parent vein node is set as another input parameter.

Provided vein nodes were introduced in the last step, the next natural procedure amounts to removing the auxin nodes which had their immediate neighborhood invaded by the new nodes, Fig. 2.8f,g. Node removal is dependent on a vein node violating the same minimum distance threshold k_d mentioned earlier. Verifying threshold violation is achieved by computing the distance from the auxin to the

nearest vein node. All of the aspects regarding the search for nearest nodes in this and the previous steps are accomplished via a point location algorithm [78] and Voronoi diagram space partitioning [79], so as to optimize nearest neighbor search time performance, as thoroughly explained in Appendix A.

The last step encompasses simulation of leaf growth, Fig. 2.8g,h. Although other types of growth could have been selected, we have chosen to simulate uniform isotropic growth for all geometries, due to its straightforward implementation. Hence, leaf blade shape is mantained while all vein and auxin node positions are updated. The repetition of this set of instructions, see Fig. 2.8i,j, produces the sought open venation patterns. As the loop advances further, the venation becomes more detailed [44].

After the venation pattern is generated, vein widths were assigned to each vein node of the arborescence data structure [44]. This was done recursively, using a traversal algorithm. Due to this we had to assign the same minimum width to every venation free ends. The width choice is arbitrary and is performed to simplify computations. Once the algorithm assigns a minimum width to the free end veins, it computes the widths of parent vein nodes as the out-tree is traversed. This process is concluded when the primary vein at the base of the leaf is reached, that is, the arborescence root. The only constraint imposed on the vein widths is, naturally, Murray's law [44, 73], which establishes the following relation between the radii of the parent vein and its children whenever a ramification occurs:

$$r_p^n = r_{c1}^n + r_{c2}^n + r_{c3}^n + \ldots = \sum_i^N r_{ci}^n \tag{2.2}$$

where r_p is the radius of the parent vein, N is the number of children veins at the ramification point, r_{ci} is child vein radius and n is the Murray's law exponent [44,52]. Once the relation between the parent and children veins is determined according to minimum width free end vein criterion and Murray's law, the algorithm goes through the tree again from the root to the ends, this time assigning an input diameter of 2 mm to the root vein and changing the vein node widths accordingly, always keeping the radius relations between parent and children veins determined in the first traversal. A minimum vein diameter of 0.25 mm was also enforced, meaning that whenever a vein width is computed to have a lower value, Murray's law is disregarded. This threshold was chosen due to the limited resolution of the 3D printer at $LabM^2$.

Murray's law was proposed initially for channels of circular cross section in a famous 1926 article [73]. The relation was obtained by applying the principle of minimization of work to the circulatory system of a human being. Two opposing factors play a role in the energy consumption by the circulatory system of a living organism: the energy necessary to mantain the blood, a living fluid, proportional to its volume, and the energy required to overcome the viscous forces that arise in the channels, which is known to obey Poisseuille's equation. The exponent n for circular cross

section case, in Eq. 2.2, was shown to be 3. The vascular system of leaves has been demonstrated to follow Murray's law in some cases, when the lower vein orders are considered. In theory, that shouldn't necessarily, as the energy necessary to overcome the viscous forces inside the veins comes from the Sun, an external energy source. These considerations may be neglected in our case, however, since the source of energy in the target applications of our work are internal and therefore, in theory, should obey Murray's law. This has been discussed in Sec. 2.2. The exact value of n, however may vary. Hence, as proposed by Runions et al. [44], n is treated as a parameter of the model. Further considereations on the relevance of Murray's law are made in Sec. 2.6.

2.4. Algorithm for closed venation pattern generation

The implemented algorithm for closed venation pattern generation, also proposed by Runions et al. [44], closely resembles the one for open venation patterns discussed in Sec. 2.3. The only major different assumption between the two involves the zone of influence of auxin sources. In the algorithm for open patterns, auxin sites only influence its nearest neighbor vein site. Comparatively, in the algorithm for closed patterns, auxin sources may induce more than one vein node simultaneously. Such situations are hypothesized to occur in nature whenever the induced veins are close to the auxin source, but relatively far from each other [44]. In order to simulate this process, the model employs the relative neighborhood criterion, see Eq. 2.3 [44, 80]. As a result, vein loops (*anastomoses*) may form when several vein sites are caught in an auxin source zone of influence.

2.4.1. Relative neighborhood graphs

Briefly, given a set S of points, a point p is a relative neighbor of another point s (with both p and $s \in S$) if it obeys the following mathematical definition:

$$\forall u \in S \quad \text{with } u \neq p, s \quad d(p,s) < \max(d(p,u), d(s,u)) \tag{2.3}$$

where d denotes the Euclidian distance. Intuitively, points are 'relatively close' if they are at least as close to each other as they are to any other point in the set [80]. A visual representation that follows from the definition of relative neighborhood of a point s is presented in Fig. 2.9. Lastly, the graph of any set of points S, constructed using the relative neighborhood criterion (Eq. 2.3), is called the relative neighborhood graph (RNG) of that set.

Notice that the definition given in Eq. 2.3 offers a straighforward method for building RNGs. In practice, all that need to be done would be verifying whether each pair

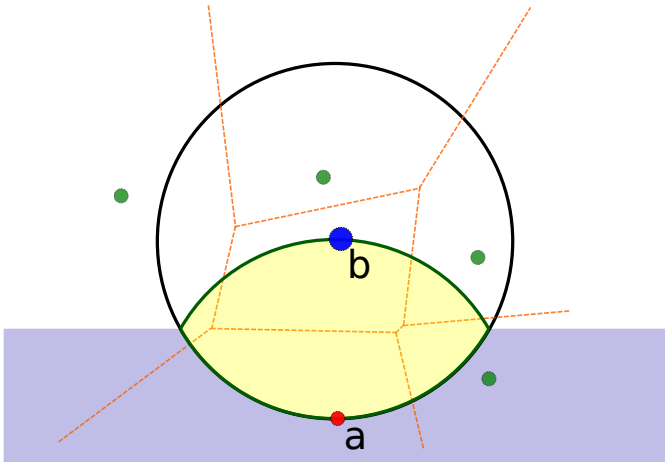

Figure 2.9.: A visual representation of the relative neighborhood criterion. In the figure, the relative neighborhood of point b with respect to point a is being decided. In fact, a fits the relative neighborhood criterion, since there are no points in the yellow region. In case there were, that would automatically exclude a from b's relative neighborhood. Additionally, point a excludes all points in the blue region from b's relative neighborhood.

of points in the set S obeys Eq. 2.3. Naturally, for each pair checked, this 'naive' method demands that every other point in the set S (excluding the pair points) be assessed. Considering the previous remarks, we proceed to analyze the time complexity of this RNG construction approach as the number n of points in the set S increases. That is done to determine the feasibility of this method. First, observe that the number of pairs in a set with n points is $\frac{1}{2}(n^2 - n)$, which translates to $O(n^2)$ in the big O notation [1]. Moreover, the inspection of $n - 2$ points is needed in order to evaluate whether the points of each pair are relative neighbors or not. Taking that into account, we find that the time complexity of this approach goes with $O(n^3)$ [44], since the total number of tests executed to generate the RNG is $\frac{1}{2}(n^2 - n)(n - 2)$. In other words, that is a very time-consuming algorithm.

Aiming to improve the performance of relative neighborhood assessment, one can resort to a property of RNGs: the relative neighborhood graph of any set of points S is a subset of the Delaunay triangulation of that set [80]. Consequently, all relative neighbors of a point are also Delaunay neighbors, although the reverse is, in general, not true. That means that if the Delaunay triangulation of S is known, it is possible to narrow the search for relative neighbors. Instead of testing every pair in the set, it is possible to check only the Delaunay neighbors to see whether they fit the relative neighborhood criterion. Considering this and the fact that there is an upper limit of

[1] The big O notation is widely employed in the analysis of algorithm efficiency [77]. It captures the way number of operations grow as n goes to infinity. Multiplying constants are disregarded. A detailed discussion explaing why and a formal definition of the big O notation can be found in Drozdek [77].

at most 6 to the average number of Delaunay neighbors per point[2], we observe that the number of pairs we must verify only increases linearly ($6n$), i.e., the number of pairs that must be checked go with $O(n)$, as opposed to the $O(n^2)$ quadratic behavior of the naive approach. Observe, however, that we must be in possession of the Delaunay triagulation of the set to make use of these attributes. Fortunately, all Delaunay neighbors can be readily identified from the Voronoi diagram of the set S, which, in turn, can be generated by employing Fortune's algorithm in $O(n \ln n)$ time, as discussed in Sec. A.2 [79]. Hence, through this method, originally proposed by Toussaint [80], we can construct the exact RNG of the set S in $O(n^2)$ time, a substantial improvement on the $O(n^3)$ of the straighforward approach previously considered.

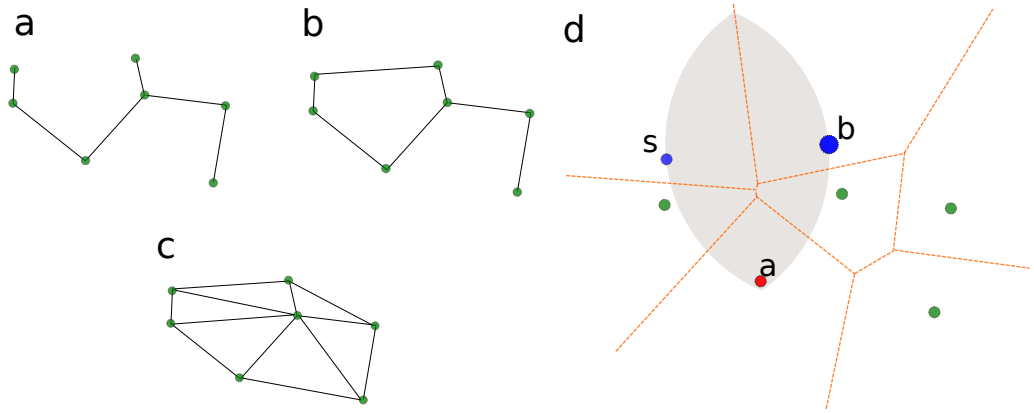

Figure 2.10.: (a) Relative neighborhood graph, (b) Urquhart graph and (c) Delaunay triangulation of the same set of points. (d) shows why the RNG and the Urquhart graph are not the same. Point a, which is not a Delaunay neighbor of neither s nor b, is excluding s from the relative neighborhood of b. This situation, however, rarely occurs in a set of randomly distributed points across the plane.

The $O(n^2)$ performance may be further enhanced by resorting to an approximation, first proposed by Urquhart [83]. Generally, we must verify every point in S to determine whether a Delaunay neighbor p of the point s fits the relative neighbor criterion. The approximation suggests that, instead, only the other Delaunay neighbors of s should be verified. The idea is that those Delaunay neighbors have a higher chance of excluding p from the relative neighborhood of s, as opposed to other points of the set S. Consequently, the number of tests that must be performed to contruct the graph is reduced to a finite number that does not grow as n

[2]This property stems directly from Theorem 1, discussed in Sec. A.4. In order to prove this, we must recall that Voronoi diagrams and Delaunay triangulations are, in fact, dual graphs [81,82]. It follows from this fact that there is correspondency between edges of a Voronoi diagram and pairs of Delaunay neighbors. Since the upper limit on the average number of edges per face is 6, and, consequently, the average number edges per site ($f = n$), the average number of Delaunay neighbors per site is also, at most, 6. Naturally, we must take into account that faces share edges, so we must consider every edge twice to accurately compute these averages.

increases. That follows from the property of Delaunay triangulations, which state that the average number of Delaunay neighbors per point is at most 6. Therefore, the approximated graphs can be generated in $O(n \ln n)$ time, much faster than the original $O(n^3)$ behavior. Finally, it has been demonstrated by Andrade and De Figueiredo [84] that graphs generated employing this method are, in fact, excellent approximations to RNGs for random samples, see Fig. 2.10. In addition, Runions et al. [44] has stated that no striking differences in produced venations appear when this modification is used. Thus, Urquhart's algorithm was the preferred method for encountering the approximate relative neighborhood of the auxin sites, even though there are methods available in the literature for the construction of exact RNGs from the Delaunay triangulations of any point set in linear time $O(n)$.

2.4.2. Closed venation pattern algorithm implementation

Provided the aspects discussed in Sec. 2.4.1 are understood, adapting the algorithm for the open venation generation to the closed venation case becomes straightforward. The major modification has to do with the algorithm's response to vein nodes entering an auxin source's *kill distance* k_d threshold. Instead of immeadiately removing the auxin node from the leaf blade, a procedure that ultimately produces open patterns such as the ones from Sec. 2.3, we monitor the evolution of the veins induced by auxin sources until they either reach the auxin node or leave their zone of influence. Computationally, that is accomplished by attaching identifiers to all vein nodes under the influence of the auxin source as soon as the first vein node is placed within the *kill distance* k_d perimeter around the auxin. Naturally, being under the influence of an auxin node means being part of its relative neighborhood. In the subsequent simulation steps, the identifiers are passed on to the children of the marked vein nodes. The transmission of the identifiers stops when a child either crosses the k_d limit or until it leaves the relative neighborhood of the auxin source. Lastly, when no vein nodes under influence of the auxin source remain, the node is finally removed and the induced veins are joined at its last location [44].

The recursive computation of the vein widths in this algorithm start at the vein nodes corresponding to the free ends of the venation, as in the case for open venation patterns, or it starts at the nodes where two or more veins were joined together. As in the algorithm for open patterns, we employed Murray's law, see Eq. 2.2, to determine the radii of the ramifying veins.

2.5. Algorithm adjustments for design generation

At this point, we have already presented the most fundamental aspects of the algorithm which generates the final geometries. In this section, we consider the implementation details of the necessary algorithm adjustments discussed in Sec. 2.2.

The main modification we examine is related to the complementary venation construction, see Fig. 2.11, which may be connected to either the inlet or outlet. Depending on the degree of reticulation of the closed pattern, creating a non-overlaping interdigitated is simply not possible. Therefore, the interdigitated geometry was only generated when the primary pattern is an open venation, since it is crucial to stop veins of different patterns from crossing each other. The closed venation case is only considered later on.

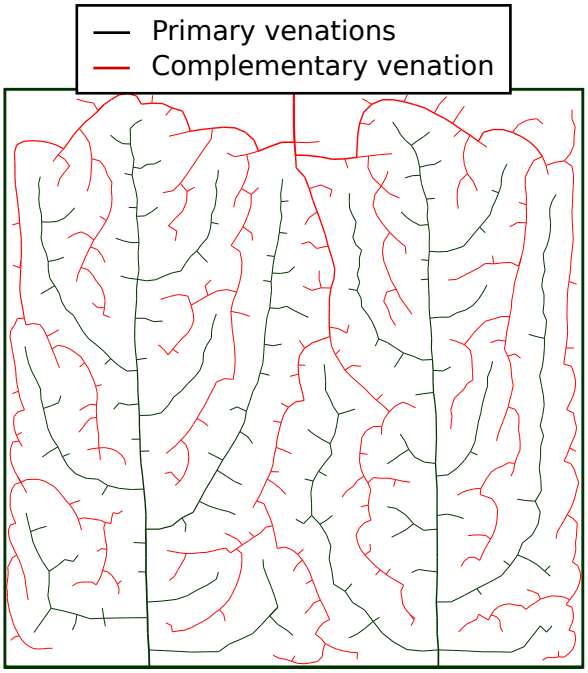

Figure 2.11.: An open three-venation design. There are two primary venations running in parallel in this arrengement.

It is clear from Sec. 2.3, that a second venation can be generated by placing a second seed at any location along the border from where the venation will grow. In fact, it is possible to use this method to generate a design with more than two venations, such as the three venation system we have formed, see Fig. 2.11. The procedure of placing an additional seed is also used by Runions et. al. [44] to form venations of grass leaves, which, in contrast to dicotyledonous venations, consist in a number of first order veins running in parallel along the blade from the leaf base towards the appex. However, there are two elements specific to complementary interdigitated pattern generation.

The first one is that the second seed is only placed after the primary pattern is formed. That is done in order to avoid venations 'sensing' each other, an effect that occurs whenever they are generated simultaneously. In case the seeds are placed at the onset of the run, both venations quickly expand and define their area of influence, making it impossible for the other to enter its zone. This has to do with

the inner workings of the algorithm for open pattern generation, already discussed in the previous sections.

The second element stems from the need to avoid venation overlapping. In other words, in order to generate an effective interdigitated pattern, it is desirable that the second pattern penetrate the area of influence of the first, without any veins crossing or becoming too close. We achieve this via a square grid. The cells occupied by the first pattern and its neighbors are disabled when the second venation grows, see Fig. 2.12a-d. Aside from this changes, the primary and the interdigitated venation are formed in the same manner. Patterns generated through this method can be seen in Fig. 2.13a,b and e. This result showcases the usefulness of the grid. The square grid, however, has two additional functions.

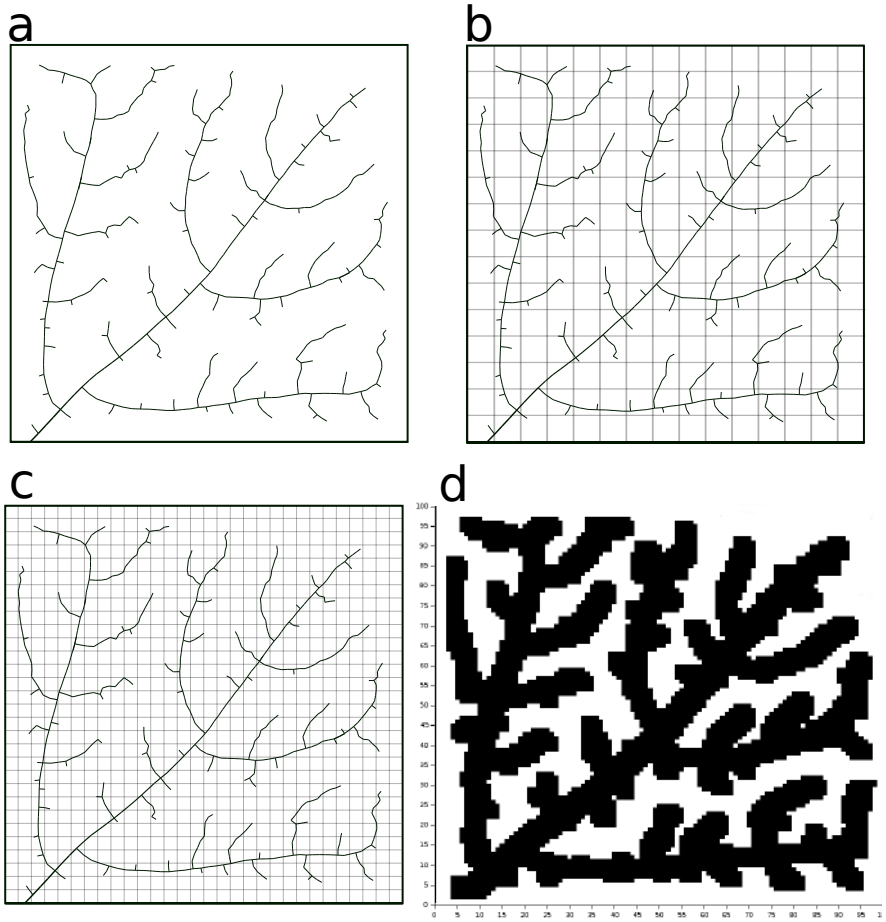

Figure 2.12.: (a) The same diagonal venation pattern covered by a (b) 15x15 cell, a (c) 30x30 cell and a (d) 100x100 cell grid. Figure (d) additionally shows cells that are disabled for interdigitated venation growth, in black, and cells which are allowed, in white.

Another of its roles is related to auxin source placement. Although the employed pseudo-random number generator (PRNG) [85] draws numbers from an uniform

distribution, the amount of auxin sources is often not large enough to produce the desired regularity and uniformity of nodes in the plane. Specially when the total number of nodes is very low, many regions appear where the local auxin density is much higher or much lower than the average. Hence, the grid is introduced to provide support in the process, such that the random distribution of auxin nodes of any given step of the simulation is as uniform and regular as possible. After the implementation of these changes, the grid cell size δ and auxin source density ρ_{auxin} are the input parameters which allows the user to control the distribution of nodes, ultimately controlling the final produced pattern, see Fig. 2.12.

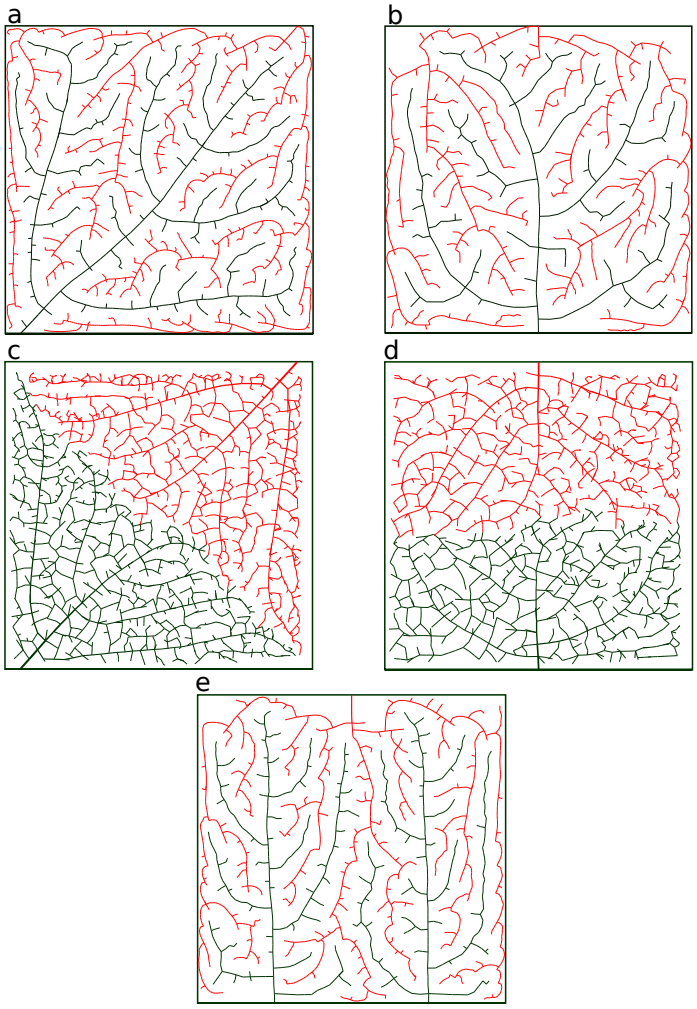

Figure 2.13.: The five different design categories generated. They are, respectively, the (a) open diagonal, (b) open centered, (c) closed diagonal, (d) closed centered and (e) open three-venation setups.

The last remaining grid function to be discussed has to do with the determination of the fractal dimension of the patterns. As hinted at in Sec. 2.1.1.3, one of the methods employed in the computation of the box counting fractal dimension relies

on square grids. In this method, the fractal is covered with a sequence of grids of different cell length and the number of cells occupied by it is, then, counted, see Fig. 2.12. Subsequently, equation Eq. 2.1 can be used to estimate the object's box counting fractal dimension. Naturally, the functions associated with square grid construction are conveniently exploited in this whole process, making it particularly straighforward. Some of the results are shown in Sec. 2.6.

Finally, we also have generated closed patterns using the algorithm presented in Sec. 2.4. Again a second venation pattern is generated. In this case, however, the second venation is not interdigitated. In fact, the first venation and second venation are generated concomitantly and in a similar fashion. Simultaneous generation is performed in order to allow both networks to join each other, see Fig. 2.13c,d. To a certain degree, the closed venation design type resemble the pin-type flow fields, see Fig. 2.1d, which are commonly found in PEMFCs [28].

2.6. Results

In this section we present the results related to the venation-based geometries. We have generated a database with 18 different designs. All designs have square borders with identical area equal to $50\,\mathrm{cm}^2$. In order to make the database more comprehensible, we have also divided it two categories: open centered, open diagonal, see Fig. 2.13a,b. The criterion that separates diagonal from centered patterns is the position where first order veins intersect the margin. In case the intersection points are not at the center of square border edges we define the pattern as a diagonal geometry. In case the opposite occurs, we label it as a centered geometry. The intersection points in the case of diagonal geometries were introduced very close to the square boder corners, always in a simmetrical fashion, see Fig. 2.13a. Finally, the number of generated samples for both the open centered and diagonal categories was 9. Their quantified properties, combined with the results from CFD, may give us an idea of whether one venation type, if any, is more efficient at distributing the reactants.

The implemented algorithms, as discussed in previous sections of this chapter, also allow the generation three additional types of venation patterns: the closed centered, closed diagonal and open three-venation designs types, see Fig. 2.13c-e. These catagories will be analyzed in future work, see Chapter 5.

The geometry properties we have quantified, were already discussed in previous sections, particularly in Sec. 2.1.3. They have been chosen due to either their importance in the study of venation patterns or their relevance in the target applications. The selected properties are the fractal dimension, the vein length per area (VLA), the bifurcation number and Murray's law exponent n.

The box counting fractal dimension D_B of each separate venation as well as the joint fractal dimension of every design has been computed using the method discussed in

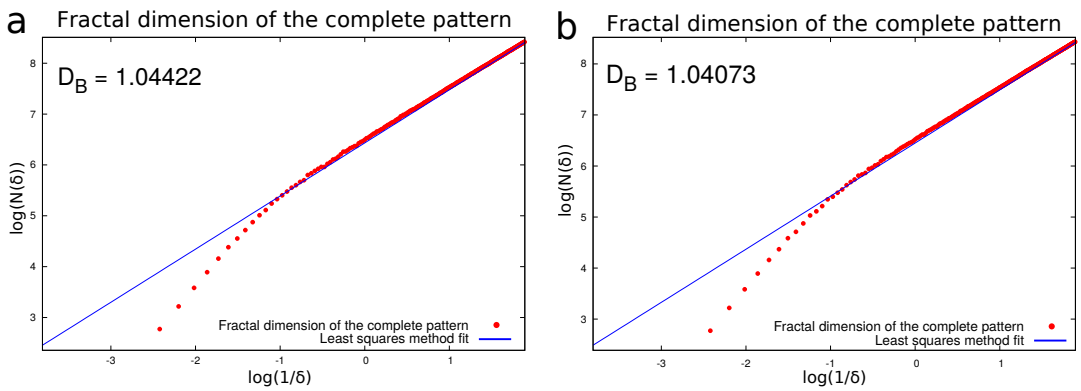

Figure 2.14.: Example plots employed in the determination of the box counting fractal dimension of our geometries. (a) was constructed from the complete diagonal design, see Fig. 2.13a, whereas (b) correspond to a complete centered arrengement, see Fig. 2.13b. Both plots stem from the application of the box counting method to the whole pattern.

Sec. 2.1.1.3 and Sec. 2.5. The plots used to compute D_B for designs a and b from Fig. 2.13 are shown in Fig. 2.14. The initial section of the plots were not used to determine the angular coefficients. Boxplots with the estimated ranges for the box counting fractal dimension based on design type are shown in Fig. 2.16.

In addition to the box counting dimension D_B, the VLAs of the patterns have also been computed. The venation volume, on the other hand, is not considered as property of our database, although it has been computed for every geometry after the 3D venation models of our designs were constructed and rendered. The reason is that we have purposedly chosen to keep the volume constant for all venations, primary and complementary. All volumes were forced to be equal to $150 \pm 0.5\,\mathrm{mm}^3$. This was achieved by modifying the Murray's law exponent n, a parameter which affects the volume and the surface area of the venations, but not their total length. The justification for this is that we wanted to compare the different geometries, by performing an statistical analysis on the instances of the database. A more detailed discussion validating this choice will be offered in Chapter 3.

The venation volume was estimated by the internal routines of the Netfabb program, a free, proprietary mesh editing software. The measurements could only be perfomed after the geometries were rendered with the OpenSCAD program, as discussed in Sec. 3.1. Additionally, the geometries had to be fixed before the Netfabb routines were used, since they require manifold geometries as input. The 3D geometry treatment was done according to the procedure presented in Sec. 3.2. Unfortunately, there is no way of estimating the total volume before rendering. Hence, n had to be adjusted by trial and error. Due to the time it took to render each geometry model, as well as the additional effort to fix them, we had to restrict ourselves to the current sample number. Were it not for this bottleneck, the database would contain more geometries. A thorough explanation about the construction and rendering of

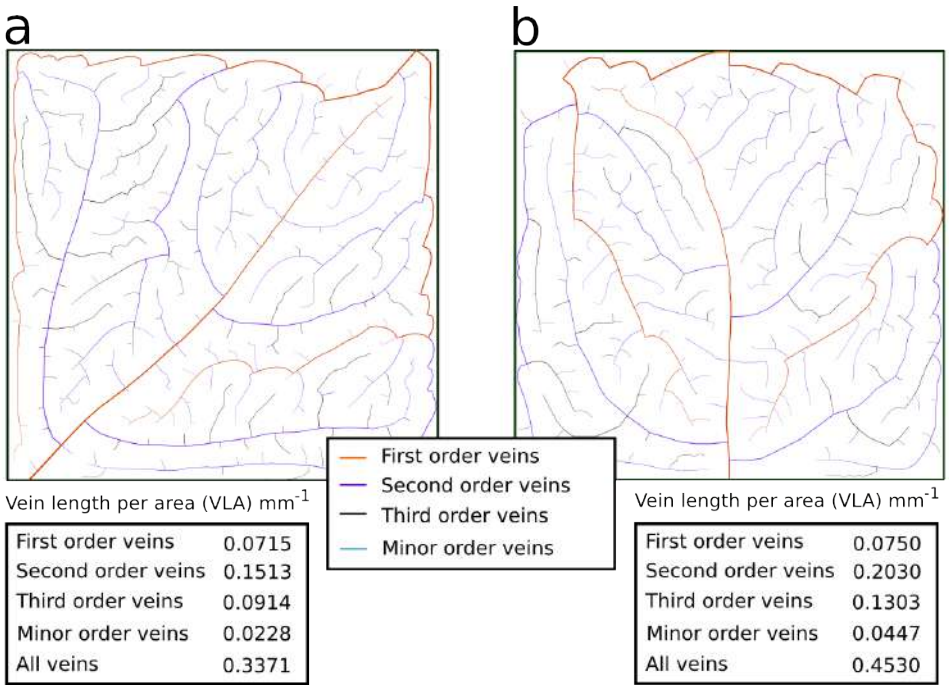

Figure 2.15.: Vein length per area (VLA) for two different designs. The VLA values shown take into all account both the primary and interdigitated venation.

the 3D models, as well as the procedure for estimating their volumes will be given in Sec. 3.2.

As for the VLA, we were able to determine it for each venation separately by including a simple recursive traversal function in our algorithm. Unfortunately, the final VLA is actually a little less than this estimate. There are two reasons behind this. The first is that minor veins are often enclosed by lower order veins after the rendering phase, specially when when the spacing between them is less than the vein diameters. This results in a lower VLA. Additionally, problematic rendering also requires some minor veins to be deleted, again resulting in a lower VLA. While the latter error source is much easier to estimate, since deleted vein's lengths are known, the former is not as easy to assess. Hence, a large error margin must be considered for each of these measurements.

We have also estimated the VLA based on vein order criterion for each venation. We have done this, because the separate study of VLA of different orders may reveal 'hidden' correlations [50] between these properties and dye concentration distribution efficacy later on, as discussed in Sec. 2.1.1.1. VLA mean values of centered and diagonal designs Fig. 2.13 have been determined and are shown in Fig. 2.15 based on vein order. We summarize the VLA values of the different designs and venation types in Fig. 2.17.

Ramification numbers were also computed for each venation with the values ranging

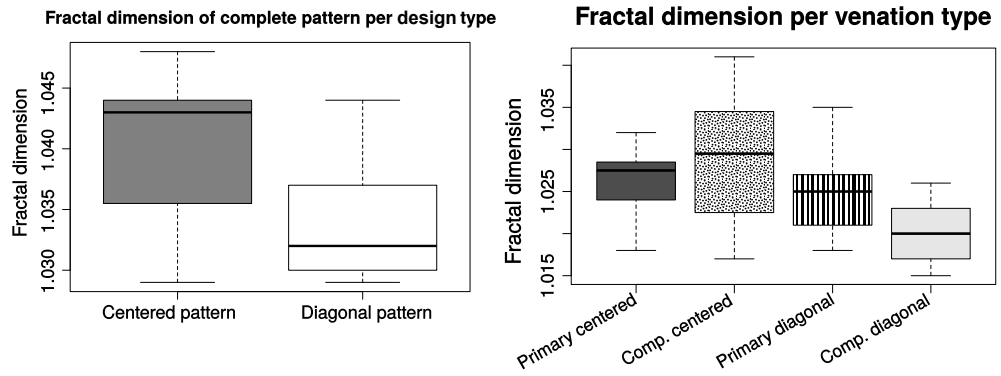

Figure 2.16.: Boxplots displaying the variation of fractal dimension per design type, right, and venation type, left.

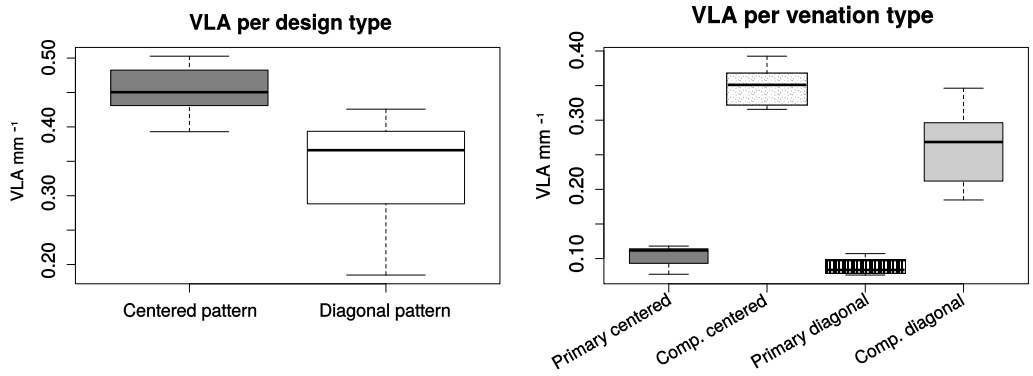

Figure 2.17.: Boxplots displaying the variation of VLA per design type, right, and venation type, left.

from about ten to over a thousand, see Fig. 2.18. The same type of recursive function used in the VLA computation was employed to determine the ramification number. Again, errors appeared due to the same reasons already discussed when the VLA measurements were considered.

The last thing we consider is the impact of a varying Murray's law exponent n. As already discussed, when we vary n for a given geometry, the final volume and surface area of the 3D model change, although the unidimensional traits of the design remain the same, see Fig. 2.19. Additionally, n for a given venation was chosen so that the volume of the venation would be as close to a fixed constant value of 150 mm^3 as possible. Since the choice of n varied from venation to venation, we also considered it as a discriminating property of the venations in our database. The impact of the n value choice will be analyzed in chapter Chapter 3, along with the results from CFD.

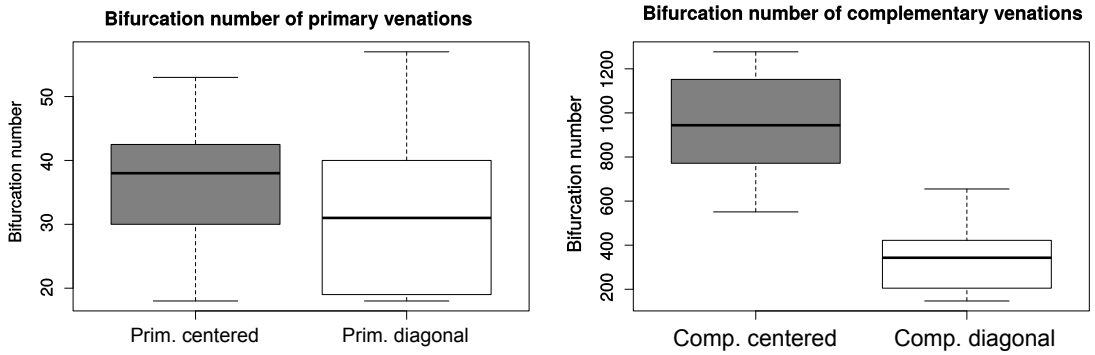

Figure 2.18.: Boxplots displaying the variation of bifurcation number per primary venation type, right, and complementary venation type, left.

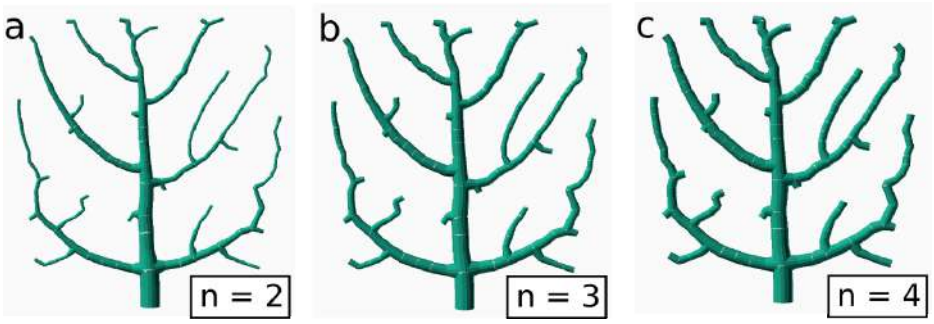

Figure 2.19.: The same geometry with varying Murray's law exponent n. Notice that the higher the value of n, the closer the end veins are to the midvein in diameter. Remember that Murray's law dictates the relation between parent and children veins at ramification points. The actual thickness of the venations has to do with the deliberate choice of diameter for the vein segment at the base of the midvein, a parameter specified by the user, which is the same for the three cases.

3. Solving the fluid flow problem through the generated geometries

After having constructed the venation-based designs presented in Chapter 2, we continue on our quest to assess whether the nature-inspired networks have a positive impact on reactant distribution across the targets. In this chapter, we focus on solving numerically the flow through the μ-FGPVs problems, which involve only a single phase, the dye solution [24]. Solving this problem was a great opportunity to learn many CFD techniques.

Next, we describe the most important steps taken to solve problem of fluid flow through the nature-inspired geometries employing CFD. Later, the results from the numerical methods are displayed in a concise fashion, see Sec. 3.4. All results were gathered in a database and an statistical analysis was performed. Finally, the highlights of the analysis are presented in Sec. 3.5 along with a discussion.

3.1. 3D venation model construction

The first step of the CFD procedure began with the construction of the geometries, see Chapter 2. There, we have shown how an out-tree data structure was generated as the output of the implemented algorithm, see Sec. 2.3. We have described how, in addition to the pointers to children vein nodes, each node stores its coordinates and the diameter of the vein segment spanning from the it to its parent, see Sec. 2.3. The data contained in the out-tree then allowed us to generate the 2D sketches of the channel arquitecture such as the one presented in Fig. 2.13. These sketches were constructed using OpenGL [86] and were exported in the Enhanced PostScript format using the GL2PS library [87]. In order to create the 3D models shown in Fig. 2.19, however, we have employed a CAD software. Next, we specify some of the basics of the program used.

The first aspect we had to consider is that the all of the channel designs generated by the algorithm were complex, meaning that manually constructing the 3D models from scratch using convetional CAD programs would be an extremely time consuming and inefficient process. As a result, most CAD programs, which were initially considered as an option, were ruled out. That is because they only offered

the option of manually constructing the designs. Nevertheless, one of the options, OpenSCAD, provided a mechanism for constructing and rendering the 3D models via a script file [88]. That is the reason why it was chosen. In contrast with the interactive modelers, OpenSCAD has its own descriptive language, which is interpreted by the software to render the models. Among other things, this feature greatly supported the automation of the whole process, since the OpenSCAD scripts could be immediately produced from our C++ venation program.

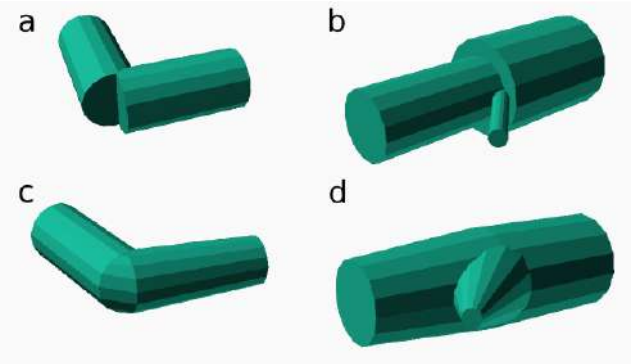

Figure 3.1.: Solid combinations used in the construction of the 3D venation models. The figure shows the problems encountered in the first attempts (a,b) and how the issues were solved (c,d).

The automation strategy, however, still presented a few challenges before it was successfuly implemented. The obstacles mainly revolved around the fabrication of 3D venation models with smooth surfaces and no sudden discontinuities from the solid building block collection available [88]. For instance, in our first attempts, we have tried using a simple combination of rotated and translated cylinders to generate the geometry. Many discontinuities appeared specially at the points of intersection of two different veins segments, see Fig. 3.1a. Furthermore, when ramification points were examined, we also observed a diameter discontinuity from a vein segment to the next in the same vein, see Fig. 3.1b. The inaccuracies similar to the one of Fig. 3.1b were immediatelly solved by employing conical frustums instead of cylinders. The radii of both extremes were then specified using the radius from the vein segment node and the radius from its parent node. Through this modification, smoother results were obtained, see Fig. 3.1d. Likewise, in order to fix the innacuracies of the type seen in Fig. 3.1a, we resorted to another conical frustum solid to fill the gap. Only the solid part which correctly fills the region is selected, see Fig. 3.1c. Although the final result still presents discontinuities, the overall product is considerably smoother. In particular, the cylinder base walls, which would block the flow, resulting in stagnant regions, are no longer present. Consequently, simulation of fluid flow becomes possible.

Other options exist within OpenSCAD to generate even smoother results [88]. Nevertheless, the functions used in the process take a substantially longer time to ren-

der. Due to the great complexity of some of the designs, the rendering time went from a couple of hours to more than three weeks when the functions which produce smoother results were applied. Another simplification analogous to the one just described goes along these lines: by changing a parameter, the user is able to choose the number of faces used to generate the solid [88]. Hence, the surfaces of the frustums could be smoothed further. Again, when the number of faces becomes to high, rendering the geometry unfortunately becomes unfeasible. As a result, we preferred simplicity, practicity and time efficiency over smoother surfaces.

Simple interdigitated models

Aside from the 3D venation models we are considering, we have also generated two additional simple interdigitated 3D models during this construction phase. One of the models was based on an interdigitated geometry, see Fig. 2.1c. The other one was inspired on the geometry proposed in the μ-FGPVs article [24], see Fig. 3.2. Both models were constructed manually, but since they are relatively simple, the generation process was straightforward. They were created with the intent of being compared later to the other venation models. That is because they offer the necessary contrast for a fair assessment of the performance of the generated venation models.

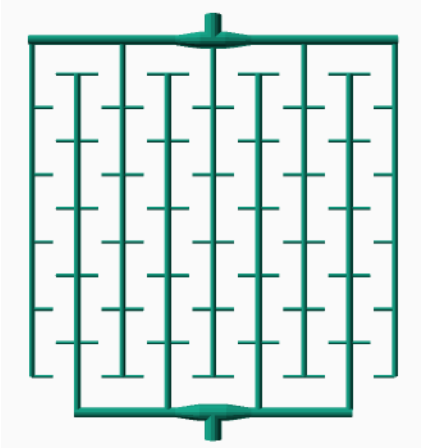

Figure 3.2.: Simple geometry inspired on the design presented in the μ-FGPVs article [24].

Final remarks

Here, we consider the last aspects of the 3D venation-based models before proceeding to the next phase of the flow problem solving process. Instead of rendering the vein channels as solids, we subtracted the channels from a solid square slab, see Fig. 3.3. As a result, the venation patterns become holes in the solid 3D rectangular prism.

The importance of these choices will become clear in the next sections. For now, suffices to say that the 3D slab represent the porous media while the hollow channels are where the fluid is injected and collected. The channels are where most of the advective transport occurs. Finally, the slab dimensions for all models were set as equal. Width and depth were defined as 50 mm and height as 3 mm exactly. Once the models were rendered, they were exported in the .STL (STereoLithography) file format. This format is accepted as input by the majority of the CAD, mesh editor and 3D printing programs. Note that the surfaces described by the .STL files are triangulated and, consequently, only triangle faces and the coordinates of their vertices are specified. At last, we highlight that .STL files describe only the surface of the 3D models and the volumes.

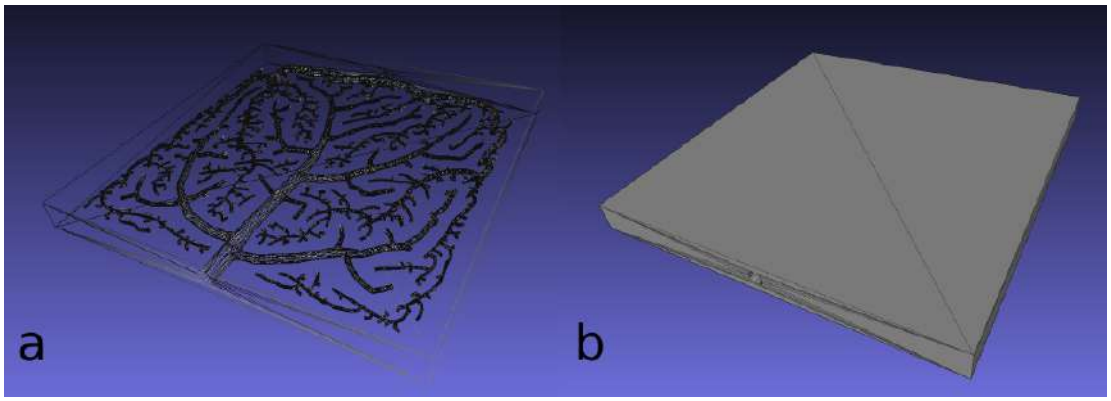

Figure 3.3.: Final 3D slab models constructed with OpenSCAD. The .STL files are being visualized with a mesh editing tool, MeshLab. In (a), the faces are transparent and only the edges are represented, while in (b) the faces are shown. The venation channels were subtracted from the slab and, therefore, are hollow.

3.2. Mesh construction

In this section, we finally enter the domain of the CFD procedure: by performing *pre-processing,* a step which mainly deals with mesh generation [32, 89]. Indeed, the actual first step when solving the equations which govern fluid flow computationally is mesh generation. The mesh is simply a representation of the geometric domain of a system by a finite set of nodes [32, 33, 90]. Variable values, e.g., pressure and velocity field values, are stored and computed at those nodes, as opposed to the continuum space of a system. Thus, this approach allows for the discretization of the governing equations and ultimately, makes solving the flow problems numerically possible. After the solution is obtained for all nodes of the mesh, the values at any other point can be inferred via interpolation [32, 91]. Note that mesh construction cannot be done carelessly, as the mesh considerably impacts the solution [32]. In fact, not only the mesh, but many other aspects may affect the results [32, 33].

Hence, estimation of errors and result validation are an essential part of the CFD process [32]. A thorough discussion of meshes, the CFD approach in general and result validation can be found in the many sections of Appendix B, along with an overview of the CFD software chosen to tackle our problems, OpenFOAM. Next, in this section, we discuss key points regarding mesh construction.

3.2.1. Geometry preparation

The first obstacle encountered when trying to generate meshes for our geometries was the sheer complexity of the designs. Again, as in the case of 3D model rendering, handling simple geometries manually is straighforward, but this approach rapidly becomes unfeasible when the complexity increases. In particular, OpenFOAM, offers an utility called *blockMesh*, which allows the construction of simple meshes, such as slabs, or cubes, for instance [89]. The constructing mechanism is cumbersome, however, as it is done by specifying *manually* some parameters in a text file [89]. Although possible in theory, generating the mesh of our designs using this method would be long and tedious. Fortunately, OpenFOAM provides another utility, *snappyHexMesh*, which generates meshes automatically from .STL or .OBJ files of a geometry [89]. As a result, construction of the meshes for our designs is more easily achieved. There are, in practice, some points we had to address, before successfully creating the meshes for our complex geometries with this method.

The main challenge before actually using the *snappyHexMesh* utility concerns the quality of the .STL file containing the geometry. Although OpenSCAD enables the automated generation of the 3D models of our designs, more often than not, it produces .STL files with an overabundance faces, many of which are totally pointless. Many faces have either virtually zero area or simply do not represent any important feature of the geometry. Moreover, a vast number of defects such as holes in the surface, intersecting faces and other problems which cause the geometry to be non-manifold are frequently present. This has to do with both the many details of the design and with the approach used in the construction of the 3D models, which was discussed in Sec. 3.1. These issues are usually harmless when the files are used for the purpose of 3D printing. On the other hand, when they are used as input to the *snappyHexMesh* utility without any pre-processing the outcome is disastrous as the output mesh is completely unusable due to the great amount imperfections. Therefore, fixing the .STL files before mesh generation is paramount.

To that end, two different programs were successfully employed: MeshLab [92], a free, open-source software and Autodesk MeshMixer [93], a free, proprietary software, see Tab. 3.1. MeshLab was chosen due to some of its filters, that when applied to the files, quickly eliminate the vast majority of the geometry problems. In particular, the quadric edge collapse decimation and merge close vertices filters proved to be extremely useful in drastically reducing the number of vertices, edges and faces of the models without any noticeable loss in details and primary surface features.

Moreover, it is possible to create scripts and run them from the terminal to fix the files, further automating the process. MeshMixer's tools, on the other hand, excel at manual refinement of the surfaces, making fine adjustments possible. Both tools played a major role in fixing the inacurracies of the .STL geometries.

Table 3.1.: Table summarizing the mesh editing tools employed in this work along with their usage.

Mesh editing tool	Software	Usage
MeshLab	free, open-source	Coarse refinement of .STL files
Autodesk MeshMixer	free, proprietary	Fine refinement of .STL files
Netfabb	free, proprietary	Determine venation volumes

Volume constraint

Another aspect worth mentioning is that we have chosen to maintain the volume of all the generated venations fixed and as close as possible to the 150 mm^3 value, as pointed out in Sec. 2.6. The value, to a certain extent, is arbitrary, but the choice to fix the volume of the channels for all venations, both primary and complementary, was not. This choice stems from the need to compare the geometries later on, when performing the statistical analysis. For some of the boundary conditions selected, we would not know whether a difference observed in the injected volume would be due to properties of the venations, e.g., bifurcation number, venation density, or due to the variation of total volume across the venations. Even in the case of a constant injection rate, we again would not know if the determined pressure profile would be due to differences in total volume of the networks. Hence, we placed the fixed volume constraint on the venations, assuming that differences observed in performance were consequences of differences in the geometric and topological properties of the channel networks, and not a result of arbitrary disparities in the total volume among them.

The decision to keep the total volume property fixed forced us to vary the Murray's law exponent n for both the primary and complementary venations until their volumes matched the 150 mm^3 value to a narrow margin of 0.5 mm^3. Unfortunately, estimating the final volume of the channel networks with enough accuracy during the venation generation phase would be a formidable undertaking. Many details which only become clear after rendering the 3D models are vary hard to predict. Truth be told, even after rendering and repairing of the .STL files, implementing an algorithm which determines the volume of the closed surfaces is no easy task. Luckily, another free mesh editing proprietary software, Netfabb [94], is distributed with internal functions which estimate the volume and surface area of the .STL files with precision much greater than the required. Hence, through trial and error, we modified n until the venation volumes fell into the desired range. This was a time

consuming undertaking, but a very important step towards the generation of our database. Lastly, we state that the simple interdigitated models were also forced to satisfy this constraint.

Inlet and outlet constraint

Another constraint we have placed on the venation geometries as well as simple interdigitated models was a fixed diameter condition for the inlet and outlet of the systems. The inlet and outlet are, respectively, the entrance and exit channels, meaning they are the only places where fluid enters and leaves the system. In the case of our geometries they can be ther vein at the base of either the primary or complementary venation. The diameter we have chosen for them was exactly 2 mm. Again, the reasoning behind this choice was to avoid the results and the statistical analysis to be affected by non-geometrical properties.

Final step of .STL file pre-processing

After we certified ourselves that the geometries met the constraints above, we proceeded to the step of fine manual repairing. In this phase, all defects of the .STL files were fixed with the Autodesk MeshMixer software. Later, after the manifold geometries were produced, we used MeshLab once more to further modify the files. We manipulated each geometry file to create .STL files for all three different regions of the system. Different regions are governed by different equations. For instance, the original geometry from Fig. 3.3 will represents the porous region, meaning the flow inside them will obey Eq. B.20. Files containing the venation models, see Fig. 3.4, where most of the advective transport occurs, are non-porous. Therefore, inside the venation channels, Eq. B.7 must be obeyed. The importace of creating an .STL file for each domain is that we would not be able assign the appropriate governing equations to each region during the CFD solving process otherwise. This is due to the way *snappyHexMesh* works. Fortunately, separating the regions into different files in this fashion greatly simplifies mesh generation down the road.

Finally, a last file containing the whole outer surface, which is literally a slab with no holes, is also produced, see Fig. 3.5. This last file is created with special care. The faces forming this slab are split into different groups, the patches. Defining the patches is vital since they facilitate the assignment of boundary conditions to different regions later on. Normally, the patches specified in this last file are the outlet, inlet, upper wall, bottom wall and side walls.

We stress that creating the venation files from the original file, as opposed to generating them from scratch, was crucial due to two reasons. The first one is because it saved us rendering time. The second, even more important aspect is that the venation faces from both the original and venation files would overlap. Were it not for this, *snappyHexMesh* would not be successful in defining a glitch-free boundary

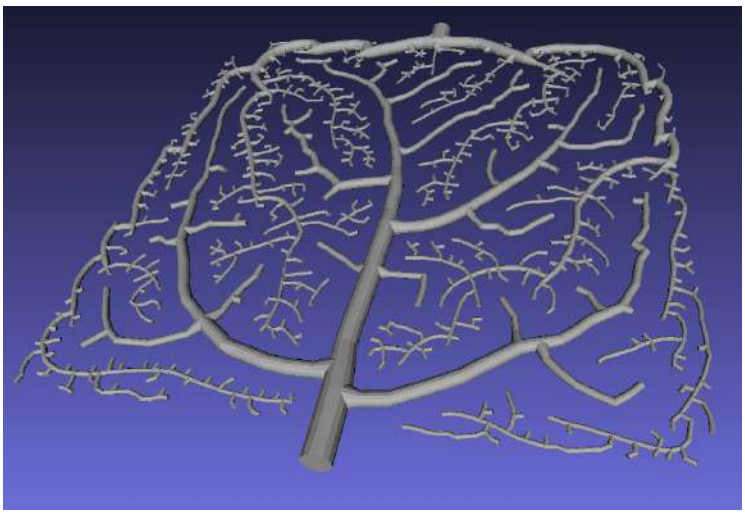

Figure 3.4.: Representation of both the primary and complementary venation. Two files, one containing each venation, were merged to produce this image.

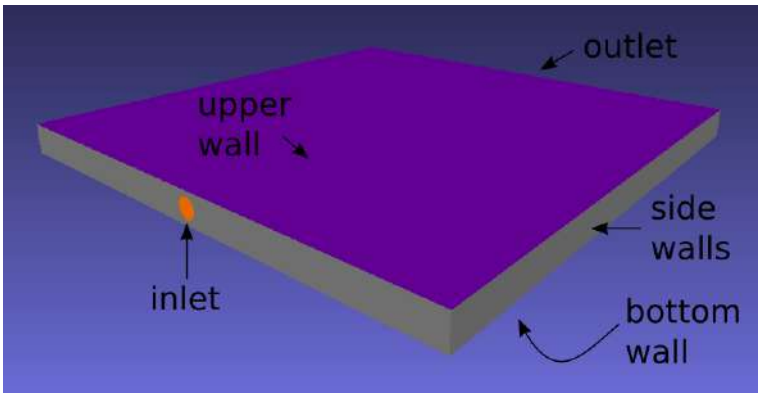

Figure 3.5.: Representation of the last slab file generated. The file is divided in 5 patches which are depicted in the figure.

between the channels and the porous medium. Needless to say that any attempt to solve problem the flow using problematic mesh would pointless.

The outer boundary file was also created from the original file, except for the inlet and outlet patches, which were naturally obtained from the venation files. The reasons for creating the files in this fashion were the same. One last thing worth mentioning is that we used a script to label each patch accordingly. This could be done because it is possible to name each patch, when the .STL geometries files are saved in the ASCII format. Naturally, it is possible for a single file to contain more than one patch. In particular, this was the case for the slab files, whichis the product of a file merging process. It is possible to see that all outer patches were combined into this single file, see Fig. 3.5.

Lastly, before actual mesh generation, we rescaled the coordinates of all .STL input

files. That is paramount since in OpenSCAD models are constructed in millimeter units, whereas OpenFOAM employs meter units [89]. We successfully used *surfaceTransportPoints*, another utility provided by the OpenFOAM toolkit, to accomplish the coordinate rescaling [89]. Moreover, we used the *surfaceCheck* tool, an utility which checks the validity of the mesh, to browse for problematic faces [89]. When not a even a single bad face was encountered, the .STL files were finally ready for the mesh generation phase.

3.2.2. SnappyHexMesh

In this last step, we conclude the mesh generation process. This was accomplished using s*nappyHexMesh,* an utility offered by OpenFOAM to construct meshes from .STL files in an automated fashion [89, 95]. This capability was so critical, that all of the processing through which the .STL files underwent was so that we could use this tool and obtain meshes that met the most important quality criteria.

The actual steps for mesh generation with *snappyHexMesh* are the following: creation of a background mesh using the *blockMesh* utility, extraction of the edgeMesh from each geometry file using the *surfaceFeatureExtract* utility and mesh generation with *snappyHexMesh* [89]. The parameters used by the other utilities, *blockMesh* and *surfaceFeatureExtract*, must be specified in their respective dictionary files, blockMeshDict and surfaceFeatureExtractDict, as discussed in Sec. B.1.2. A discussion on the purpose of both utilities, as well as their usage, are out of the scope of this text, but an in-depth explanation of why using them is important can be found in the OpenFOAM user guide [89]. Once these steps were taken, *snappyHexMesh* was finally applied.

Next, we briefly discuss the stages of the mesh construction process using *snappyHexMesh*, which can be divided into three parts [89]. The control parameters for each stage must be specified in their corresponding subdictionaries present in the *snappyHexMeshDict* dictionary file. Each stage can be set to on or off by modifying a boolean parameter at the beginning of the dictionary file. The first stage, the castellatedMesh phase, covers the generation of a very crude mesh, which serves a steppingstone to final mesh construction. The produced mesh is orthogonal and has a jagged aspect due to the construction process [89]. The product can be further refined in the next steps. The second stage, the snapping phase, has to do with vertex displacement at the mesh surface so that the resulting mesh resembles the input geometry as much as possible [89]. If this phase is successfully executed, it eliminates the jagged aspect. The mesh produced in this step is already fit for usage in our case. The additional stage concerns the addition of new layers at the surface to further smooth out and refine the grid close to the surface [89]. Additional information on the three steps and the parameters specified in the s*nappyHexMeshDict* file can be found in the OpenFOAM user manual [89]. Here, we only stress that it is paramount to specify each domain of the mesh, the venations and the porous region,

under the *refinementSurfaces* keyword during the castellatedMesh stage. We specify the regions according to the patch names they were given earlier. In particular, it is critical to specify the mode *inside* on the *cellZoneInside* keyword for each region, except the outer boundary patch. This has to be done, otherwise *snappyHexMesh* will not mesh all three regions.

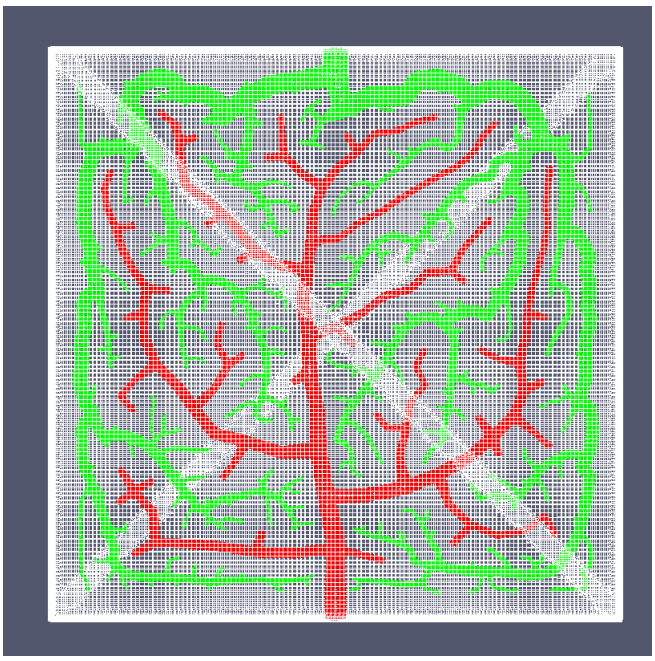

Figure 3.6.: Mesh generated with the *snappyHexMesh* utility. This mesh was visualized using the *paraFoam* utility.

Once the meshing process is complete, the final mesh can be checked using *paraFoam*, a post-processing tool that comes along with OpenFOAM toolkit [89], see Fig. 3.6. With the mesh created, we proceed to the second step of the CFD procedure, the *solving* phase [32, 89].

3.3. Solving the fluid flow problem

In this section, with a valid mesh finally at our disposal, we were able to run the OpenFOAM *solver application* for *transient* problems, *pimpleF*oam, and determine the flow through the system [89]. The reason why *pimpleFoam* was chosen was because it uses the PIMPLE algorithm, which combines traits from both the SIMPLE and PISO algorithms implemented in OpenFOAM [32, 33, 89]. SIMPLE and PISO are highly popular algorithms for solving the velocity-pressure coupled equations and are available in most CFD packages [32, 33]. In Appendix B, a thorough explanation of these algorithms and many other aspects regarding CFD and the finite

volume scheme is provided. The discussion is guided towards making the fundamental features of the PIMPLE algorithm comprehensible by the end of the chapter, in Sec. B.7.3.

We stress here that our goal was to determine the reactant concentration field ϕ profile after a time period. In principle, the velocity and pressure fields could vary for quite some time before reaching the steady state. In addition, they could possibly not even reach the steady regime during the selected duration. Naturally, that would influence the ϕ profile, as the tranport of the scalar field does depend on the velocity, see Eq. B.27 [31–33, 91]. Hence, we ended up using *pimpleFoam,* a *transient* solver [89]. We learned, however, that the flow very quickly reaches the *steady* regime, as will be shown in Sec. 3.4. Hence, the SIMPLE algorithm, which is implemented to solve *steady* problems in OpenFOAM, could have been applied to our case [89]. However, we did not know that beforehand. Moreover, there were no benefits to switching to SIMPLE, as the PIMPLE algorithm offers all features SIMPLE does, including the possibility to employ under-relaxation [96], see Sec. B.6.7. Therefore, we kept using *pimpleFoam* to solve the pressure-velocity coupled equations and the *scalarTransport solver* to solve the general transport equation for ϕ, see Sec. B.3. The instructions for the *scalarTransport* solver were introduced at the end of the *controlDict* file, under the *system* folder of the case. The solving order is clear, first the pressure-velocity coupled equations were solved and then the transport of the scalar ϕ was calculated at each time step [33]. This is done because the coupling is one-way. Next, we discuss important aspects of the flow problems, including the boundary conditions and the input parameters to the *solver*.

3.3.1. Porous Medium region inclusion

Here we discuss how the inclusion of the porous region is performed. In OpenFOAM the porous properties of a region are simulated by adding a source term to the momentum equations, see Sec. B.4. That is accomplished by including a *fvOptions* dictionary in the *system* folder of the case, with a porosity subdictionary in it. The most important property that must be specified in our case is the tensor $\boldsymbol{d}$, see Sec. B.4. For a homogeneous porous medium, such as the media found in this work, all entries of $\boldsymbol{d}$ are equal and the tensor acts as a scalar. In this case, $d = \frac{1}{k}$, where k is the *permeability*. Thus, we see that the source term is, in fact, the Darcy term for porous media. Additional details of how the addition of this source term translates into the well-known governing equations of porous media can be found in Sec. B.4.

Finally, in the channel regions of the system, the flow is governed by the incompressible Navier-Stokes equations, see Eq. B.7. Moreover, the mass conservation restraint, given by the continuity equation for incompressible fluids, see Eq. B.6, is also satisfied in both the porous media and the channels. Naturally, all body forces were neglected, meaning the action of gravity on the fluid CVs $\rho\boldsymbol{g}$ was disregarded.

3.3.2. Input parameters

The fluid and porous properties specified as input for the solver were the kinematic viscosity of water ν, set as 1×10^{-6} m^2/s, the effective diffusion coefficient for the agar hydrogels 5×10^{-10} m^2/s and the *permeability* of the medium, set to 1.77×10^{-11} m^2 [18, 24]. As for the time step length, it was set to 0.2 s for all models, a feat only possible due to the use of *pimpleFOAM* [96], given the considerable step size. Moreover, we defined the initial and ending times to 0 and 3 s respectively. In this short time span no significant changes in the velocity nor pressure profiles were observed. Later, in order to speed up the solution for the concentration field, we have used a version of the *pimpleFOAM* solver modified to copy the steady solution for the pressure and velocity fields to the next time step and then solve the transport problem for ϕ. In this manner, we were able to determine the evolution of the dye profile for another 3600 s in under 10 min of processing. Finally, we set a threshold value of 5×10^{-4} for the pressure and velocity residuals at the end of each inner iteration. The threshold for the residuals at the end of the time step was set to 1×10^{-6} for the pressure and 1×10^{-5} for the velocity, meaning the solutions at a time step would only be accepted when the residual values went below these number, refer to Sec. B.6.6 [32, 96]. We highlight that the residuals are all normalized so as to make the analysis problem independent [89].

3.3.3. Boundary conditions

The chosen boundary conditions (BCs) for the velocity field $\boldsymbol{u}$ and the pressure field p are depicted in Fig. 3.7. They follow from our discussion in Sec. 1.3, noting that the flow regime of our problem is laminar. We set a no-slip BC for the velocity field, that is, $\boldsymbol{u} = 0$, since the patches are fixed, and a zero gradient BC for the pressure, $\boldsymbol{\nabla} p = 0$, at the upper, bottom and side walls of all the models. Moreover, we have chosen to solve the flow problem using two different groups of BCs for the inlet of each model. The first BC set was a constant volumetric flow rate Q of 10 μl/min and a fixed pressure at the oulet [24]. The second group of BCs chosen was a fixed pressure at both the inlet and oulet of the system, with a pressure difference Δp of 0.02 Pa. This value for Δp was chosen based on the solutions with constant Q. Finally, the BCs for ϕ at the walls were all set to zero gradient and at the inlet we have chosen a contant concentration value of 5 mol/L. Naturally, the flux in and out is conserved in all cases.

3.4. Results

In this section, we present the most important results obtained from the CFD numerical solutions. We have also performed a statistical analysis with these results, which will be shown in this chapter later along with a brief discussion.

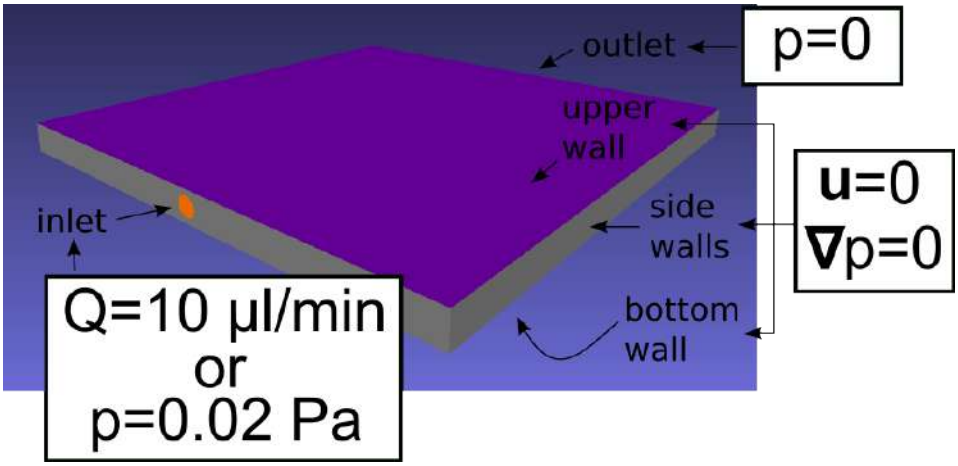

Figure 3.7.: Boundary conditions for $\boldsymbol{u}$ and p at the different surface patches of the mesh.

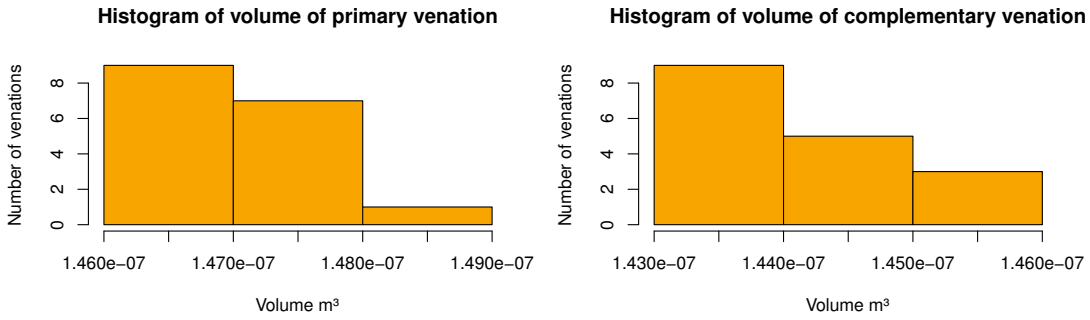

Figure 3.8.: Histograms of primary and complementary venation volumes. The mean mesh volume values of the primary and complementary venations are 147.5 and 144.5 mm^3 respectively.

We begin highlighting that the number of cells in most meshes of this study was in the 850000-1000000 range. At first, this mesh resolution was chosen due to it being the lowest resolution which preserved the main geometrical features of the models. Moreover, hardware limitations also prevented us from further refining the mesh enough to perform a convergence analysis. We were able, however, to refine the mesh of one of our models, producing an approximately 4000000 cell mesh. A comparison between the coarse and refined mesh is shown in Tab. 3.2. Although the difference in percentage between the mean solutions values for p, u_x and u_y are acceptable, the difference in percentage for u_z is still very large. Even considering the fact that the magnitude of u_z is very small, about 4-5 orders of magnitude lower than the other components due to the symmetry of the problem, further refinement of the mesh is urgent. Ideally, a separate convergence study for each of the models is required to validate the results [32]. We will further refine the mesh and perform

the appropriate convergence analysis in future work. That will be possible since, recently, a workstation has been purchased by our group, see Chapter 5.

Table 3.2.: Comparison between solutions in a coarse and a refined mesh for the same geometry. The differences between the solutions are not negligible.

mesh	cells	p	u_x	u_y	u_z
coarse	946796	8.61E-010	9.42E-012	1.06E-011	-3.67E-016
refined	3820824	8.61E-010	9.38E-012	1.05E-011	-5.05E-016
difference	75.22%	0.0084%	0.44%	0.34%	-27.5%

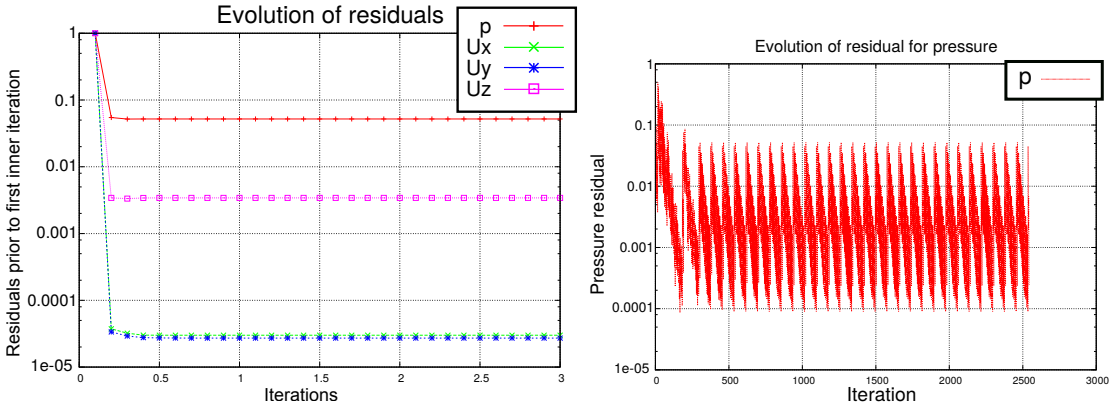

Figure 3.9.: Residuals prior to the first inner iteration vs. time, left, and all residuals prior to every inner iteration vs. the iteration step.

Residuals that stem from the use of iterative solvers, see Sec. B.6.6, can offer an idea about the magnitude of the iteration errors [32]. Ferziger states that when the residual norms fall by a given order of magnitude, it is likely that the iteration error falls by a comparable amount [32]. Iterative solvers in OpenFOAM yield normalized residual values during the solving process. We display the output residuals prior to the first inner iteration at a time step for p, u_x, u_y and u_z, see Fig. 3.9. The pressure field requires a much greater number of iterations before the threshold value criteria is satisfied. Thus, we showcase all pressure residuals prior to every inner iteration for p, see Fig. 3.9. The periodic nature of the pressure residual graph is due to the use of the PIMPLE algorithm [96], see Sec. B.7.3. Each cycle correspond to a time step. Initial residual values for the u_x and u_y components are acceptable, below 10^{-4}. The magnitude of the p and u_z residuals, however, are still far from ideal, being above 10^{-3}, see Fig. 3.9. In particular, the value for the initial pressure residuals are in the 10^{-2} range could be considered high. Nonetheless, the residuals remained stable and did not grow. Moreover, analysis of the continuity errors have shown values in the 10^{-10} to 10^{-11} order range, near the 10^{-12} reference used by many as the machine epsilon. Hence, we took the results as acceptable. Ideally, however, all residuals should be below 10^{-4} for proper convergence.

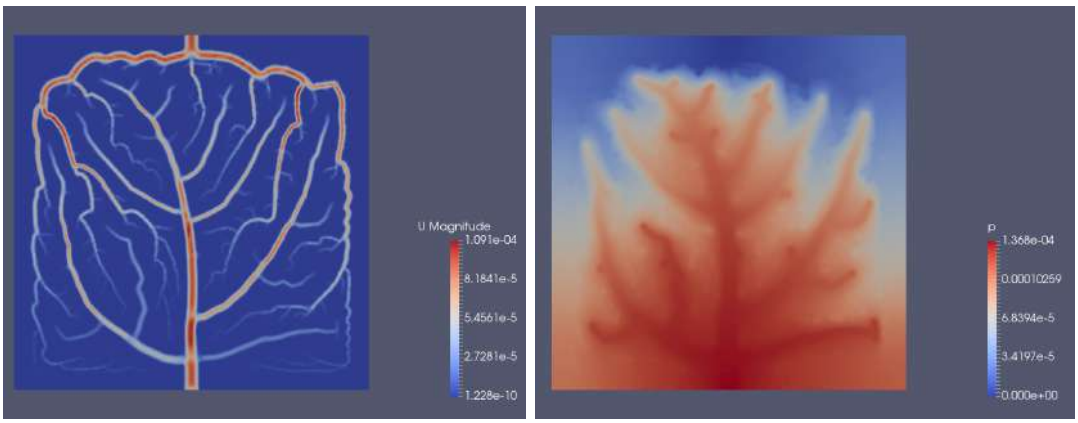

Figure 3.10.: Velocity and pressure profiles for one of the centered geometries.

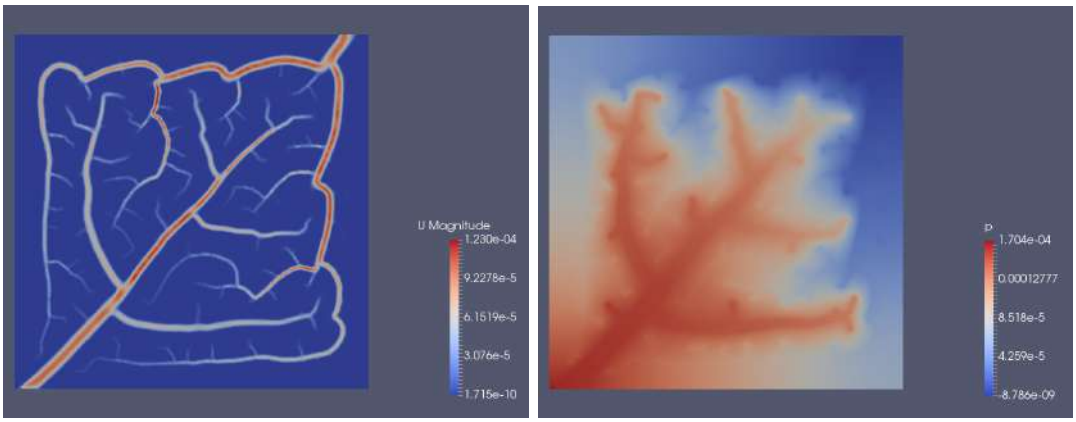

Figure 3.11.: Velocity and pressure profiles for one of the diagonal geometries.

Another aspect we have not mentioned earlier is that the channel volumes in .STL files and in the meshes were not the same. That has to do with the way *snappyHexMesh* generates the mesh. The venation volumes in the meshes were always lower than the volumes defined for the .STL files. As a rule, complementary venations exhibited lower volumes than primary venations. Finally, primary and complementary venations, respectively, had volumes that could be found in a narrow range of $3\,\mathrm{mm}^3$. That can be seen in their corresponding histograms, see Fig. 3.8.

We can now display examples of the determined velocity and pressure profiles for the centered, diagonal and simple models, see Fig. 3.10 and Fig. 3.11 respectively. The darker the blue/red, the lower/higher the field values are. All of the profiles shown represent the solution at the $z = 0$ plane for the constant Q BC at the inlet. The corresponding concentration profiles for centered and diagonal models after a time t equal to 3600 s may also also be observed in Fig. 3.12 and Fig. 3.13. Notice how the highest values of ϕ lie inside and in the surroundings of the channel pattern connected to the inlet. That occurs due to very low velocity values inside the porous medium, where most of the transport has a diffusive nature. The transport inside

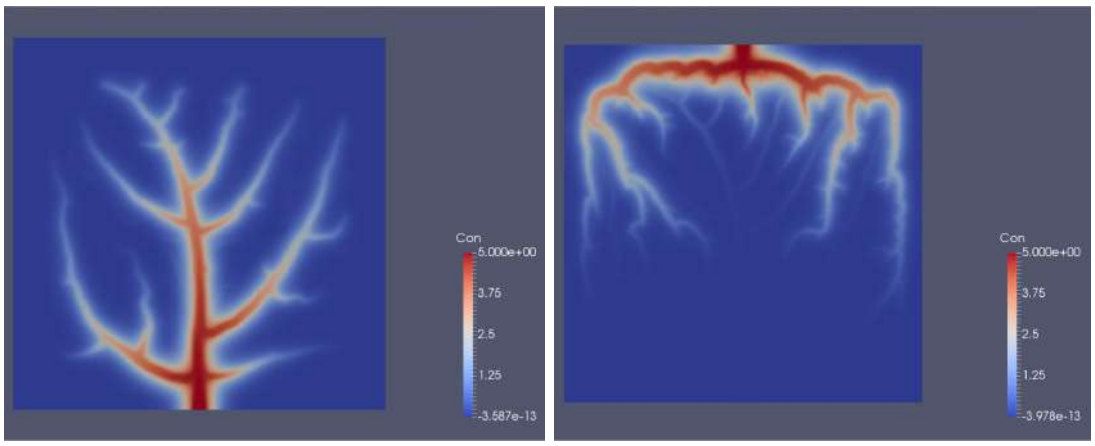

Figure 3.12.: Concentration profile at $t = 3600\,\mathrm{s}$ for the centered geometries when the inlet is connected to the primary or the complementary venation.

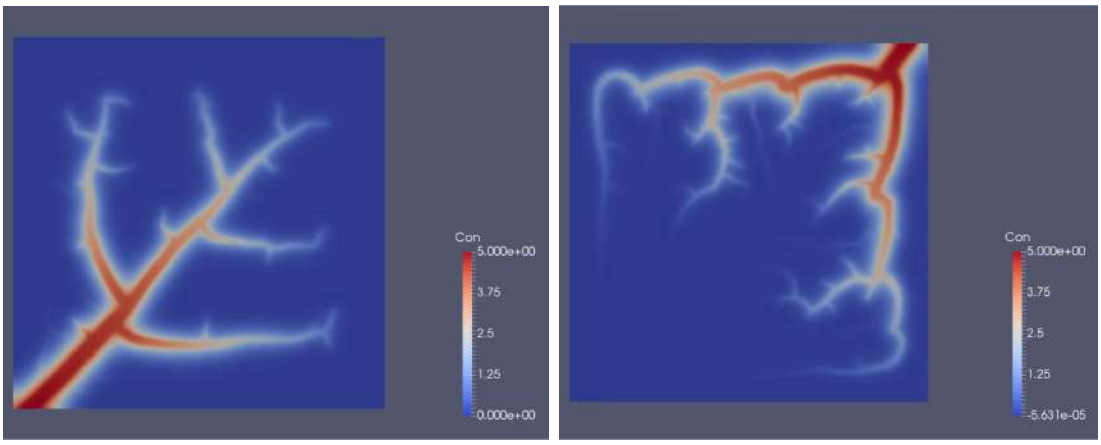

Figure 3.13.: Concentration profile at $t = 3600\,\mathrm{s}$ for the diagonal geometries when the inlet is connected to the primary or the complementary venation.

the channels, however, are dominated by advection. Thus, we can observe relatively high concentration values inside the inlet pattern rather quickly, while we must wait much longer to see a change in concentration in its immediate surroundings. We quantified this by tracking the concentration along some lines in the $z = 0$ plane which are perpendicular to a channel. Concentration profiles along lines crossing first and second order veins can be seen in Fig. 3.14. Notice the sharp decay in the porous region in the immediate surroundings of the channels.

A useful quantity for characterizing the performance of the geometries is the volume coverage percentage, defined simply as the volume of the system above a threshold concentration value over the total volume:

$$\text{Volume coverage} = \frac{\text{Volume above a concentration threshold in mol/L}}{\text{Total volume}}$$

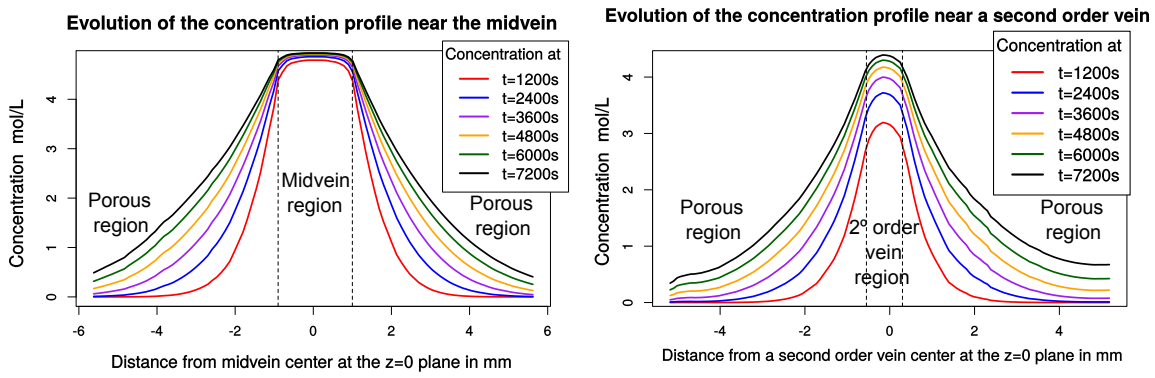

Figure 3.14.: Comparison between the concentration profiles at different times over a line cross-sectioning a midvein, left, and a second order vein, right. Notice the typical diffusive behavior in the porous regions, in contrast with the advection in the vein region. The concentration in the veins already have a high value for the first measurements.

A similar criterion was used in one of our references, where the area coverage percentage was defined [24]. This type of criterion is appealing because it offers some information about the distribution profile of the dye. In case most of the dye is confined in a small volume, the geometry will tend to score less than in a scenario where the dye is well distributed in all directions. Naturally, this will also depend on the threshold value. In case the threshold chosen is high, geometries with confined dye may in fact have greater coverage than the others. That can be seen in one of our results shown in Fig. 3.15. When high threshold values are used, the complementary centered venation model type, which concentrates a great amount of dye near the inlet, performs much better than the others, see Fig. 3.12. When lower thresholds are employed, on the other hand, the primary centered venation models have a better performance. This becomes even more evident we we display the same results in another fashion, see Fig. 3.16.

The differences between results with the different BCs at the inlet were not negligible. Although some models did present some a slight better performance when constant Q at the inlet was used, their overall performance was quite similar, see Fig. 3.17. On the other hand, when the pressure at the inlet was fixed instead, differences between the models were striking under the same 3600 s time period, see Fig. 3.18. In part, that is likely connected to the fact that the amount of dye that entered the system was equal for all models when the constant Q BC was employed, but not equal when the constant pressure drop BC was applied. Due to these variations, we proceeded with an analysis of the models under the constant pressure drop BCs for a longer time period, see Fig. 3.15. These results further confirmed the observed perfomance diversity between the different models. Motivated by these results, in the next section, we will present an analysis of the solutions with the constant

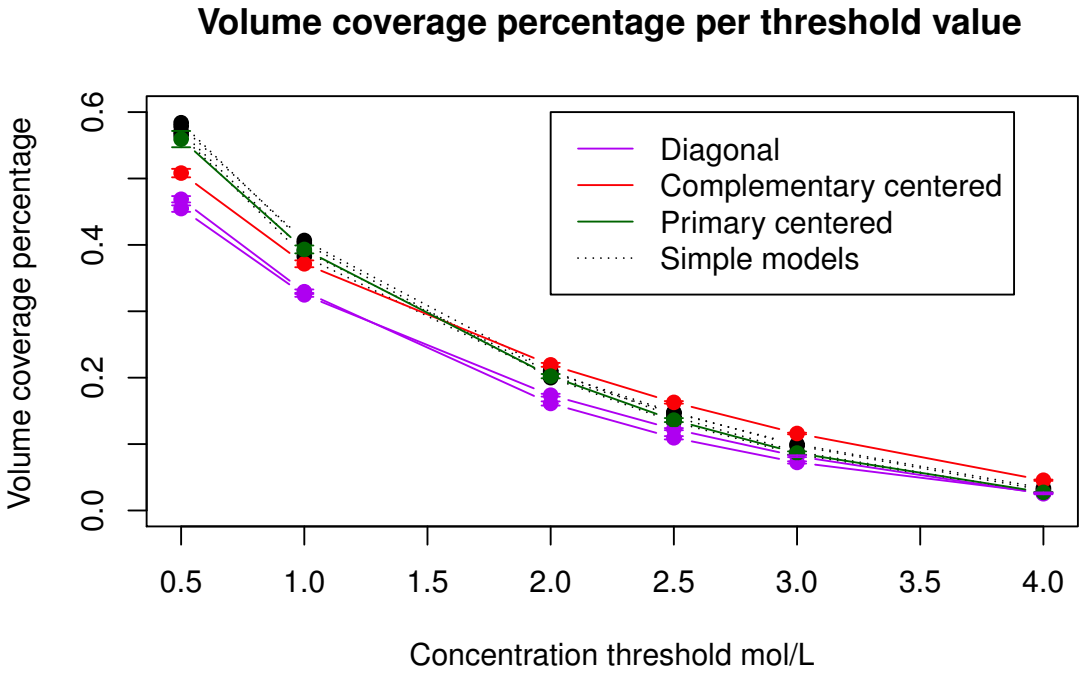

Figure 3.15.: Volume coverage percentage per threshold value graph for the contant pressure BC at the inlet after 7200 s. Diagonal models are represented in purple, the models where the primary centered venation is the inlet channel network is depicted in green, and the ones where the complementary centered venations is connected to the inlet are shown in red. The simpler models are represented by dashed lines in black. The concentration at the inlet is 5 mol/L.

pressure BC at the inlet.

Finally, we present the variation of the mean velocity components and pressure fields during the initial 3s period for one of the models, see Fig. 3.19. Notice that the values remain practically constant during the whole period, except for u_z, which rapidly converges at the end, indicating that the steady solution has been reached. Analysis of all models have shown a similar behavior. Furthermore, we observed the pressure and velocity profiles along many lines in this time range and observed no variation. Hence, we felt justified in the chosen procedure described earlier in Sec. 3.3.2 of halting further computations of velocity and pressure fields and determining only the ϕ profile in subsequent time steps. This approach is extremely important as it considerably sped up the solving process and allowed us to obtain a significant amount of data.

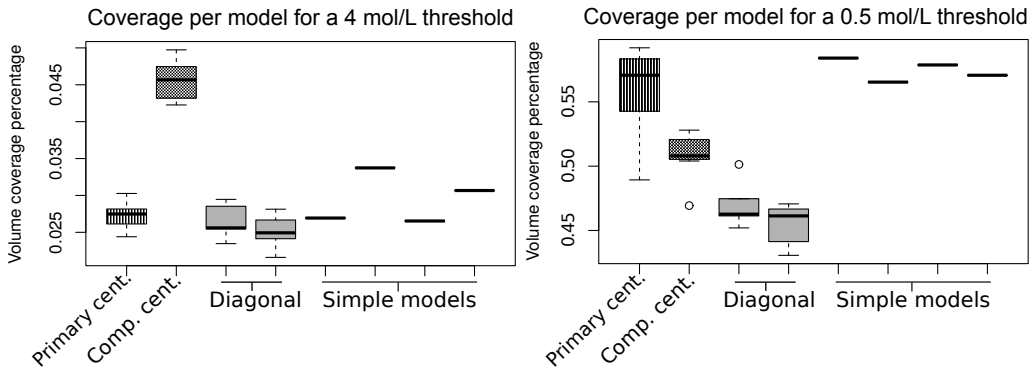

Figure 3.16.: Boxplots of the coverage percentage per model type for the contant pressure BC at the inlet after 7200 s. Only the Boxplots for the thresholds of 4 mol/L and 0.5 mol/L were displayed. The concentration at the inlet is 5 mol/L.

3.5. Discussion and statistical analysis

In this section, we discuss our findings after probing the data for correlations. We also offer some possible interpretations for the results.

The main analysis we performed was a search for significant correlations between volume coverage percentages with varying threshold and other attributes we computed. This was achieved through the use of correlograms. A correlogram is a simple and comprehensive method of displaying correlations between different attributes of database in graphical form [97]. In this way, significant correlations between two properties can be quickly spotted. In order to generate the correlograms, we employed the *corrgram* package of the R programming language. The correlograms for the different centered models can be seen in Fig. 3.20 and Fig. 3.21. The Pearson correlation at each matrix element is represented with colors: dark blue indicates high positive correlation, dark red indicates anticorrelation and light colors and white indicate there is no significant correlation between the attributes.

The first significant correlations we were able to single out from the data are the ones between the coverages with lower threshold values and the average dye concentration, see Fig. 3.20 and Fig. 3.21. That came as no surprise, since high average concentration indicates a higher amount of dye in the slab, which, in turn, is expected to produce an increase in coverage percentages. In addition to this, when we considered the actual amount of dye that entered the slabs for all models, we found that the poor performance of the diagonal models could be attributed to this, as a considerably lesser amount of dye enters the slabs in these, see Fig. 3.15. Likewise, the good performance of the simple models can also be attributed to this correlation, as an appreciably greater amount enters slab in these models. The second correlations we address are the ones between the coverages themselves: the closer two threshold values are, the stronger the correlation between coverages. Again, this is expected, as this is an indicator of linearity, meaning that a small change in

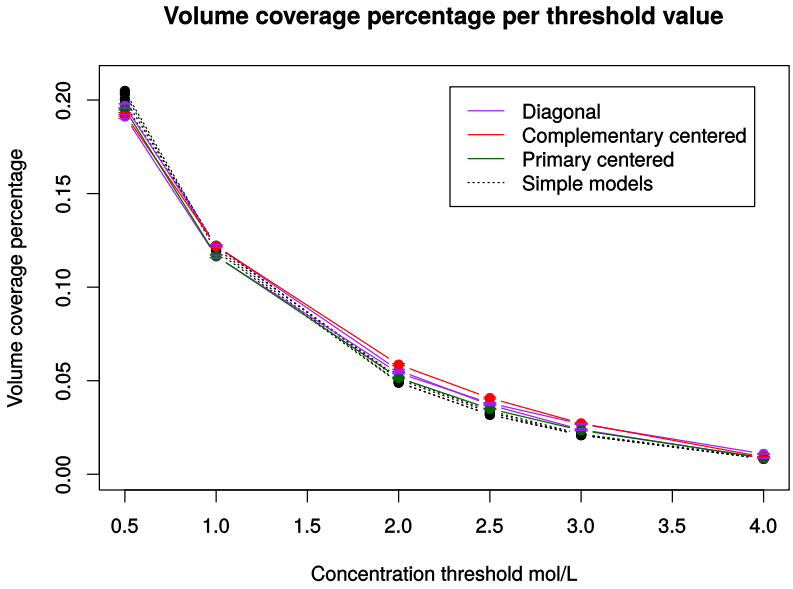

Figure 3.17.: Volume coverage percentage per threshold value graph for the contant Q BC at the inlet after 3600 s. Primary networks of simple models performed slightly better than other models when the 0.5 threshold was used. Primary centered and complementary diagonal venations, in sequence, performed better than remaing ones. We verified this via one t-tests, see Sec. 1.4.

threshold value will not produce a completely different coverage behavior.

Next, when the complementary centered models were considered, meaning that the complementary venation was the one connected to the inlet, we observed strong correlations between most coverages and the Murray's law exponent n for the inlet venation, see Fig. 3.20. This is an interesting finding, as Murray's law, which originates in the context of minimization of work, also seems to play a role in the increase in coverage percentage of these geometries. There may be a limit to this, however, for when we observed the same correlations for the primary centered venations, the correlations found were much weaker, see Fig. 3.20. An explanation for this may be that n for complementary venations varied between 1.63 to 1.95, while n for primary venations ranged from 2.48 to 2.82, refer to Sec. 2.6. This means that the end channels in the case of complementary venations are significantly thinner than the primary ones. Perhaps, when the channels are very thin, an increase in channel diameter may improve the coverage, but when the channels are thicker, there may be no longer a benefit from this increase. Naturally, the Murray exponent of the inlet venation also correlates strongly with the amount of dye in the slab, which, in turn, correlates strongly with the coverages. Finally, when the diagonal models were considered, the same correlations were again weak. We speculate, however, that the reason behind this could be the same reason behind their overall poor perfomance

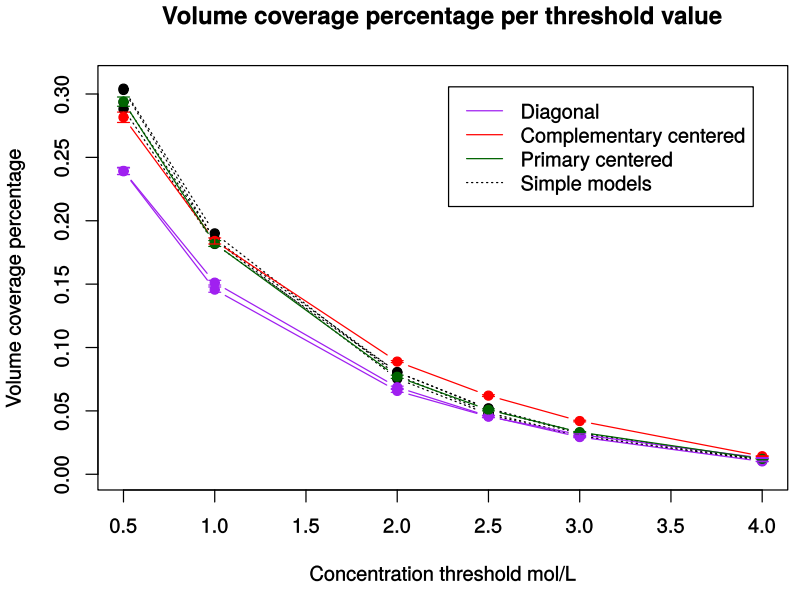

Figure 3.18.: Volume coverage percentage per threshold value graph for the contant pressure BC at the inlet after 3600 s. Differences between the models could already be observed. Diagonal models had a lower performance. That is confirmed by analyzing the results after 7200 s, see Fig. 3.15.

when compared to other models. Moreover, the average distance between the primary and complementary venations is greater for diagonal models than for centered ones.

A similar discussion may also be applied to the correlations found between the low threshold value coverages and fractal dimensions, bifurcation numbers and some of the computed venation lengths per area (VLAs). Specially when the diagonal models are not considered, see Fig. 3.22 and Fig. 3.23. Again, high threshold values coverages behave in a generally different way then low threshold value ones. At times, in particular when we the primary centered model is considered, strong anticorrelations may be observed between many properties and the high value threshold coverages while a simultaneous strong correlations between the same properties and low value threshold coverages exist, see Fig. 3.22 and Fig. 3.23. All things considered, we feel that more conclusive remarks can only be made once additional data is gathered and assessed.

Finally, when we evaluated the diagonal model correlograms, we observed a difference between the complentary and the primary venation models. While significant anticorrelations between many properties and high threshold value coverages were present in the primary models correlogram, the complementary models correlogram exhibited only weak correlations and anticorrelations between all coverages and other properties, see Fig. 3.24 and Fig. 3.25. The strong anticorrelations in the primary

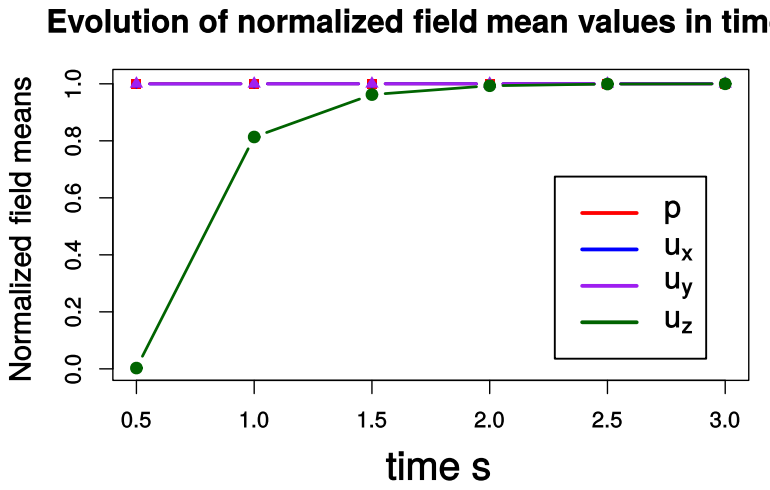

Figure 3.19.: Pressure and velocity component field mean values normalized by the last value. All components quickly converge at the end of the 3 s time period.

models, however, again seemed to be dominated by the total dye attribute, which also had strong anticorrelations with the same attributes. When we examined the low threshold value coverages, however, we found that they had weak correlations with the other attributes. Possibly, an increase in VLAs, fractal dimension or bifurcation number do indeed have a positive impact on the low threshold coverages. Determining what are the properties that have an effect on them, however, is a still difficult at this point. Hence, we believe that more data and further studies are necessary before we obtain a definitive answer.

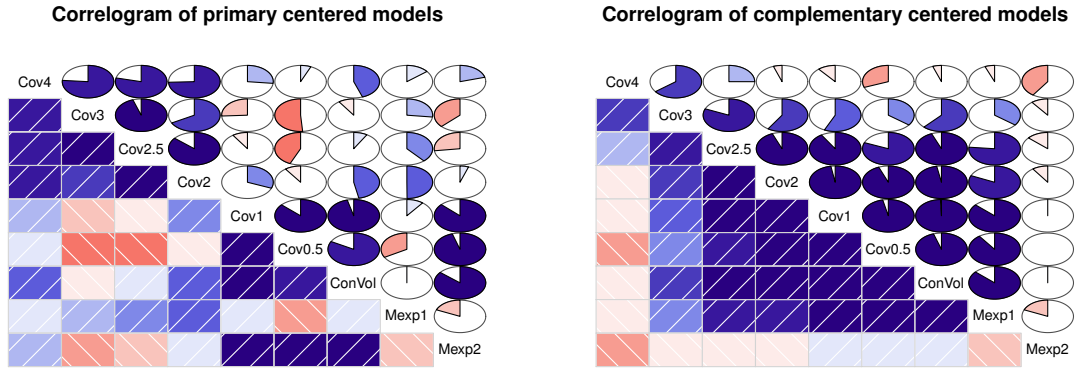

Figure 3.20.: Correlogram of the primary and complementary centered models, when the constant pressure BC at the inlet was used. Data was extracted at $t = 7200\ s$. Dark blue represents a correlation close to 1, while dark red represents a anticorrelation near -1. White represents no correlation: pearson coefficient of 0. The matrix is simmetric and the upper and lower triangular regions are simply displayed in a different fashion. Cov stands for volume coverage percentage and the number at its side is the threshold. ConVol is an abbreviation for the average dye concentration. Mexp is the murray exponent and 1 and 2 correspond to inlet network and outlet network respectively.

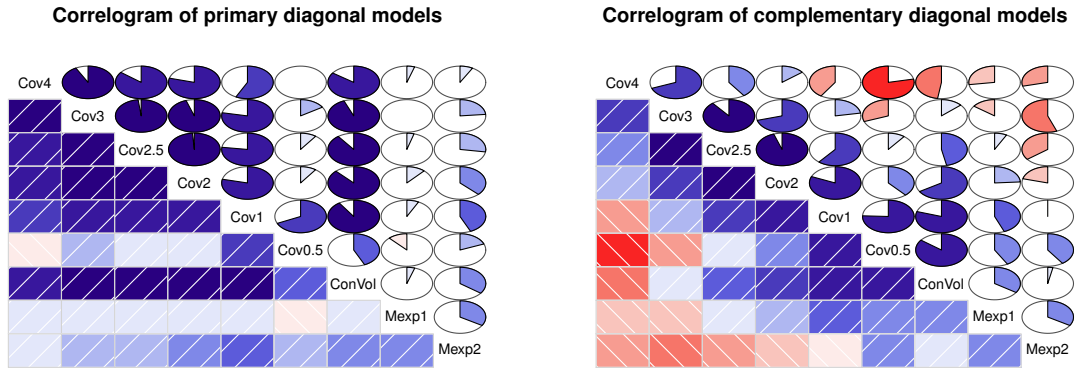

Figure 3.21.: Correlogram of the primary and complementary diagonal models. Both were constructed in a fashion similar to the ones in Fig. 3.20.

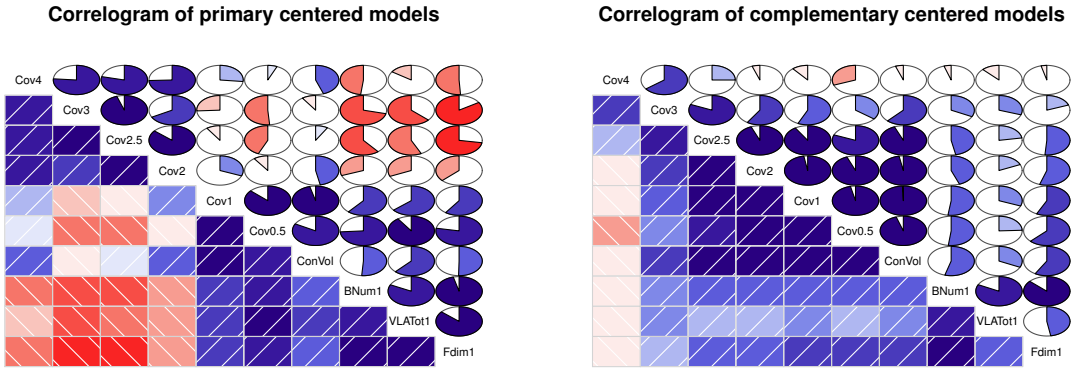

Figure 3.22.: Correlogram of the primary and complementary centered models. Both were constructed in a fashion similar to the ones in Fig. 3.20. Here, however, BNum1 stands for bifurcation number of the inlet network, VLATot1 corresponds to the total VLA, considering all vein orders, of the inlet network and, finally, FDim1 stands for its fractal dimension.

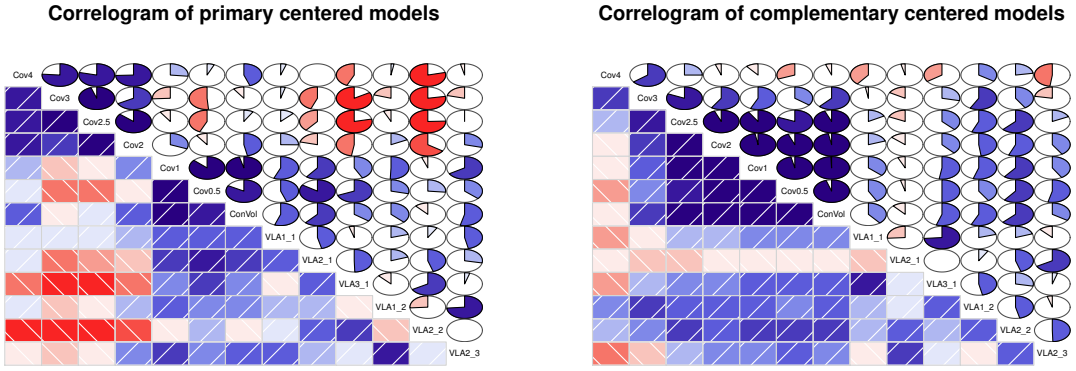

Figure 3.23.: Correlogram of the primary and complementary centered models. Only the VLA attributes are considered together with the coverages in these correlograms. VLA1_1 stands for the VLA of first order veins of the inlet network, while VLA1_2 corresponds to the VLA of first order veins of the outlet network. Other traits follow the same nomenclature but correspond to the secondary and tertiary vein VLAs respectively.

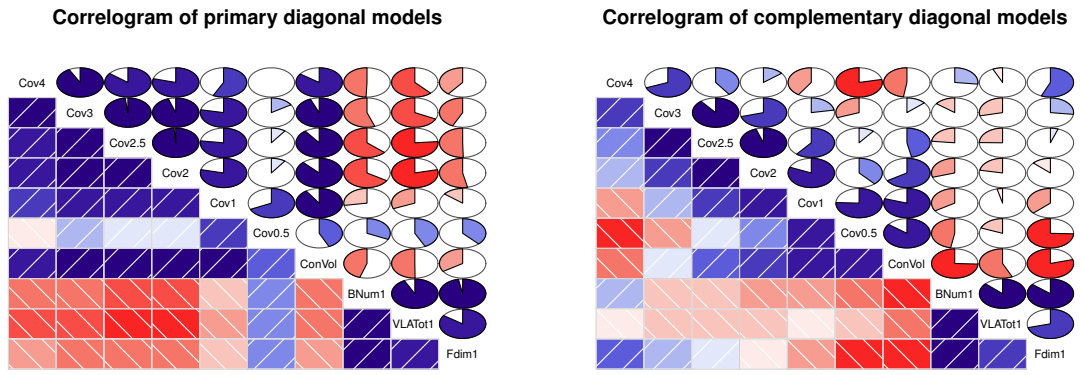

Figure 3.24.: Correlogram of the primary and complementary diagonal models. Both were constructed in a fashion similar to the ones in Fig. 3.22.

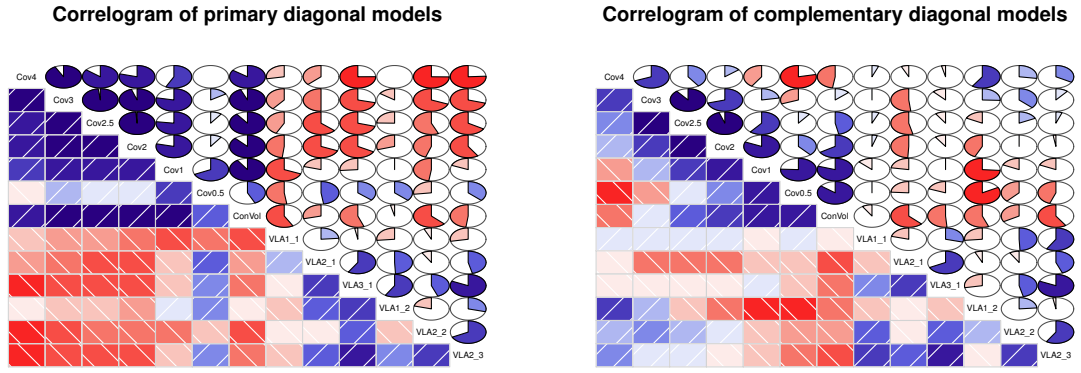

Figure 3.25.: Correlogram of the primary and complementary diagonal models. Both were constructed in a fashion similar to the ones in Fig. 3.23.

4. Experimental models

In this section, we discuss our attempts to validate the results from the CFD step via experimental models. The models built were based on the experimental set up of the μ-fluidic gel photovoltaics (μ-FGPVs) target application, the variation of the *dye-sensitized solar cells* (DSSCs). Although validation was not yet succesful, we strongly felt it was necessary to briefly showcase our efforts towards it, as validation of the results from CFD is an important part of the modeling procedure [32]. Next, we present the steps taken in order to fabricate our set-up. Subsequently, we consider our plans for further enhancing the current configuration so as to eliminate the obstacles encountered.

4.1. OpenSCAD mold generation

The construction of the 3D molds with the patterns considered in Chapter 2 was again performed using the OpenSCAD software [88]. The whole process for generating the 3D molds .STL files is analogous to the one from Sec. 3.1. This time, however, we generated a stamp-like mold instead of a hollow slab, see Fig. 3.3. We have adopted this approach to generate the molds as it would be unfeasible to 3D print the venation patterns without a scaffold to maintain the structural integrity of the networks. Moreover, many other defects could arise in the subsequent phases of model fabrication, a possibility that only further justifies our choice. All molds are square and have the same size as the meshes generated in the CFD process, 50 mm.

In contrast to the 3D models generated for the CFD procedure, this time, we have chosen to improve the quality of the channels, even though rendering time was longer and a greater number of defects appear in the surface of the .STL causing the geometry to be non-manifold. In particular, what we mean by improving the quality of the channels is an increase in the number of faces of the frustrum solids that form the venation, further smoothing their surfaces. This could be done because we have only attempted generate an experimental setup for one geometry, as the validation process is still in its early stages. Additionally, we plan to use tools to generate molds with higher resolution in future work. Hence, molds generated that way could benefit from the increase in quality, see Sec. 4.4. Finally, filtering of the .STL files for 3D printing purposes can be much less rigorous, as the software for 3D printing accepts non-manifold geometries and no observable gain by using completely fixed geometries was detected.

Next, we showcase the molds produced from the .STL files and briefly decribe their fabrication using the 3D printer.

4.2. 3D printed molds

After the .STL files of the molds were generated, we proceeded with the construction the molds with the 3D printer readily available at $LabM^2$, see Fig. 4.1. As we are attempting to produce our first prototypes, this low resolution 3D printer was used. As discussed in Sec. 4.4, however, we expect to employ either 3D printers of higher resolution or use different techniques to fabricate the molds altogether.

In the first step, we use the files as input to the 3D printing software Repetier-Host [98]. Next, we slice the geometry with the CuraEngine. This step is fundamental, as it transforms the .STL surfaces into individual layers and creates the code the 3D printer uses to fabricate the object [98]. Finally, we raise the temperature of the extruder, which melts the 3D printing material and ejects the polylactic acid (PLA) filaments to produce the molds. Only then, we start the printing phase.

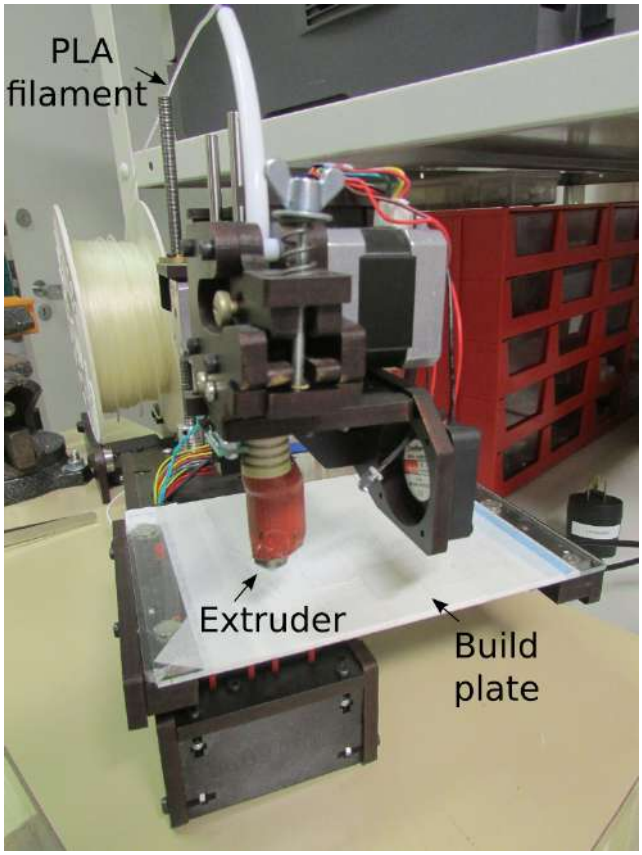

Figure 4.1.: 3D printer present at $LabM^2$. The picture showcases some of the 3D printer components.

The only challenge we encountered while 3D printing the molds was that due to the high temperature of the extruder, the molds tend to deform and become irregular when low filament densities are used. Moreover, large mold thickness also seemed to cause the same problem. Fortunately, we were able to bypass these issues by simply setting the filament density of the molds to 100% and reducing the mold thickness as much as we could. This way the molds visibly did not display the same irregularities observed earlier. One of 3D printed molds contructed in this fashion can be seen in Fig. 4.2.

Figure 4.2.: 3D printed mold with venation-inpired geometry.

4.3. Experimental set-up

In this section, we offer an overview on how we assembled the current configuration of the experimental models, see Fig. 4.3.

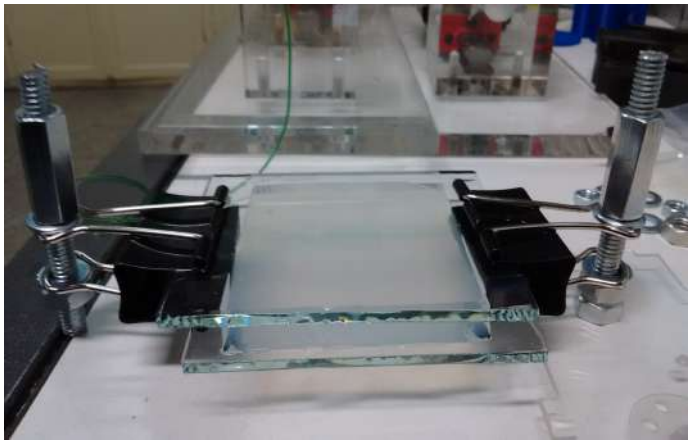

Figure 4.3.: Current experimental model set-up.

In the first stage, we employed the 3D printed molds to form the generated patterns on the agar slabs, see Fig. 4.4. We have utilized agar instead of purified agarose, due to cheaper cost of the former. Moreover, their properties are somewhat similar, as agar is composed of agarose and a smaller portion of agaropecting [99]. Hence, we felt justified in opting for agar to produce the prototypes. Later on, when we fabricate a successful set-up, we plan on switching to purified agarose, as its properties are characterized due to its extensive use in gel electrophoresis [100]. That will be a necessity when aiming to generate quantitative results.

Figure 4.4.: Agar slab generated using a 3D printed mold.

The 3D printed molds were fixed at the bottom of a petri dish coated with an aluminum sheet. We then heated an agar solution in a microwave for about 1-2 minutes eventually halting the heating to stir up the solution to minimize bubble formation. Next, we waited for the agar solution to cool as much as possible without becoming jelly-like and only then we poured it on the coated petri dish. We proceeded removing the remaining bubbles from the region immediately above the mold, before the solution solidified. Finally, we carefully extracted the now solidified gel slab from the petri dish without damaging it and then discarded the undesired regions of the gel retrieving the pattern seen in Fig. 4.4.

In the next stage, we carefully placed the gel slab on top of a glass plate, with the pattern facing upward, and at each vertex we positioned a 3D printed support. Next, we poured water on the slab surface to fill the channels and prevent bubble formation. We then placed a second glass plate over the slab cautiously so as to avoid the appearance of bubbles. The final step was to attatch two binder clips at the two opposing sides of the apparatus, see Fig. 4.3. The clips exert great pressure on the slab and were it not for the 3D printed support pieces, the high pressure could damage or completely crush the gel. Lastly, we used screw threads and nuts to make the pressure exerted by the clips ajustable, see Fig. 4.3.

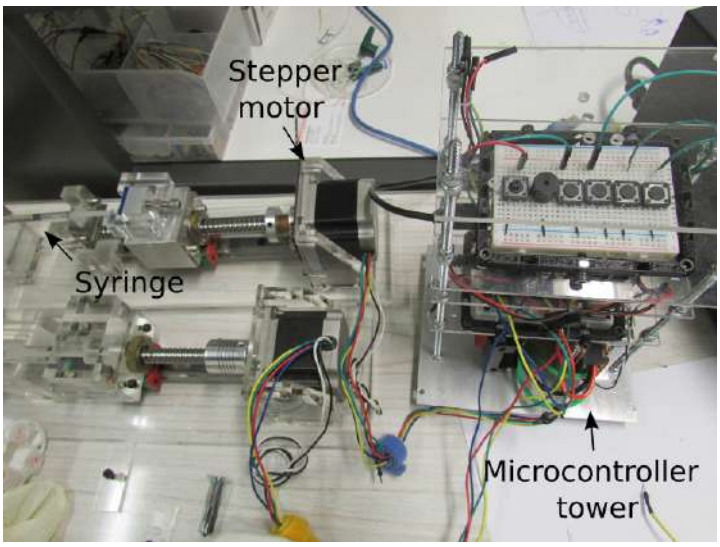

Figure 4.5.: Microfluidic pump developed by a fellow group member Juan Enrique Rivero Cervantes at $LabM^2$.

In our first tests, we employed a microfluid pump injecting a dye solution at a constant volumetric rate of 10 μl/min. The microfluidic pump was developed by a fellow group member Juan Enrique Rivero Cervantes at $LabM^2$, see Fig. 4.5. The pump uses a stepper motor attatched to a screw thread as a mechanism to precisely control the force applied onto a syringe. The motor, in turn, is controlled by a microcontroller allowing us to adjust the injected rate, which can be considered constant, even though, in practice, the motor does not rotate continously. A rubber tube connects the syringe to the inlet of the system, see Fig. 4.3. Naturally, in this configuration, the pressure at the outlet equals the atmospheric pressure.

4.4. Discussion and challenges

Overall, we attempted to mimic the traits of interest from the μ-FGPVs original set-up as much as possible. We have even employed the same parameters whenever we could, e.g., the constant volumetric rate of 10 μl/min [24]. That is because the variant of DSSCs was the main target of the CFD simulations from Chapter 3. Nevertheless, a valid comparison between our experimets and the ones from the article is not possible as we are not aware of all the parameters of the original device.

Another obstacle we face is that the current configuration of the CFD meshes cannot be validated by the set-up we fabricated. That is because the channels are inside the slab in the CFD simulations, see Fig. 3.3, and not at one of its faces, see Fig. 4.4. As it would be a formidable challange to construct an experimental model with an interdigitated pattern inside the slab precisely adjusted to be parallel to both faces,

the more reasonable approach is to construct meshes with the pattern at one of the faces to solve the flow. We have assessed this approach and concluded that it is feasible, although challanges are still present. Thus, in future work, we plan on adopting this methodology.

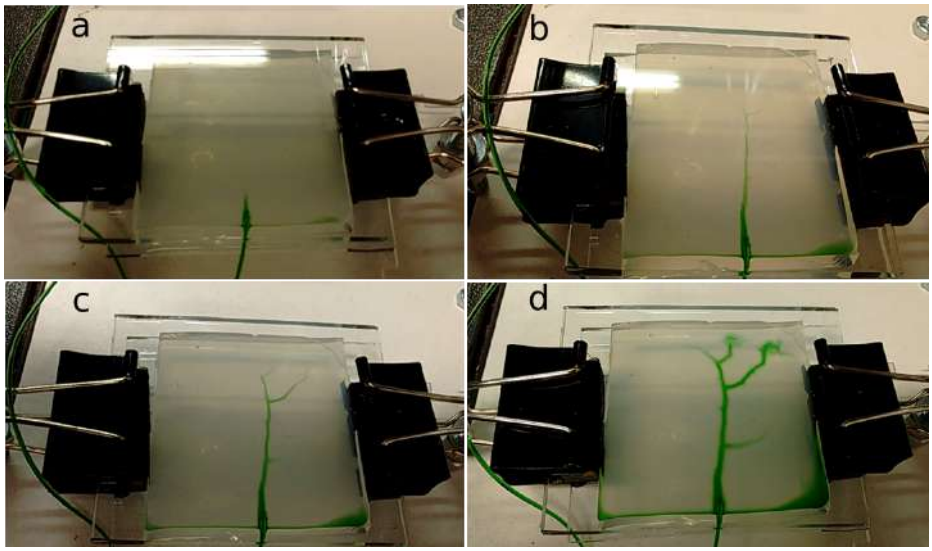

Figure 4.6.: Experiment performed using the current set-up. (a) picture 10 s after injection was initiated. (b) picture after 1 min. (c) picture after 2 min. (d) picture after 7 min.

As for the fabricated models, we were able perform some experiments with them, see the results in Fig. 4.6. The concept of the current model works. The dye profile at the end, however, did not meet the more uniform distribution we expected among the veins. The reason for this that the slab does not have a constant thickness. Although the bottom face of the slab can be considered plane, the upper face, the one which was in contact with the 3D printed mold, was very irregular. Hence, regions with lower thickness formed regions of low pressure, which attracted most of the solution, while thicker regions repelled the dye solution. The result is the uneven dye distribution we observed. As for the upper face, it became irregular due to the mold deforming when in contact with the heated agar solution. The mold deformed because it is made of PLA, as previously mentioned in Sec. 4.2. This is the current challenge that kept us from performing accurate experiments. The solution around this problem may be to either coat the mold with a thermal insulation material, e.g., silicone. In case the coating process works, we will employ 3D printers of higher resolution to produce the molds. In case it does not, we can use a completely different material altogether to fabricate the molds. The material could be a metal, so that it does not deform when subjected to the temperatures of the agar solution. In particular, our collaboration with the Laboratório de Micromanufatura (LMI) at the Instituto de Pesquisas Tecnológicas (IPT) may aid us in fabricating molds with different materials. In fact, we have already assessed the feasibility of the process

and obtained very sactisfatory results, see Fig. 4.7. Moreover, the mold fabricated there, has a much higher resultion than the 3D printed molds we constructed.

Figure 4.7.: Mold fabricated at Instituto de Pesquisas Tecnológicas (IPT) using machining technique. The mold has a different material and higher resolution.

After we produce a proper model, switch from an agar to an agarose gel and standardize the fabrication procedure, the next step will be to develop a tool that allows us to perform quantitative measurements of dye concentration in the different regions of the hydrogel. The simplest approach seems to be imaging techniques. It has been shown that it is possible to determine the concentration at each pixel of an image after calibration with solutions of known concentration, making this a viable option [101, 102]. Another tool that can further aid us in obtaining data from the experiments are optical techniques. Laser speckle flowmetry experiments that may be performed at $LabM^2$ could help us in determining the velocity of the flow at different regions of the channels [103]. This data can be compared with the results from CFD and help us validate the models.

As a last remark, we are also striving to implement a technique to maintain constant pressure at both the inlet and outlet of the system. In Chapter 3, the constant pressure boundary condition at the inlet yielded interesting results which must be validated using experimental models.

5. Final considerations

In this chapter, we outline the advances of our research. In addition, we list a number of possible directions future reasearch may take based on the current work. We present options which are essentially the natural next steps of this research as well as exciting longer-term ideas that could possibly guide our future efforts.

5.1. Achievements

In this section, we summarize the most important achievements of this work. In the upcoming sections, we will present some possibilities for upcoming research.

In the first stages of this work, we successfully implemented Fortune's algorithm in C++ for optimal Voronoi diagram construction in 2D, see Appendix A. Next, we implemented the slab decomposition algorithm to perform Nearest neighbor searches in optimal time using the previously constructed diagrams. We then implemented both algorithms proposed by Runions et. al. [44] for open and closed venation generation. Finally, we added another step to the open venation program, in which a second, complementary venation was generated. As discussed in Chapter 2, that was done because two networks, one connected to the inlet and another to the outlet, are necessary in the target applications we were inspired by. The results can be seen in Fig. 2.13. Algorithms that computed the fractal dimension, VLAs and bifurcation numbers for the geometries were also developed and incorporated into the code. The user can retrieve the data at the end of the geometry construction process, see Sec. 2.5. Diameters for each of the veins were assigned based on Murray's law [73] and the input diameters for the inlet and oulet channels, see Sec. 2.6.

In the next phase, we successfuly generated 3D models for 18 geometries using the OpenSCAD program [88]. The construction and repairing of the models was thoroughly described in Sec. 3.1. These models served as the foundation for the subsequent fabrication of our current experimental set-up. They were also important in the meshing stage of the CFD process.

Next, we focused on employing CFD methods to solve the flow problem in the slab-shaped porous systems, see Fig. 3.3. We successfuly employed the OpenFOAM CFD toolkit to obtain the numerical solutions for each of the geometries in our database. Mesh generation was executed using the *snappyHexMesh* utility and computation of the solutions was done using both the *pimpleFoam* and the *scalarTransportFoam*

solvers. Solutions for two different types of boundary conditions were added to the database along with the corresponding properties of the geometries. A brief statistical analysis of the results was done employing the R programming language and while it may still be early to offer definitive remarks, we observed that centered geometries have seemed to perform better than diagonal ones when the constant pressure drop BC was used, see Sec. 3.4 and Sec. 3.5. In addition, models where the primary centered venations were connected to the inlet distributed the solution more uniformly based on our current results.

Finally, we have created the first experimental models which will be used for empirical validation in future work. Although, we were still not able to employ these models in the validation of the CFD results, we feel that we have proved that the concept of this design works. Moreover, the main obstacles we came across may be overcome using other mold fabrication techniques which are available to us, as outlined in Sec. 4.4.

5.2. Follow-up studies

Among the most urgent steps for subsequent work is the improvement of the experimental models and development of the equipment necessary for the empirical validation of the CFD solutions. An in-depth discussion on this topic can be found in Sec. 4.4. Here, we add that measurements using the imaging techniques could benefit greatly from use of microcontrollers, as they not only can assist in controlling important parameters, e.g., temperature [102], injection rate, etc., but also can be employed to automate parts of the experiments, potentially reducing measurement errors. In view of this, the efforts to employ of microcontrollers seem justified.

Along the same lines, validation of the CFD solutions through mesh refinement can be started right away. That is specially true as very recently our group has purchased a workstation capable of handling finer meshes. As a result, hardware limitations are no longer an issue.

In parallel with the validation of the current geometries, the inclusion of new entries in the database is also an option. We expect to obtain more robust results with the additional entries. Moreover, we plan to add entries for different models to the database, e.g., the closed architectures or the interdigitated patterns with three venations, which were presented in Sec. 2.6. We hope that an analysis including the new models will enable us to gain further insight into what makes a geometry efficient. Another aspect we have not covered in our study was how the average distance between the primary and complementary venations affects performace of the geometry, as discussed in Sec. 3.5. Future studies considering this aspect are also in our plans.

In the subsequent sections, we present two ideas that, while incorporating many aspects of the current work, also include new challanges. Hence, either one or both

could serve as our next goals.

5.3. Models for PEMFCs

The first possible target could be to create models of PEMFCs with the current geometries. In this process, in the first stages, we could employ CFD to compute the field solutions in the flow field plates, for instance [18]. Achieving this is not as straighforward as employing CFD techniques to solve the flow inside the μ-FGPV devices, as there are two-phases in the of the flow field plates, liquid and water-vapor. Therefore, this could be an excellent opportunity to incorporate more advanced aspects of CFD in our research.

In later stages, we could eventually fabricate actual PEMFCs prototypes, not only to validade the CFD results, but to measure a possible improvement in performace of these cells.

5.4. Application to 3D cell cultures

The second target we have thought of are 3D cell cultures. 3D cell cultures have been gaining ground recently, as they represent more realistic models for real living tissues than conventional 2D cultures. In part, that is because many of the complex cell-to-cell and cells-to-matrix interactions only arise in a 3D environment [104,105]. In addition, studies have shown other inadequacies of 2D cultures, offering evidence that gene expression of cell grown in 2D and 3D are different [105]. Hence, 3D cell cultures is a still fresh and compelling area of reasearch.

The particular topic in this vast research field that mostly relates to our work is the problem of vascularization in 3D cell cultures, specifically in dense cultures [106,107]. In order for a culture to be an accurate representation of a living tissue, to a certain extent, the availability of nutrients and of gas exchange must be similar to that of a living tissue. That generaly is only possible with the aid of channel networks to perform these needed funtions. As there are many similarities between this problem and our research, we consider it would be an excellent opportunity, for instance, to move from the realm of 2D to 3D. We could conceivably develop efficient geometries that could serve as the vascular system of these cultures, while acquiring expertise in this stimulating field of research in the process.

A. Appendix: Voronoi diagrams and the Nearest Neighbor search

A.1. Overview

In brief, a Voronoi diagram is a form of space partitioning, in which an Euclidian plane, or any other metric space[1], containing a set S of sites s, a.k.a. Voronoi sites, is divided so that points which lie closer to a site s than to any other site of S define that site's corresponding cell, a Voronoi face in $2D$. Points whose two closest sites are equidistant define an edge, the boundary between two faces. Finally, points which possess three or more equidistant closest sites are called vertices or nodes[2] [81, 82]. All of those definitions are summarized in Fig. A.1.

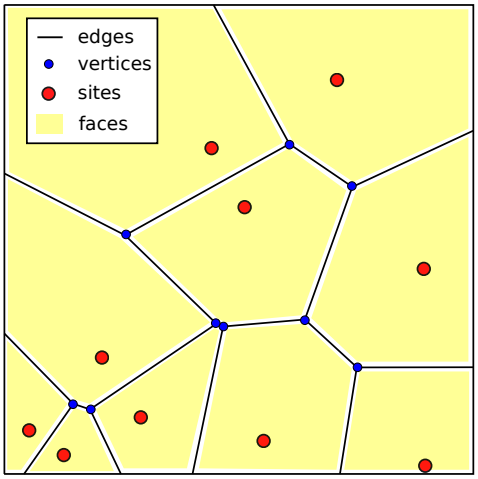

Figure A.1.: Chart of a Voronoi diagram illustrating the definitions of edges, sites, vertices and faces.

Voronoi diagrams find a wide range of uses throughout many different fields in science from astronomy and physics to social geography and biology [81, 109, 110]. Its importance stems from the simple property of partitioning space so as to have all of

[1] Notice that neither the metric space needs to be an Euclidian space nor the metric used necessarily needs to be the Euclidian distance [108]. In fact, the Voronoi diagram can be defined in a general manner although a more rigorous definition is out of the scope of this text.

[2] More properties of Voronoi diagrams are discussed in Sec. A.4.

the points inside a partition lying closer to the corresponding partition site. That property makes encountering the nearest neighbor site of a new query point, an ubiquitous problem in science and one we face in this work, remarkably straightforward. Aside from this, another reason Voronoi diagrams are utilized to solve the nearest neighbor search problem is due to the possibility of generating these diagrams computationally in $O(n \ln n)$ time [79], where n is the number of Voronoi sites. Once the Voronoi diagram related data structures are generated, a point location query can be done in $O(\ln n)$ time [78,111]. All things considered, that elevates the Vornoi diagram approach to one of the optimal solutions to the nearest neighbor search (NN search) problem in $2D$. Naturally, it is much faster than the naive linear search $O(n)$ time complexity.

A.2. Constructing Voronoi diagrams: Fortune's sweepline algorithm

Fortune's sweepline algorithm, proposed by Steven Fortune [79, 81], is the optimal method for generating Voronoi diagrams computationally in $2D$ for any set S of Voronoi sites. It possesses a time complexity of $O(n \ln n)$. Essentially, the algorithm sweeps the plane once in one direction, stopping at specific sites called events [81], where calculations and some other considerations need to be made. Once all events are handled, Voronoi diagram construction ends.

It is interesting to note that, computationally, the sweeping is not continuous, but discretized. In other words, the algorithm jumps from one event to the next without wasting any processing at sites inbetween them, since that would not have any impact on the final output. The position in the plane where the processing occurs at a given instant is labeled sweepline and it moves only in the direction of the sweeping. The events are not evenly spaced and since one does not know all of them beforehand, some must be identified on the fly. The only restriction placed on the location of the newly idenfied events is that they must be ahead of the sweepline [81], otherwise the sweeping direction would turn back and forth many times during one run, something unnecessary which could, in fact, yield an incorrect output.

A.2.1. Site and circle events, breakpoints and beachline

As far as the Fortune's algorithm is concerned, there are only two types of events: site events and circle events [79,81]. They are labeled differently due to their distinct nature. The site event type deals with the complications that stem whenever the sweepline reaches a Voronoi site. Since all Voronoi sites are known initially, the site events don't have to be determined during the run and therefore are simply stored and dealt with in order whenever the sweepline arrives at them [81]. The other class of events, the circle events, are more subtle. Each one of those events are

related to the possible occurence of a Voronoi vertex during the algorithm execution. The reason they receive that name is due to the vertex being equidistant from the Voronoi sites that generated it, i.e., if one traces a circumference around the vertex, with the radius being the distance from the vertex to one of the Voronoi sites, it follows the other sites will also reside at the circumference. Possible circle events are spotted on the fly whenever three consecutive Voronoi sites are encountered by the sweepline. Circle events happen when the line reaches the outermost point of the circumference around its associated Voronoi vertex. We emphasize these events may not happen, being discarded whenever the sweepline encounters a Voronoi site inside a circumference associated with a would-be Voronoi vertex [81].

In addition, two other concepts are further introduced due to their crucial role in the algorithm: the beachline and the breakpoints. In order to comprehend what those are, one must recall that a point and a line can be used to define a parabola and when they do, they're named focus and directrix respectively. With that given, we may proceed establishing an analogy between the Voronoi sites and focuses and between the sweepline and the directrix, as sketched in figure Fig. A.2. Sites not yet reached by the sweepline do not define parabolas. Therefore, new parabolas appear only when site events occur. In fact, those new parabolas are simply straight lines perpendicular to the sweepline when the correponding Voronoi sites are being intersected, see Fig. A.2. As the sweepline moves, those straight lines become sharp parabolas and as the sweepline moves further the sharp parabolas become broader and broader. A number n of parabolas can be defined after the sweepline crosses a number n of sites, however the beachline comprises only the parabola segments which are closest to the sweepline along the Voronoi diagram width [81]. The reason behind the name beachline is quite evident after one analyses its countour, see Fig. A.2. It does ressemble a real beachline or shoreline. Points that separate two consecutive parabola segments in the beachline are denominated breakpoints [81]. If two consecutive breakpoints can be associated with only two parabolas, i. e., only two Voronoi sites, then a Voronoi edge is defined by tracing a line between those points. In contrast, two consecutive breakpoints which are associated with three different parabolas will not have an edge defined inbetween them. By definition, whenever circle events occur, a Voronoi vertex is placed where two (or more) consecutive breakpoints meet. Furthermore, two or more Voronoi edges meet at the vertex and one or more parabola segment cease to exist. Everytime a circle event occurs a new Voronoi edge segment leaving the vertex is created and associated with the remaining breakpoint. In Fig. A.2, all points discussed in this section are illustrated.

A.2.2. Algorithm data structures

In this section, key data structures used by the program are discussed. Some knowledge of computational geometry algorithms, which can be readily found in any computational geometry book [77, 81, 82], is expected.

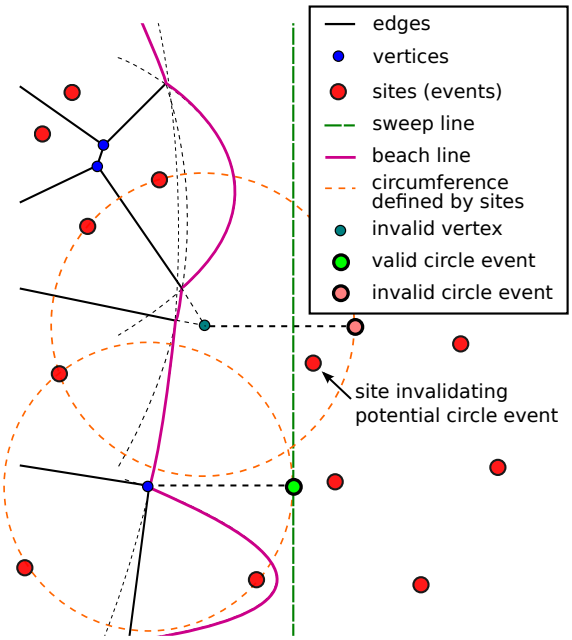

Figure A.2.: Chart summarizing some of the main aspects of the Sweepline algorithm. The sweepline (in green) is handling a circle event in the picture. Observe that the next event to be handled in the diagram is a site event, which will invalidate a future circle event. It follows from the properties of Voronoi diagrams that no Voronoi site (in red) can exist inside a vertex defining circumference (in orange).

When implementing the algorithm, both event types instances are stored in a priority queue data structure [77, 81]. A priority queue allows for data storage in a vector-like structure. In general, its implementation uses dynamic memory allocation, and thus, usage does not demand initial knowledge of maximum vector size [77]. In addition, the priority queue data structure sorts queue elements according to a comparison funtion, the priority criterion. The top element of the queue, the next instance to be removed by definition, can, for instance, always be the one with lesser value in the x-direction. Hence the next element must have a greater value in the x-direction since the priority queue data structure always sorts the elements everytime a new element is included in the queue [77]. This is how the one-way sweeping is implemented [81]. The basic structure of the algorithm, therefore, is to simply process all events, site or circle types, in the order they appear on top of the priority queue. After all sites are processed, the run is completed and the diagram built. In the C++ programming language, one can use the default implementation of the priority queue data strutcture by including the queue library in the header.

One of the great achievements of the algorithm is its ability to recognize that everything prone to changing lies on beachline. The section of the plane which still hasn't been sweeped has no effect on the diagram construction until it is reached.

Meanwhile, the section which was already sweeped and no longer belongs to the beachline, is already part of the final output and, thus, does not affect the diagram construction, see Fig. A.2. Therefore, the real bottleneck is the beachline. The algorithm cleverly takes advantadge of this fact and implements the beachline as a binary search tree (BST) [81], an optimal data structure which allows for the deletion and insertion of new elements in $O(\ln k)$ time [77], whereas other more naive implementations in general waste $O(k)$ time in the processing. Here, k is the average number of sites and breakpoints in the beachline, which in general is a small fraction of the total number of Voronoi sites. Thus, the BST implementation is the key factor which reduces the processing time from $O(n^2)$ to $O(n \ln n)$, where n is the total number of Voronoi sites in the diagram. This is the actual computational reason Voronoi diagrams can be used solve NN search problem. BSTs are readily implemented in C++ even though they are not part of the standard library. It is convenient to employ dynamic memory allocation when implementing them, by using the *new* and *delete*[3] operators, since frequently one does not know what the size of the tree will be [77].

A.2.2.1. Binary search trees

A brief introductory example of BSTs will be given, so as to guide the understanding of the actual implementation of the beachline as a BST data structure.

Suppose there is a set of n integers stored in the computer memory and one wants to find a specific integer among them. Provided the integers are stored in an array or vector, the more obvious and naive approach would be to simply sweep the array for the sought element, checking each and every instance of the array until the desired one is found, see Fig. A.3. Considering the time to check one array element is constant, on avarage, the time it takes to find the wanted integer grows linearly with array size. Since, in general, one is interested in the behavior for very large n, unless the behavior of the functions is similar, e.g., linear, logarithmic, etc., proportionality constants are unimportant when judging efficiency of different algorithms. Hence, in the big O notation, the time complexity of this naive search approach is simply $O(n)$, which means performance is acceptable if the array is not very long. However, as the array grows, the approach becomes increasingly unfeasible.

In order to enhance search speed, one may employ a different storage method. Instead of simply storing the integers in an array or vector, one may store them in a binary search tree. BSTs are data structures which store data in its nodes according to a simple comparison rule [77, 113]. When constructing a BST, given the previous example, one starts by picking an integer and storing it in the first node, also known

[3] It is of utmost importance to utilize the *delete* operator before deleting all pointers that refer to a memory section in the heap, since, otherwise, that would characterize a memory leak and bad programming. Too many memory leaks can result in memory crashes if the operational system runs out of RAM. We employed the Valgrind utility to discover and eliminate memory leaks [112].

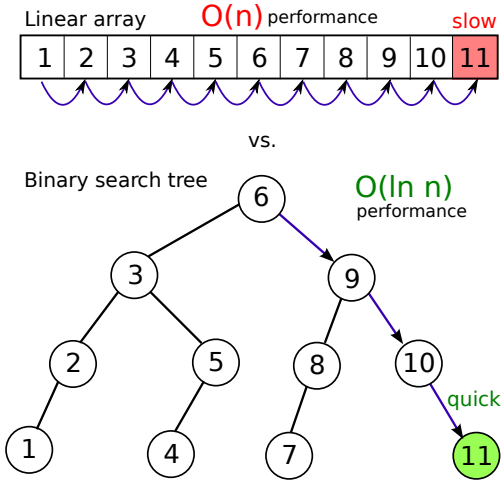

Figure A.3.: Comparison between the naive linear search and binary search. In the case illustrated, aside from number 1 and 2, binary search has a performance at least similar to the linear one.

as the root node. All searches, insertions and deletions start at the root. In the BST arrangement, every node of the tree, including the root, has at most one parent and at most two children nodes, see Fig. A.3. Hence the name binary. In addition, one or both children nodes may be void. If a node has no children, it is called a leaf node, and in case it has at least one child, it is referred to as an internal node. Children nodes are usually labeled as right or left node and the criterion which defines whether a new data is to be stored in the left or right node will depend on the comparison rule and on data value. In this example, starting at the root, the integer value of tree nodes are examined and compared to the new value as one moves down the tree until a void node is reached. In case the new value is greater one moves down to the right child node, and in case it is lesser, one moves down to the left child one. That is the comparison rule. Since the root, in this example, is also a leaf node because both children are void, a new child node is created, either the left or right one, and the new integer is stored in it. The procedure is followed until every integer is stored. Once the storage process is completed, one is able to search for an integer, the query, by using the same comparison rule used to construct the BST. Whether the query is in the tree or not is irrelevant. The key feature of BSTs is that query searches, insertions or deletions can be done, roughly, in $O(\ln n)$[4] time [77,113], i.e., for large n it completely outperforms the naive method.

As a final remark, in graph theory, BSTs are analogous to arborescences or out-trees, since they are directed graphs, with all of its edges pointing away from a

[4]A remark needs to be made: if the order which numbers are inserted into the tree when building it is roughly crescent or decrescent, then the BST will behave approximately as an array, and BST performance will dramatically decrease. In contrast, if the order which the data is inserted is random, tree performace will be acceptable on average.

single node, the root node [76]. It follows that it is possible to reach any node from the root. However it is not possible to reach the root from another node. That is in contrast with data structures that originate from undirected graphs, for instance, a doubly-connected edge list, discussed in Sec. A.3 [77].

A.2.2.2. Beachline BST

The data structure utilized to represent the beachline is a BST. Therefore, it possesses the same basic properties which characterize all BSTs. Nevertheless, it bears some peculiarities which will be discussed in this subsection.

In the preceeding case with the integer BST, for instance, the comparison rule was to proceed to the right child node whenever the query value was greater than the value of the current node or to continue to the left child, in case it was lower. However, in the Voronoi diagrams present in this work, we are not handling integers in $1D$, but points in the $2D$ Euclidian plane, meaning that there are two real numbers assigned to each data element, which, consequently, allows for a greater variety of comparison choices. The chosen comparison criterion, for the beachline BST case, is to decide between children nodes based only on the coordinate normal to the sweepline moving direction, e.g., the y coordinate in Fig. A.4 [81]. Furthermore, leaf nodes and other nodes represent different features of the beachline: leaf nodes correspond to parabola segments, the arcs, while internal nodes play the role of the breakpoints, see Fig. A.4 [81].

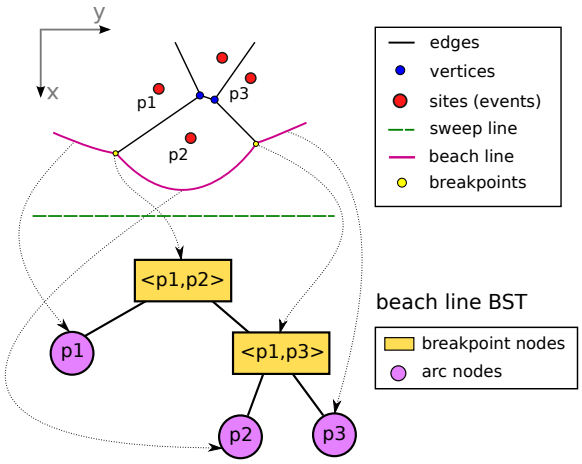

Figure A.4.: Chart representing a beachline section and the corresponding beachline binary search tree.

Implementation-wise and in contrast with the integer BST, where the number is simply stored in the node along with pointers to both children, storing breakpoint coordinates in the beachline BST internal nodes would be pointless, since they change at every step during the run. Hence, to change all nodes of the BST at

Listing A.1: Main data structures used by Fortune's algorithm.

```
1
2  typedef pair<double,double> point;
3  #define x first
4  #define y second
5
6  priority_queue<point, vector<point>, siteAscendingXComparison> siteevents;//site
       events priority queue
7  priority_queue<circleevent*, vector<circleevent*>, circleAscendingXComparison>
       circleevents;//circle events priority queue
8
9  struct beachlinenode{
10         beachlinenode *left,*right,*parent;//leaf, right and parent node pointers
11         point p1,p2;//focuses of both intersecting breakpoint-generating parabolas.
               Only one point is valid in case this is a leaf node
12         bool breakpoint1or2;//allows program to tell which breakpoint is being
               handled
13         circleevent *potential;//pointer to potential circle event: only possibly
               non-null for leafnodes
14         beachlinenode(point pp1, point pp2, bool b1orb2, beachline *prt)
15         : p1(pp1), p2(pp2), left(0), right(0), parent(prt), breakpoint1or2(b1orb2),
               potential(0) {} //beachline node constructor
16 };
17
18 struct circleevent{
19         double circlemaxx;//x coordinate associated with the circle event: circle
               center x coordinate plus circle radius
20         point circlecenter;//Voronoi vertex coordinates associated with the circle
               event
21         beachlinenode *arc;//employed in beachline BST manipulation, e.g., arc
               deletion or new potential circle events after deletion
22         bool valid;//used to invalidate the circle event when deemed necessary
23         circleevent(double x, point p, beachlinenode *leafarc)
24         : circlemaxx(x), circlecenter(p), arc(leafarc), valid(true) {}//circle
               event constructor
25 };
```

every step would be so time consuming it would render the beachline BST itself useless. Thus, the focuses, the sites, of both intersecting parabolas are stored in every internal node, making it possible to compute the two breakpoints positions, points where the parabolas intersect at a given step [81]. An additional parameter is stored in breakpoint nodes, so the program is able to distinguish between the two possible breakpoints.

An outline with key features of the beachline BST data structure and event priority queues implemented in C++ are shown in Listing A.1.

A.2.3. Fortune's sweepline algorithm implementation

In this section, detailed flowcharts representing the implemented algorithm are presented, along with their respective explanations. In Fig. A.5, the backbone of the algorithm is sketched. The discretized motion of the sweepline along a direction in the plane, which from now on will be taken as parallel to the x direction, is represented by the loop showcased in the flowchart. Each interation of the loop amounts

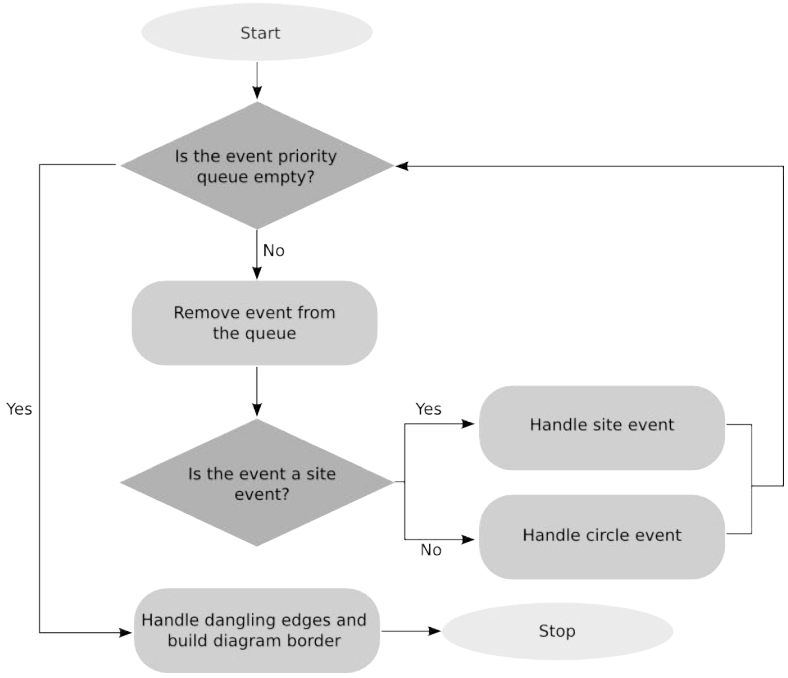

Figure A.5.: Event queue handling flowchart [81].

to a step taken by the sweepline along the x direction, since events in the queue are sorted on increasing x-coordinate [81].

In the following, flowcharts are provided detailing the set instructions executed by the algorithm whenever it handles site or circle events. In other words, when the algorithm reaches the handle event instruction in the flowchart of Fig. A.5, it is in fact executing the instructions in either the flowchart of Fig. A.6, if the event is a site event, or of Fig. A.7, in case the event is a circle event [81].

It is worthwhile noting that in order to check the existance of a potential circle event, brought about by arc deletions or site events, one does not have to check all arcs, leaf nodes of the beachline BST, but only the triples, three consecutive arcs which possess arcs that were submitted to change in that given step of the main loop.

After all events of the priority queue have been dealt with, only a beachline BST with no associated potential circle events will remain. Nevertheless, the remaining beachline will contain breakpoints with affiliated edges. Each of those affiliated edges could be extended to infinity, since there are no more circle events defining its second extreme left. Therefore, in order to complete the crafting of the Voronoi diagram, the algorithm must deal with the remaining dangling edges. That is accomplished by defining a bounding box encompassig the diagram. The bounding box is allowed to have any shape, as long as it terminates all dangling edges at the points where they intersect with the bounding box curve.

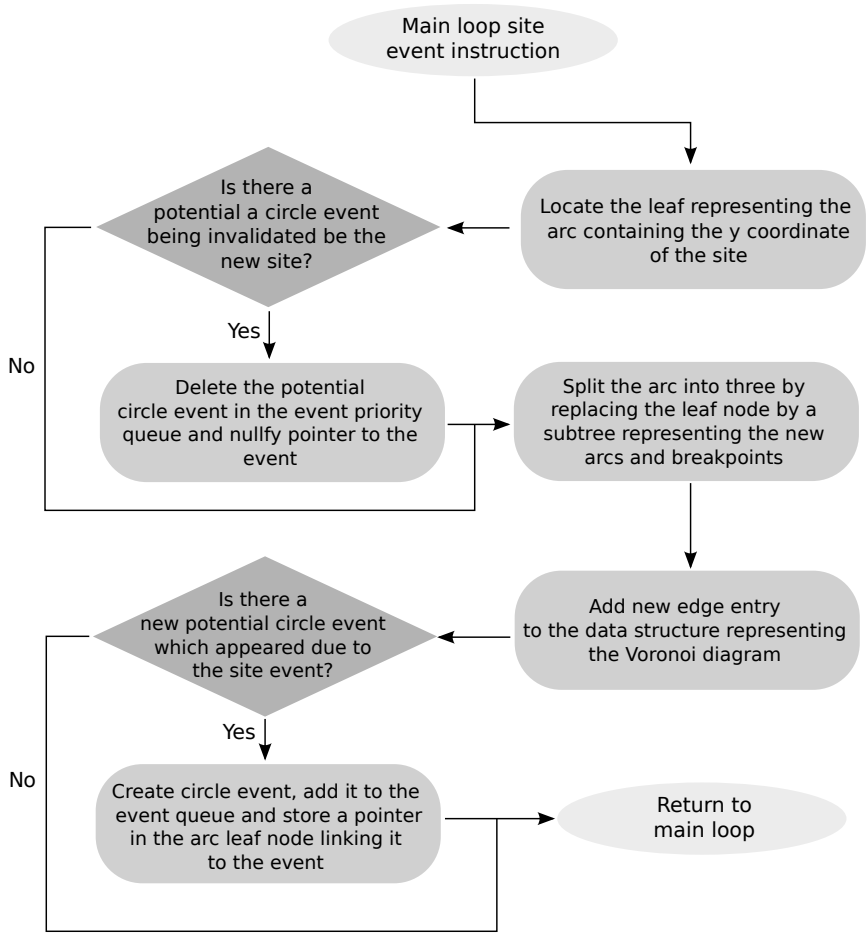

Figure A.6.: Flowchart representing how the algorithm handles site events [81].

A.3. Doubly-connected edge list (DCEL)

In previous subsections of this appendix chapter, concepts underlying Voronoi diagram construction with the Sweepline algorithm were presented along with a detailed description of the algorithm itself. Notably, a description of the data structure employed in the Voronoi diagram representation was not provided. This subsection aims to scrutinize the features of this data structure, the Doubly-connected edge list (DCEL) [77, 81].

Before the DCEL data structure is thoroughly discussed, an elementary, but indispensable, linked list introduction is furnished. Linked lists are array-like data structures consisting of nodes, which are connected to the next node by a pointer [77]. They are not necessarily sorted and can store any data type inside them. Their single most important feature is that they form a chain of nodes, which are always connected to the next via a pointer. The entrance pointer to the linked list is called head, and it always points to the first node in the list. The list, evindently, ends at

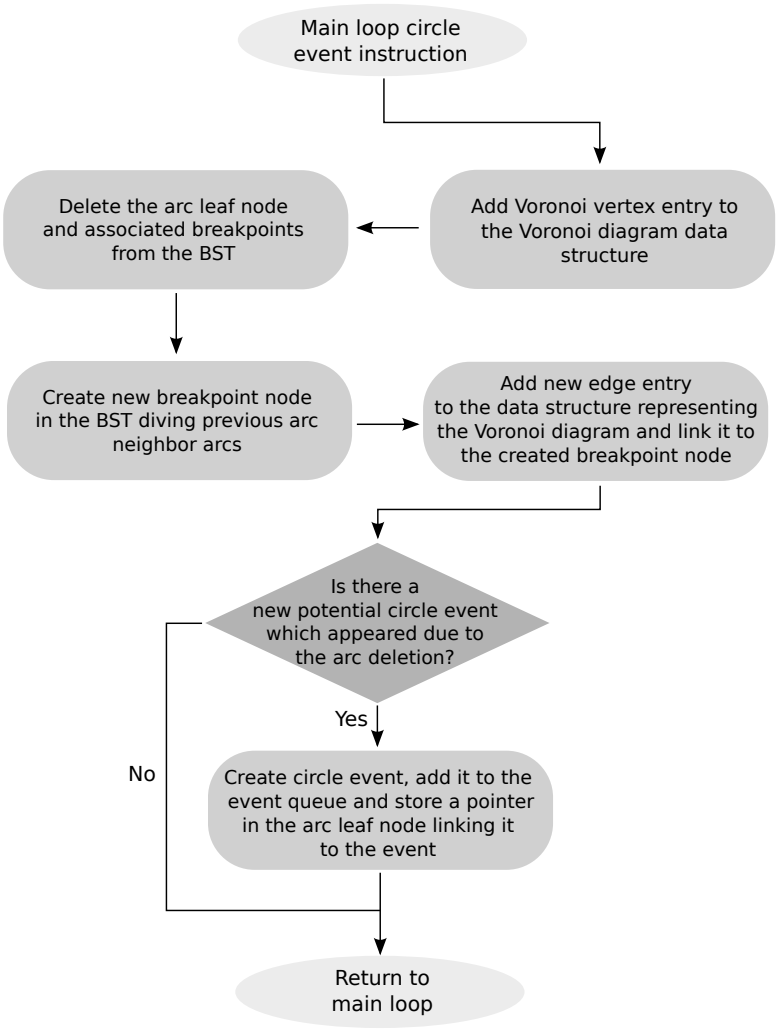

Figure A.7.: Flowchart representing how the algorithm handles circle events [81].

the node which contains the null pointer (if the head is null, then the list is empty). In a similar linked list arrangement, the nodes are not only linked the next ones, but to the previous nodes as well by a second pointer. These linked lists are known as doubly linked lists, as opposed to the first type, that contain only the next pointers and which are known as singly linked lists [77].

As the reader might have guessed, DCEL data structures hold a ressemblance to doubly linked lists, even though they possess many other unrelated key features. DCEL data structures, in general, can efficiently describe and facilitate manipulation of planar graphs, in particular, planar straight-line graphs (PSLGs) [82]. PSLGs are comprised of nodes embedded in a plane and of edges, straight line segments, which might connect pairs of nodes, see Fig. A.8. In contrast to nonplanar graphs, the edges of PSLGs do not intersect each other except at the end points, namely, the

nodes or vertices [114]. That property is critical, since it assures that all kinks in the boundary defining each face are nodes of the graph. Naturally, the faces of the graph are merely space partitions bounded by the edges.

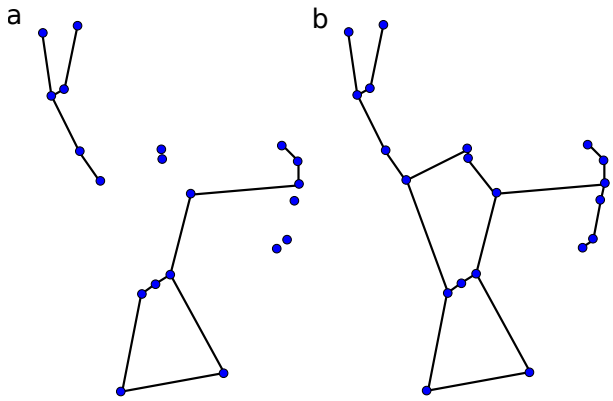

Figure A.8.: Planar straigh-line graphs. Nodes are depicted in blue and edges, in black. Edges intersect only at the nodes. Notice how nodes may or may not be linked to other nodes via edges. The graph is labeled disconnected (a) in case there is at least one pair of nodes {x, y} without a path leading from x to y, and connected (b), otherwise. The exception is a single node in the plane, which is considered a connected graph. Finally, in the second graphs, for instance, one can count 3 faces, 2 enclosed by graph edges and the outer face, which encloses the graph itself.

As a side note, it follows that Voronoi diagrams in the $2D$ Euclidian space can be regarded as PSLGs, allowing them to be implemented as DCELs [77, 82]. Observe, however, that due it its nature, Voronoi diagrams place additional restrictions on the angles of two consecutive edges delimiting a face. Faces of Voronoi diagrams in the Euclidian space can be accounted as convex polygons, meaning its interior angles must be less than 180 degrees [109].

Among the many features of DCELs, the concept of half-edges, faces and vertices stand out. As a matter of fact, the complete implementation of a DCEL requires that these three different structures be defined accordingly in the C++ code header, as shown in Listing A.2.

Half-edges are utilized to assign orientations to the edges with no orientation in undirected planar graphs. In doing so, they allow and support the usage of next and previous pointers, as was the case with doubly linked lists. Each original edge of the undirected graph is split into two half-edges with opposing orientations and with the same length as the original edge [77, 81]. The specification of half-edge orientations is accomplished by imposing the condition that the half-edges enclosing a face form a closed counterclockwise directed path, see Fig. A.9. In other words,

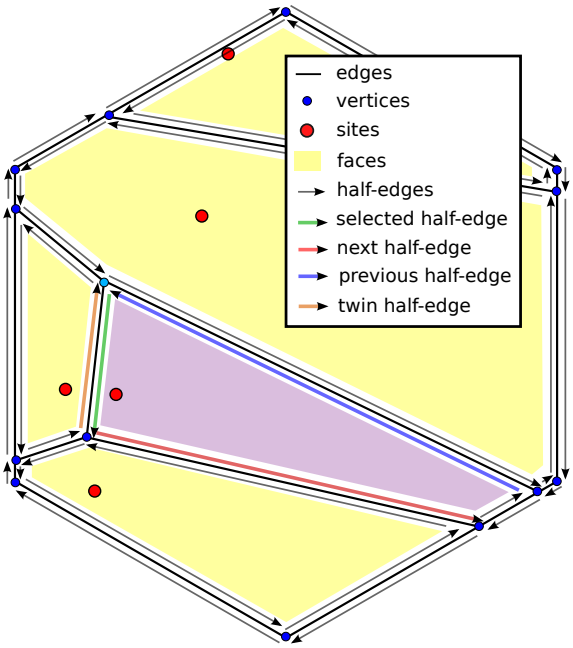

Figure A.9.: Voronoi diagram with five sites (in red) and a hexagonal bounding box. The half-edges, face and vertex associated via pointers with the selected half-edge (in green) are highlighted with different colors. Observe that half-edges delimiting a face always form a closed counterclockwise path. The only exception to this is the half-edge path demarcating the bounding box, the 'outer face', which forms a clockwise path.

after a half-edge of any face of the DCEL is selected, one is able to reach any other half-edge of that face by advancing enough steps via the next or previous half-edge pointers at each half-edge node [77, 81]. It is interesting to notice that if one moves forward enough times by using next pointers in each node, the original half-edge node can be selected multiple times. Conversely, the previous pointer allows one to move along the same half-edge path, but in the clockwise direction. In order to leave the half-edge path surrounding a face, one must resort to the twin pointer, which points to the twin of that half-edge [77, 81]. As already stated, the twins have the same length as the original edge but aim at opposing directions, meaning that they posses different origin vertices. Moreover, each one belongs to a different face. Pointers to both face and origin vertex nodes are stored on each half-edge node (see Listing A.2), and since they are unique to every single half-edge, they allow the program to distinguish half-edges, even twins [77, 81]. Note, that aside from the pointers aformentioned, no data is in fact stored on half-edge nodes.

Faces also receive their own class due to their importance, as is the case with half-edges. In general, in its defining struct, only a single pointer to any half-edge node belonging to the face is stored [77,81]. In case the pointer does not link the face node to the sought half-edge, one can simply resort to previous or next pointers. In our

Listing A.2: Possible DCEL data structure C++ implementation.

```
1  struct halfedge{
2          halfedge *next,*prev,*twin;//pointers to next, previous and twin halfedges
3          vertex *origin;//pointer to the origin vertex of this halfedge
4          face *f;//pointer to the face this halfedge belongs to
5          halfedge() : next(0), prev(0), twin(0), origin(0), f(0) {} //halfedge
               constructor: all pointers are first defined as null to avoid
               segmentation fault issues
6  };
7
8  struct face{
9          halfedge *hedge;//pointer to any halfedge of this face
10         point site;//site of corresponding Voronoi face: not a requirement!
11         face(halfedge *edge) : hedge(edge), site(0,0) {}//face constructor: any
               face edge can be provided when the new operator is called
12 };
13
14 struct vertex{
15         point origin;//vertex point coordinates
16         halfedge *hedge;//pointer to any halfedge emerging from this vertex
17         vert(point p) : origin(p), hedge(0) {}//vertex constructor: point p must be
               provided when new operator is called
18 };
```

implementation of Voronoi diagrams, face nodes additionally keep the Voronoi site coordinates which define the cell. That is done to subsequently simplify the point location query algorithm, even though it is not a prerequisite to its implementation.

Finally, we discuss the vertices, the last of the three constituents, that, when combined, form the DCEL. The vertices, as well as the faces and half-edges, are also managed by a struct specified in the code header, see Listing A.2 [77, 81]. Vertex nodes are the ones responsible for storing the Voronoi vertices coordinates, which are, in reality, the only data, apart from pointers, maintained by the DCEL. Vertex nodes are joined to the rest of the DCEL data structure through a single pointer to any half-edge emerging from it, that is, any half-edge that possesses the vertex as its origin in space [77, 81]. Observe that more than one half-edge can have the same vertex as its origin. As a result, there can be many vertex pointer in different half-edge nodes pointing to the same vertex node. The converse, however, is generally not true. Therefore, in order to select a half-edge node emerging from some vertex, which is not pointed to by any pointer, one can use the following instructions: first, proceed to the half-edge stored in the vertex node, then move to the previous half-edge node and finally reach for its twin half-edge. The vertex pointer stored on the selected half-edge node points to the same orginal vertex. In case the retrieved half-edge is not the one sought, the process can be repeated until the desired half-edge is found. Thus, the apparent limitation of the data structure is overcome. In a similar fashion, any other possible inquiry can be worked out with an appropriate algorithm, due to the practical and convenient nature of the DCEL data structure.

A.4. Voronoi diagram validation

As discussed in Sec. A.1, a Voronoi diagram of a set S of sites possesses a collection of properties which guide its construction. As a rule, a general planar graph does not fit this restrictive number of characteristics. Consequently, if one were to manually determine the corresponding Voronoi diagram of any set S it would be reasonable and wise to perfom validity checks, to ascertain whether the produced diagram is in fact the Voronoi diagram of that set. Visually, for a small set of sites it is easy to observe whether the formed Voronoi diagram is at least plausible or not. For a large number of sites, however, this qualitative method of validation becomes impractical. Hence, a quantitative and easy-to-apply method of validation must be employed. As a matter of fact, although, computationally, an error free Voronoi diagram output is expected, implementation mistakes can lead to inacurracies, such as missing vertices or edges, for instance. As a result, validity checking computer generated diagrams is imperative to avoid obtaining unreliable results. Fortunately, to circumvent this problem, it is possible to take advantadge of a theorem, which follows from the Voronoi diagram definition.

Theorem 1. *Let v be the number of vertices and e, the number of edges of a Voronoi diagram with s sites over the plane. For diagrams with $v \geq 3$, it always follows that $v \leq 2s - 5$ and that $e \leq 3s - 6$ [81].*

The proof, which can be found in most computational geometry textbooks [81, 82], resorts to Euler's formula for planar graphs, a well-known relation applicable to any connected, finite planar graph, see Fig. A.8b. Finite graphs are bounded, i. e., all edges are depicted as segments, as opposed to half-lines, e.g., the Voronoi diagram of Fig. A.9. Euler's formula states that:

$$v - e + f = 2 \tag{A.1}$$

where f is the number of faces in the graph, including the outer face [81]. That relation can be shown to hold for any finite, connected graph through the following reasoning: if the graph is not a tree-like graph, that is, $f > 1$ for that graph, remove any edge completing a cycle from the graph; notice that when that is done, both f and e decrease by 1 and, consequently, $v - e + f$ remains constant; repeat the process until only the outer face remains, i. e., a tree-like graph is produced ($f = 1$); since $e = v - 1$ for all tree-like graphs, it immediately follows that $v - e + f = 2$ for any such graphs; as a result, the original graph complies with relation Eq. A.1 by induction; finally, since no assumptions were made about the form of the original graph, it follows the relation must hold for any finite, connected graph.

Considering that bounded Voronoi diagrams are finite, connected graphs, they, as a consequence, satisfy Eq. A.1. It turns out that verifying whether the generated Voronoi diagram obeys Euler's formula is already a fast and effective validity check, as f,v and e can be readily obtained after DCEL construction. Nonetheless, Euler's

formula is more general, while Theorem 1 invokes the following, more particular property of Voronoi diagrams:

$$e \geq \frac{3}{2}(v+1) \tag{A.2}$$

it can be demonstrated that Voronoi diagrams always obey this result [81]. However, in contrast to the Euler's formula, relation Eq. A.2 cannot, in general, be applied directly to a bounded diagram, since boundary vertices and edges only serve the purpose of enclosing the diagram and do not stem from the Voronoi diagram properties. Having excluded the boundary edges and vertices, some edges become half-lines. In order to be able to apply relations Eq. A.1 and Eq. A.2 to the remaining diagram, a vertex placed at infinity is conceived, where the half-line edges terminate, see Fig. A.10. Provided this modification is made, it is straightforward to see that there is a maximum to the number of edges in the diagram. As discussed in Sec. A.2.1, every vertex is connected to at least 3 edges, including the vertex placed at infinity, hence the $v+1$ in Eq. A.2. Therefore, the minimum number of edges in Voronoi diagrams must be $3(v+1)$ divided by 2 so that each edge is not computed twice. After all, each vertex is connected to at least 3 others through the edges. Finally, Theorem 1, can be easily demonstrated substituting Eq. A.1 into Eq. A.2 and keeping in mind that f is equal to s for a Voronoi diagram [81]. We must also consider the fact that a vertex at infinity must be introduced ($v \rightarrow v+1$), so Eq. A.1 can be applied.

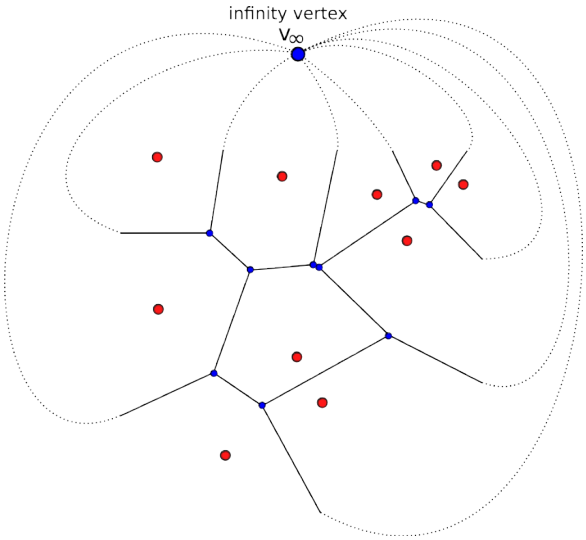

Figure A.10.: Unbounded Voronoi diagram with a vertex introduced at infinity where the half-line edges terminate.

Recalling the considerations made at the beginning of this section, we stress that every Voronoi diagram generated using the algorithm presented in Sec. A.2 was validated using both Theorem 1 and Euler's formula, before being utilized with other purposes.

A.5. Point location, Voronoi diagrams and the Nearest-neighbor search

In this section, we take the last step towards the implementation of a program capable of solving the NN search problem with an optimal time complexity. To accomplish that, a simple and quick point location algorithm is presented [78]. Once the associated point location data structure is in place, it will provide us with the capacity to carry out NN searches in $O(\ln n)$ time.

In computational geometry, point location refers to the problem of identifying which partition a query point resides in, among the many partitions of a planar subdivision [81]. When the planar subdivision is, in fact, a Voronoi diagram, point location becomes an efficient and clear-cut method of performing NN searches, since all points inside a face lie closer to its corresponding Voronoi site, as discussed in Sec. A.1.

Among the set of algorithms available that tackle the point location problem, the easy-to-follow slab decomposition algorithm was the earliest to achieve optimal point location search time, $O(\ln n)$ [78]. However, it falls short when comparared to other more recent point location algorithms, such as the Kirkpatrick or trapezoidal decomposition algorithms [111,115]. That is a result of the inefficient $O(n^2)$ memory storage performance associated with the slab decomposition data structure, which is in constrast to the optimal $O(n)$ behavior of the latter algorithms. Nonetheless, seeing that the search time behavior of the slab decomposition algorithm is optimal and taking into account the fact we had a virtually unlimited memory pool at our disposal, there was no reason to adopt a memory efficient algorithm which, in contrast, entailed an ardous implementation. Therefore, due to its conceptual simplicity and straightforward implementation, it turned out to be the preferred one in this work.

The essence behind slab decomposition is the slicing of the planar subdivision into several parallel slabs [78]. That is done so as to facilitate subsequent binary searches based on the query coordinate normal to the sliced interfaces. Once the slab containing the query point is identified, another binary search is performed to locate which section of the slab encloses it, see Fig. A.11. Finally, after that section is determined, the algorithm immeadiately traces the section back to one of the faces of the planar subdivision, ultimately determining the query point location.

Notice that the slicing is performed by demanding that interfaces intersect each vertex of the planar subdivision, see Fig. A.11. As a result, slabs widths are non-uniform and no vertices reside within them. Moreover, partitions inside each slab stem naturally from the presence of edges crossing them. The algorithm takes advantadge of this fact by associating a binary search tree with each slab. Therefore, by its very nature, the slab decomposition data structure can be understood as a binary search tree of binary search trees. The first BST bases its search criterion solely on one of the plane coordinates, e.g., the x coordinate in the case shown in Fig. A.11, while the searches on slab BSTs must resort to both coordinates to

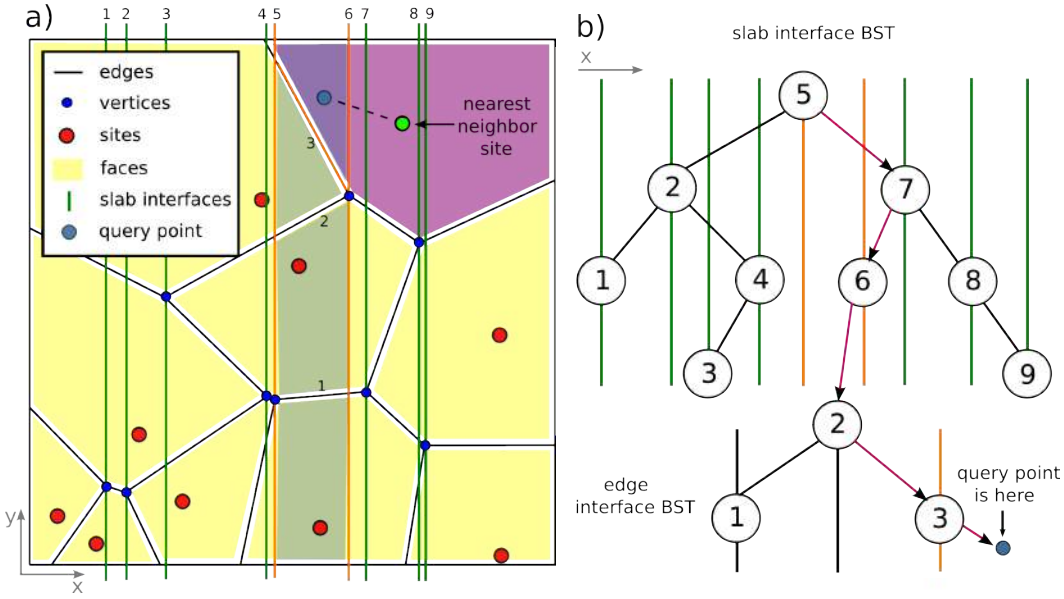

Figure A.11.: (a) Slab decomposition of the Voronoi diagram presented in Fig. A.1. The slab, face and face section containing the query point are shown in a different colors. The slab interfaces enclosing the highlighted slab are also stressed. (b) Chart representing both binary search trees which enabled the identification of the query point location. The path followed during the search down the trees is indicated with arrows. The numbers inside the tree nodes do not represent the actual coordinates. They are simply presented so as to facilite algorithm understanding.

unequivocally locate the query point. For instance, examine the middle face sections enclosed by interfaces 7 and 8 in Fig. A.11. Recognition that both the y and x coordinates are needed is immediate.

The last tricky point when building the slab decomposition data structure is identifying which face of the planar subdivision contains the encountered slab section. Pointers to planar graph faces are stored on the leaf nodes of the second-tier BSTs to settle this problem. The faces, for instance, may be defined as a class, in accordance with the DCEL data structure, as discussed in Sec. A.3. Once the face is found, the search is over.

As for Voronoi diagrams, this concludes our discussion regarding its application to the NN search. Finally, we analyze the search performance of the slab decomposition algorithm. Considering that two binary searches are necessary to determine the query point location and that binary searches are perfomed in $O(\ln n)$ time, it is possible to conclude that search behavior of the algorithm itself is $O(\ln n)$. That follows from the fact that proportionality constants are not explicit in the using the big O notation [77, 78, 81].

B. Appendix: Computational Fluid Dynamics - CFD

B.1. CFD Overview

Computational fluid dynamics is a field dedicated to solving the equations governing fluid flow numerically [32, 33]. It offers many method to solve the equations and powerful tools to the study of flows in a wide range of systems. The CFD approach is particularly useful in engineering designs, since, very frequently, constructing small models which scale the flows may be, at times, nearly impossible for real systems, such as the flow around an aircraft or a ship [32]. Flow through porous media is no exception [116]. Needless to say that creating physical models for porous media, which mimics real systems can be a difficult problem, e.g., the modeling of porous bed rocks to determine the petroleum flow [116,117]. Hence, employing CFD techniques to porous media problems may be of great use.

The CFD approach, however, has its limitations. It is important to consider that the obtained numerical solutions are ultimately *approximations* [32, 33]. In fact, there are at least three types of approximations performed to solve the equations numerically. The first type is related to the approximations and idealizations in the equations that govern the flow. Examples are the mathematical model equations for incompressible, inviscid, potential or Stokes flows, or even the boundary layer approximation [31–33]. The second category of approximations stems from the discretization of the equations of the mathematical models over the mesh representing the system [32, 33]. Generating a mesh is an essential procedure when solving the equations numerically as it enables solution computation at a finite set of nodes [31]. Solving the governing equations numerically would otherwise be impossible. Discretization methods and mesh properties will be discussed later in this chapter. Finally, the last type of approximations originates from the use of iteration methods to solve the discretized equations. Solving the algebraic equation systems with direct methods is costly and generally pointless [32]. Errors due to discretization are much larger than machine precision, leaving no reason to solve the discretized equations that accurately [32]. These approximations, in general, can be accounted for and treated. Defining appropriate criteria for solution **convergence**, **stability**, equation **consistency**, **conservation** of physical quantities, **boundedness** and **accuracy** of solutions helps estimate and minimize the impact of systematic errors and ensure the validity of the solutions [32, 33]. All things con-

sidered, we have chosen to employ CFD to solve the problem of fluid flow through the target applications, due to the great flexibility and simplicity that it provides.

B.1.1. OpenFOAM Overview

The CFD package used in this work was OpenFOAM, a powerful, free, open-source CFD toolkit [89]. OpenFOAM has been successfuly employed to solve a variety of problems relevant both to the industry and the scientific community [118–121]. It offers many utilities, applications and tools for all three major steps that appear in any CFD approach for solving fluid flow numerically: pre-processing, solving and post-postprocessing [89], see Fig. B.1. Pre-processing involves geometry preparation and mesh generation. The solving step encompasses the discretization of the governing equations and actual solving of the corresponding algebraic equation system. Finally, the post-processing part covers the analysis of the results.

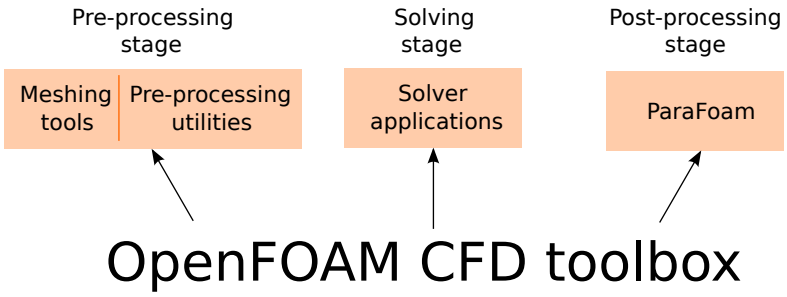

Figure B.1.: Different application categories OpenFOAM provides for each step of the CFD process [89].

In this work, we have used pre-processing tools such as *surfaceCheck* to verify the validity of the .STL geometry files and *snappyHexMesh* for mesh generation. Moreover, in the solving step we have employed a standard OpenFOAM *solver* application, *pimpleFOAM*, and a user modified version of the same application. We highlight that *solver applications* are codes provided within the OpenFOAM package and they differ from the normal usage word *solvers* has in the context of CFD, refer to methods used to solve the algebraic system of equations [32,89]. As a matter of fact, *solver applications* can even use different *solvers* for each field [89]. Finally, it is also important to stress that OpenFOAM *solvers* uses the finite volume (FV) method to discretize the governing equations [89], which will be discussed in Sec. B.6. Regarding the post-processing step, OpenFOAM provides the paraFoam tool to visualize the solution and manipulate the data [89].

B.1.2. Solving CFD problems using OpenFOAM

OpenFOAM is not provided with a graphical user interface (GUI) [89]. Aside from paraFoam, which has a GUI, all other steps are generally run from the terminal. Before running the commands, however, we set up a case with the directory structure shown in Fig. B.2 in order to solve our problem in OpenFOAM 3.0.0. The most important folders are the *constant*, the *system* and the 0 folder[1] [89]. Many text files must be included in each of these folders. They are important since they allow the user to edit simulation parameters by modifying the text files.

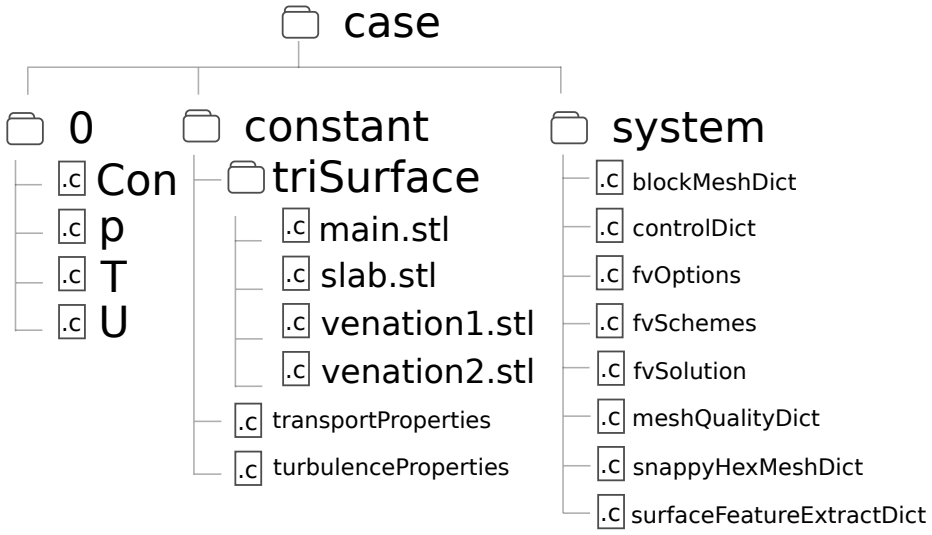

Figure B.2.: Initial directory structure of each of our OpenFOAM cases.

In this work, the most important files employed in the pre-processing step are the *blockMeshDict,* the *surfaceFeatureExtractDict,* the *snappyHexMeshDict* and the *MeshQualityDict* dictionary files, all located in the *system* folder of the case, see Fig. B.2. The solving phase uses other equally important dictionaries. Under the *system* folder there are the $fvSolution$, $fvSchemes$, $fvOptions$ and $controlDict$ dictionaries. They define *solver* and *algorithm* choices, discretization properties, porous medium traits and important simulation features respectively. In the *constant* folder, there are the *transportProperties* and *turbulenceProperties,* which allow the user to set important material properties of the fluid and flow regime, which for our problem was set to *laminar*, see Fig. B.2. Lastly, under the 0 folder, we specify the boundary conditions (BCs) and initial conditions of the problem. Each field has its own file, see Fig. B.2. The velocity field boundary conditions can be altered in the U file while the pressure field boundary conditions may be modified editing the p file. Additional fields will have their corresponding file. For instance, in this work, we have included an additional scalar field which corresponds to reactant concentration. Its corresponding BC file is Con, see Fig. B.2.

[1]The 0 folder is named after the initial time we chose. The initial time could be different though.

A detailed explanation of the procedure necessary to set up a case, as well as a description of each dictionary file and its parameters are out of the scope of this text, but can be found in the OpenFOAM user guide [89]. Throughout this appendix chapter, however, some of the most important aspects regarding solver and algorithm choice will be discussed and additional features of some of the dictionary files will be presented.

Lastly, we declare that the examples in the following sections were adapted from material found in our references, mostly from Ferziger, Versteeg, Lomax or Pozrikidis [32,33,90,91]. This was done in an attempt to make this review of the CFD approach and the FV method as brief and comprehensive as possible.

B.2. Incompressible Navier-Stokes equations

In this section, in Sec. B.3 and Sec. B.4 we briefly describe the equations governing the flow in the target applications of this work. Here, we offer a quick derivation of the incompressible Navier-Stokes equations, which are central to most studies of fluid flow.

The Navier-Stokes equations can be obtained by a careful analysis of conservation of momentum acting on a fluid parcel moving along a streamline[2]. Although there are other ways to derive the Navier-Stokes equations, we choose this method, which can be found in Pozrikidis [91], due to its simplicity. We start the derivation by observing that the rate of change of momentum of a fluid parcel $\frac{d\boldsymbol{M}_p}{dt}$ must be equal to the surface and body forces acting on the fluid parcel, in agreement with Newton's second law,

$$\frac{d\boldsymbol{M}_p}{dt} = \boldsymbol{F}^S + \boldsymbol{F}^B = \iint_{Parcel} \boldsymbol{n} \cdot \boldsymbol{\sigma} dS + \iiint_{Parcel} \rho \boldsymbol{g} dV \tag{B.1}$$

where $\boldsymbol{\sigma} \cdot \boldsymbol{n}$ is the traction $\boldsymbol{f}$ which is exerted on the surface of the fluid parcel, while $\boldsymbol{\sigma}$ is the stress tensor. The unit vector $\boldsymbol{n}$ is normal to the surface of the parcel and points outwards. Notice the only body force being considered is the one due to gravity. The next step consists in rewriting the rhs of Eq. B.1 by recalling that the momentum $\boldsymbol{M}_p$ is

$$\boldsymbol{M}_p = \iiint_{Parcel} \rho \boldsymbol{u} dV$$

Inserting the equation above into the rhs of Eq. B.1, we obtain,

[2]For definitions of what is a fluid parcel, control volume, streamline, traction and stress tensor refer to [91].

$$\frac{d\boldsymbol{M}_p}{dt} = \frac{d}{dt}\iiint_{Parcel} \rho \boldsymbol{u} dV = \iiint_{Parcel} \frac{d(\rho \boldsymbol{u} dV)}{dt}$$

We stress that the interchange between the time derivative and the volume integral is possible since the time derivative in question is the total derivative. In other words, we are using the Lagrangian approach, which follows the fluid parcel along its trajectory. This approach takes into account any observable change in time of parcel volume and is in contrast to the Eulerian approach, in which the study of the flow is done by fixing control volume to a position [91]. By exploiting the simple form of the conservation of mass in the Lagrangian formalism $\frac{d\delta m_p}{dt} = \frac{d(\rho dV)}{dt} = 0$, which simply states that the mass of a fluid parcel cannot change in time, we obtain the following:

$$\frac{d\boldsymbol{M}_p}{dt} = \iiint_{Parcel} \frac{d(\rho \boldsymbol{u} dV)}{dt} = \iiint_{Parcel} \left(\rho \frac{d\boldsymbol{u}}{dt} dV + \boldsymbol{u} \frac{d(\rho dV)}{dt} \right) = \iiint_{Parcel} \rho \frac{d\boldsymbol{u}}{dt} dV$$

Finally, we use the Gauss theorem to transform the surface integral of the traction in Eq. B.1 into a volume integral. We are, then, allowed to express the conservation of parcel momentum only in terms of volume integrals,

$$\iiint_{Parcel} \rho \frac{d\boldsymbol{u}}{dt} dV = \iiint_{Parcel} \nabla \cdot \boldsymbol{\sigma} dV + \iiint_{Parcel} \rho \boldsymbol{g} dV$$

Since the volume of integration is arbitrary and the combined integrals yield zero, it follows the integrand itself must be null. We, thus, retrieve a differential equation known as Cauchy's differential equation [91]:

$$\rho \frac{d\boldsymbol{u}}{dt} = \nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{g} \tag{B.2}$$

Expanding the total derivative of the velocity field $\boldsymbol{u}$ in terms of partial derivatives, we derive an alternative form of Cauchy's differential equation,

$$\rho \frac{\partial \boldsymbol{u}}{\partial t} + \rho \boldsymbol{u} \cdot \nabla \boldsymbol{u} = \nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{g} \tag{B.3}$$

The stress tensor $\boldsymbol{\sigma}$ must be specified, so we can solve the equations above for a particular case and obtain the evolution of the velocity field with respect to time and spatial coordinates. The relations which specify the components of the stress tensor $\boldsymbol{\sigma}$ are known as the constitutive relations for the fluid in question [91], since they establish the way the fluid properties are modeled. The Navier-Stokes equations

(NS equations) are retrieved by employing the Newtonian model for a viscous fluid [91]. The components of $\boldsymbol{\sigma}$ for a Newtonian viscous fluid according to the Einstein summation convention are:

$$\sigma_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right) \tag{B.4}$$

where μ is the dynamic viscosity of the fluid and p is the static pressure exerted on the parcel surface. If we take the fluid as being incompressible, then the condition $\boldsymbol{\nabla \cdot u = 0}$ must be satisfied. That follows from the continuity equation:

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla \cdot j} = \frac{\partial \rho}{\partial t} + \boldsymbol{\nabla}\cdot(\rho \boldsymbol{u}) = \frac{\partial \rho}{\partial t} + \boldsymbol{u \cdot \nabla}\rho + \rho \boldsymbol{\nabla \cdot u} = \frac{d\rho}{dt} + \rho \boldsymbol{\nabla \cdot u} = 0 \tag{B.5}$$

where $\boldsymbol{j}$ is the mass flux, and from the definition of an incompressible fluid as a fluid which has its parcel volume conserved along the flow. From the conservation of mass of the fluid parcel in the Lagrangian formalism, we obtain:

$$\frac{d(\rho\delta V)}{dt} = \delta V \frac{d\rho}{dt} + \rho\frac{d\delta V}{dt} = -\rho\delta V \boldsymbol{\nabla \cdot u} + \rho\frac{d\delta V}{dt} = 0 \Rightarrow \boldsymbol{\nabla \cdot u} = \frac{1}{\delta V}\frac{d\delta V}{dt}$$

For this reason the divergence of the velocity field is also called expansion coefficient. Since parcel volume is preserved for an incompressible fluid, the incompressibilty condition takes the sought form:

$$\boldsymbol{\nabla \cdot u} = 0 \tag{B.6}$$

In other words, the density ρ of the fluid parcel remains constant as it is convected in a flow. Naturally, that occurs because we are dealing with an incompressible fluid. Using this condition, we can compute the divergence of the stress tensor of a Newtonian fluid with uniform viscosity and retrieve:

$$\boldsymbol{\nabla \cdot \sigma} = -\boldsymbol{\nabla} p + \mu \boldsymbol{\nabla^2 u}$$

Combining this result with Eq. B.3, we find the NS equations for an incompressible fluid in the Eulerian form [91]:

$$\rho\frac{\partial \boldsymbol{u}}{\partial t} + \rho \boldsymbol{u \cdot \nabla u} = -\boldsymbol{\nabla} p + \mu \boldsymbol{\nabla^2 u} + \rho \boldsymbol{g} \tag{B.7}$$

We can rewrite the advection term in the above equation, using the continuity equation for incompressible flows $\boldsymbol{\nabla \cdot u} = 0$ as

$$\boldsymbol{u \cdot \nabla u} = \boldsymbol{\nabla}\cdot(\boldsymbol{uu}) - (\boldsymbol{\nabla \cdot u})\boldsymbol{u} = \boldsymbol{\nabla}\cdot(\boldsymbol{uu})$$

In its integral form, the NS equations may then be written as [31]:

$$\frac{\partial}{\partial t}\iiint_V \rho \boldsymbol{u} dV + \iint_S (\rho \boldsymbol{u}\boldsymbol{u} \cdot d\boldsymbol{S}) = -\iint_S \boldsymbol{\sigma} \cdot d\boldsymbol{S} + \iiint_V \rho \boldsymbol{g} dV \tag{B.8}$$

where the stress tensor σ takes the form of Eq. B.4, and V and S represent the volume and surface of the fluid parcel.

The incompressible Navier-Stokes equations along with the continuity equation and an energy conservation equation are sufficient to determine many types of flows, ranging from laminar to turbulent regimes [91]. That occurs whenever the incompressibilty assumption is valid, which is usually the case for liquids. The set of equations are non-linear due to the advective term. Hence, turbulent flow tends to occur whenever the advective term $\rho \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u}$ is much greater than the vicous term $\mu \boldsymbol{\nabla^2 u}$ [31, 91]. In fact, a method to analyse when the flow transitions from the laminar to the turbulent regime is to simply observe the ratio between the advective term and the viscous term. When the typical dimensions of the problem are used this ratio becomes the widely known Reynolds number [31, 91]:

$$\frac{\rho \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u}}{\mu \boldsymbol{\nabla^2 u}} \Rightarrow \frac{\rho U^2 L^2}{\mu L U} = \frac{\rho U L}{\mu} = Re \tag{B.9}$$

where U and L are the typical velocities and length scale of the flow problem. Many flow types can be estimated by determining the Reynolds number Re [31, 32, 91]. For instance, the Reynolds number is important in engineering designs. In order to simulate a real scale flow with a small model, the Re in the small model must match the Re of the normal scale flow [32]. Typically, the transition between the laminar and turbulent regime occurs when $10^3 < Re < 10^4$. In the target applications considered in this project the Re is much lower than 10^3. Hence, we will be working in the laminar regime [31].

As a last remark, we highlight that the Navier-Stokes equations provide us with an accurate mathematical model, which can account for and describe real flows. However, for turbulent flows, two-phase flows, combustion etc., the difference between the actual flow and the behavior predicted by the mathematical models, the *modeling error*, may be very large, resulting in a *qualitatively* wrong solution. Fortunately, in the laminar regime these errors are neglibible [32].

B.3. Transport equation

In order to study the transport of a scalar field ϕ, which can be a measure of reactant concentration, for instance, we must make sure that the field satisfies the transport

equation given below [31]:

$$\frac{\partial(\phi)}{\partial t} + \boldsymbol{\nabla}\cdot(\boldsymbol{u}\phi) = \boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi) + q_\phi \tag{B.10}$$

where ρ is the density of the fluid, D is the diffusion coefficient and q_ϕ is a source term, which can account for sources and sinks in the system. This relation allows the evolution of ϕ in time to be determined. We may rewrite the above equation as:

$$\frac{\partial(\phi)}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}(\phi) + \phi\boldsymbol{\nabla}\cdot\boldsymbol{u} = \boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi) + q_\phi$$

Considering the diffusivity D as constant and the fluid as incompressible, we may write:

$$\frac{\partial\phi}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}\phi = D\boldsymbol{\nabla}^{\mathbf{2}}\phi + q_\phi \tag{B.11}$$

The main properties of the transport are the advection, represented by the second term on the left side, which is a measure of the tendency of ϕ to be transported along the flow and the diffusion transport, accounted for by the first term on the right side. The diffusion coefficient D is a property of the medium but it also depends on the geometrical features of the colloids which are being diffused. The famous Stokes-Einstein-Sutherland equation allows one to estimate D for spherical particles by knowing their radius r and the viscosity μ of the fluid [122]:

$$D = \frac{k_B T}{6\pi\mu r} \tag{B.12}$$

where k_B is Boltzmann constant and T is the thermodynamic temperature. The theoretical estimate of the effective diffusion coefficient D_{eff} in a porous medium is an even more complex problem. It involves properties such as the porosity, permeability, tortuosity and etc. which are in general very hard to predict. Hence, the determination of D_{eff} of porous media is in most cases made empirically using methods such as fluorescence correlation spectroscopy (FCS) or nuclear magnetic ressonance (NMR) microscopy [123], for example.

In OpenFOAM, the transport equation is solved using a *scalarTransport* algorithm [89], always after the momentum equations are solved at that time step [33].

B.4. Darcy-Brinkman equation for porous media

In a previous section, we have discussed the incompressible Navier-Stokes equations. With these equations, the appropriate boundary conditions and the continuity equation, it is possible to determine a wide range of flows. In the case of the geometries

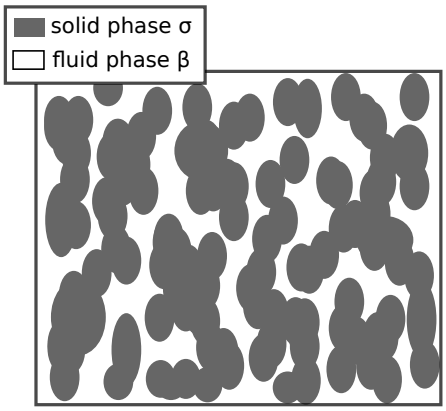

Figure B.3.: Representation of a porous media cross-section. The fluid phase is in white, while the solid phase is in gray.

discussed in Chapter 2, it is possible to simulate the flow within the channels using them. Inside the porous medium, on the other hand, the situation calls for greater attention.

A porous medium is a complex structure comprising, in general, a solid phase σ and at least one fluid phase β, see Fig. B.3 [116]. The fluid phases reside inside the pores, which are the void spaces whose volume will be called V_β when the fluid phases completely fill the pores. Naturally, flow occurs through these spaces. The volume of the solid phase, on the other hand, is V_σ. That said, in principle, it is possible to determine the flow through a porous medium using the momentum and continuity equations and the appropriate boundary conditions [116]. In fact, this is one of approaches used to model porous media. It is called *direct numerical simulation* (DNS) and has been employed successfully in some fields [116]. For instance, in the petroleum industry, rock samples from oil reservoir can be extracted from the ocean floor and their structure can be determined in high resolution using techniques such as X-ray computed microtomography [124]. DNS is then possible since the boundary conditions are known. The greatest drawback of DNS is that the boundary conditions tend to be extremely complex [116]. Solving such problems using CFD demands a very fine mesh, which, in turn, requires considerable computing power. Solving the flow through very large samples is simply unfeasible. To make matters worse, the boundary conditions may even depend on time, making the solving process even more cumbersome. Sometimes solving the flow with a high degree of detail is completely unnecessary or even undesirable. In these cases, another approach has to be used.

The alternative is to use a macroscopic approach [116]. The quantities of interest such as the velocity and pressure fields are averaged according to the Representative Elementary Volume (REV) appropriate to the problem. The details of the complex structure of the pores, that is, the internal geometry features of the porous medium and the associated boundary conditions can then be disregarded [116]. That makes

the solving process much more straighforward. The boundary conditions are enormously simplified. For instance, specifying the boundary conditions at inlet, oulet and the system walls, together with the continuity of the fields at the interface between the channels and the porous media are enough to solve the problems in this work. The difference is that the fields determined are macroscopic fields, in other words, they are averaged fields and do not contain any local details of the flow in the actual medium.

When the averaging procedure is performed an additional term naturally arises in the governing equations for the macroscopic fields. In order to see this observe the following. For a Stokes flow, the governing equations for the local fields are [31, 32, 91, 116]:

$$\begin{cases} \boldsymbol{\nabla} \cdot \boldsymbol{v} = 0 & in\ V_\beta \\ -\nabla p + \rho \boldsymbol{g} + \mu \boldsymbol{\nabla^2 v} = 0 & in\ V_\beta \end{cases} \tag{B.13}$$

where these equations govern the flow that occurs in the β phase. It can be shown that when the Stokes equations are averaged, the following equation is retrieved [116]:

$$-\nabla \langle p \rangle^\beta + \rho \boldsymbol{g} + \mu^* \boldsymbol{\nabla^2} \langle \boldsymbol{v} \rangle - \mu K^{-1} \cdot \langle \boldsymbol{v} \rangle = 0 \tag{B.14}$$

where the average operator $\langle . \rangle$ is the average of the field over the control volume of volume V, e.g., $\langle \phi \rangle = \frac{1}{V} \int_{v_\beta} \phi dV$, where μ^* is an effective viscosity, where K^{-1} is the inverse of the *permeability* tensor and where $\langle . \rangle^\beta$ obeys the following relation:

$$\langle \phi \rangle = \epsilon \langle \phi \rangle^\beta \tag{B.15}$$

where ϵ is the volume ratio between the void spaces and the total volume $\epsilon = \frac{V_\beta}{V}$, which is also known as the *porosity*, a property that varies from one porous medium to another [117]. Equation Eq. B.14 is known as the Darcy-Brinkman equation [116]. It was proposed by Brinkman [125], who used a theoretical approach to determine it. Moreover, it is closely related to the Darcy equation, which was formulated by Henri Darcy in 1856 on the basis of empirical evidence [126]:

$$\langle \boldsymbol{v} \rangle = -\frac{K}{\mu} \cdot \left(\nabla \langle p \rangle^\beta - \rho \boldsymbol{g} \right) \tag{B.16}$$

where K is the *permeability* tensor, which is a property specific to the porous medium one is working with [117]. Although there are theoretical models that can be used to calculate the permeability K and the porosity ϵ [127], these are limited, sometimes even producing inaccurate results [128]. Due to this reason, these material properties are usually determined experimentally as was the case with the effective diffusion coefficient D_{eff} [123], discussed in Sec. B.3.

The extra term which appears in Eq. B.14, $\mu^* \boldsymbol{\nabla^2} \langle \boldsymbol{v} \rangle$, is often negligible in magnitude when compared to the Darcy term $\mu K^{-1} \cdot \langle \boldsymbol{v} \rangle$ [116], as was the case for the flow through beds of sand measured by Darcy. The extra term, which is, in fact, the vicous term that appears in the NS equations with a modified viscosity, is related to the drag forces that arise when one layer of fluid moves in relation to another. In contrast, the Darcy term has to do with the drag forces due to the viscous friction at the interface between the fluid and solid phases of the medium [116]. These forces can, in general, slow down the flow considerably. The greater the interface area between the solid and fluid phases, the greater the Darcy term becomes in comparison to the viscous term. In commom porous media, the interface area tends to be large and, more often than not, it is possible to disregard the viscous term to a first approximation.

In OpenFOAM, the porous media is modelled by including a source term in the Navier-Stokes equations [129]:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u} = -\boldsymbol{\nabla} p' + \nu \boldsymbol{\nabla^2} \boldsymbol{u} + \boldsymbol{g} + \boldsymbol{S} \tag{B.17}$$

where ν is the kinematic viscosity, p' is the pressure divided by the fluid densisty ρ and $\boldsymbol{S}$ is a source term given by the following equation:

$$\boldsymbol{S} = \left(\nu \boldsymbol{d} + \frac{1}{2} \boldsymbol{f} \left| \boldsymbol{v} \right| \right) \cdot \boldsymbol{v} \tag{B.18}$$

where both $\boldsymbol{d}$ and $\boldsymbol{f}$ are tensors of the porous medium which are defined in a local set of coordinates. The tensor $\boldsymbol{d}$ is related to the Darcy term, while $\boldsymbol{f}$ is related to the Forchheimer term, which is only siginificant for flows with greater velocities, where the inertial terms play a greater role. Hence, in our case, we consider $\boldsymbol{f}$ as null. Moreover, in the case of a homogeneous porous medium, which is the case of our study, the tensor $\boldsymbol{d}$ becomes a constant, d. We can then relate it to the permeability of the homogeneous medium k, which, naturally, is also constant through the following relation:

$$d = \frac{1}{k} \tag{B.19}$$

Hence, the incompressible Navier-Stokes equations in the porous medium becomes:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \boldsymbol{\nabla} \boldsymbol{u} = -\boldsymbol{\nabla} p' + \nu \boldsymbol{\nabla^2} \boldsymbol{u} + \boldsymbol{g} + \frac{\nu}{k} \boldsymbol{v} \tag{B.20}$$

where the Darcy term now becomes evident. Except for the inertial terms, the terms in Eq. B.20 have a counterpart in Eq. B.14. These terms do not appear in Eq. B.14 because we had initially considered a creeping flow. Although the advective terms

appear in Eq. B.20 even inside the porous medium, the terms on the right side dominate the flow and we retrieve Eq. B.14. Additionally, Eq. B.20 can account for transient problems as well due to the transient term on the left side, a trait which benefits us. Inside the channels, on the other hand, d is null and the laminar flow can be determined with the unmodified NS equations.

Finally, one must bear in mind that the velocity and pressure fields that appear in Eq. B.20 are locally averaged [127], that is, we are favoring the macroscopic approach over the DNS approach. At last, all that is required for the solving procedure to start are the boundary conditions. At the interface between the porous medium and the channels they are the continuity of the velocity and pressure fields as well as the continuity of the stress tensor [127]. Although that is an approximation, as there is no pressure jump, the approach can be considered as valid [127]. In OpenFOAM, the porous medium properties, such as the $\boldsymbol{d}$ and $\boldsymbol{f}$ tensors, can be assigned to a region of the mesh, or cellZone, by setting up a *fvOptions* dictionary in the *system* folder of the case [129].

B.5. Meshes

Meshes or *grids* define discrete locations where the variables, that is, the unknown field values, must be computed [32]. Hence, they make possible the partitioning of the geometric domain of some target problem. That is important as this allow for discretization of the governing equations, making them solvable numerically. Each point location of the mesh, the nodes, can be associated with some subdomain of the space, a cell [32]. In the finite volume approach, the nodal points are associated with the center of mass of the control volumes, in other words, the cells are the CVs [32]. In order for the mesh to be considered valid, it must satisfy different levels of constraints, which are imposed on the points, edges, faces and cells [32]. Presenting the criteria for mesh validity, in particular, mesh validity in OpenFOAM, is out of the scope of this text, but a thorough explanation may be found in the user manual [89]. Here, we only stress that mesh treatment in OpenFOAM is very robust and general and that it can account for cells of arbitrary shape with an unlimited arbitrary number of polygonal faces [89]. Next, we discuss additional properties of the meshes, which influence how a geometrical domain is represented.

B.5.1. Orthogonal and Non-orthogonal meshes

An important trait of the mesh, which influences the overall discretization process, is whether the cells of the mesh are orthogonal or not [32]. There are different definitions of orthogonality. Here, we consider that two cells are orthogonal with respect to each other if the line connecting two nodes inside them is parallel to the vector normal to the face dividing the cells [32]. In contrast, if the line is not

parallel to the face normal, the cells are considered non-orthonal. Following this definition, the mesh is considered orthogonal if all cells are orthogonal with respect to their neighbors and non-orthongal otherwise. For instance, cartesian grids are orthogonal, see Fig. B.5, while the meshes shown in Fig. B.4 are not. Although the treatment of non-orthogonal meshes is more cumbersome, they do a much better job at accurately representing the complex geometries generated [32]. Therefore, all of the meshes generated in this work using OpenFOAM tools are non-orthogonal. Fortunately, OpenFOAM is equipped not only to generate these meshes with tools such as *snappyHexMesh,* but to treat these arbitrary meshes efficiently [89].

B.5.2. Structured, Block-Structured and Unstructured meshes

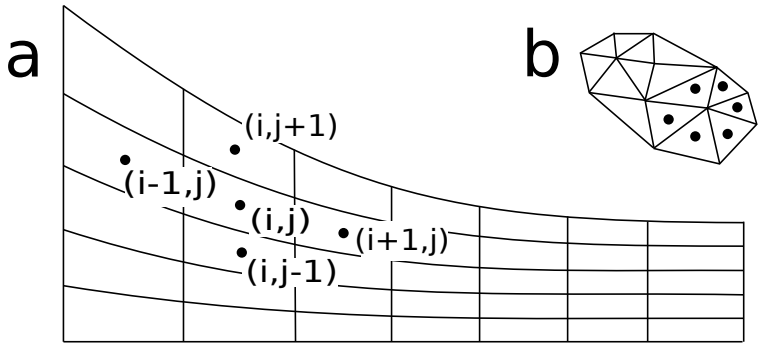

Figure B.4.: Structured (a) and unstructured (b) non-orthogonal meshes.

Meshes can also be classified according to another property. In order to understand what that property is, first observe that some meshes can be depicted by families of lines running from one end of the mesh to the other [32]. Such meshes may possess a property dictating that all lines, members of one family, cross members of another family only once, while not crossing a member of the same family even once, see Fig. B.4a. The meshes that possess this property are called *structured*. This trait allows for the nodes to be uniquely identified according to their position within the domain by a number of indices which equals the number of dimensions of the problem, that is, (i, j) for 2D or (i, j, k) for 3D. This ability makes representing these meshes numerically a straightforward task. The downside is that meshes with this property, in general, are not ideal for representing complex geometries.

A second possibility, which can account for more complex geometries, are *block-structured* meshes. These are simply various *structured* meshes, representing domains of the geometry, which are assembled to represent the system. This additional partitioning allows for a critical domain of the geometry to be represented with additional detail, improving the overall efficiency with which the computational

resources are used. Inside each domain, the mesh is treated as *structured*. Only at the interfaces between the different domains, some care must be taken. Frequently, the benefits of this type of mesh out-weigh this drawback [32]. OpenFOAM offers the *blockMesh* utility, with which the user may generate *structured* and *block-structured* meshes [89].

Finally, the most arbitrary type of mesh, which can represent complex geometries with an even greater accuracy, is the *unstructured* mesh, see Fig. B.4b. Nodes cannot be recovered using a simple set of indices and, thus, more complex data structures must be employed to manage the mesh. This type of mesh can be employed with finite volume schemes without major problems [32]. OpenFOAM, a CFD package which uses the finite volume approach, has all of the built-in functions necessary to deal with these meshes [89]. In fact, it even provides the *snappyHexMesh* utility, a mesh generating tool which can create *unstructured* meshes from CAD geometries [89].

B.5.3. Collocated and Staggered arrengements

Another aspect of the meshes we must analyze is whether we choose to store all field variables at the same nodes or not. In particular, that is important when discretizing the momentum equations. In case we store all field variables at the same nodal points, the mesh is called *collocated* [32,33]. Although *collocated* meshes are simple, they have some disadvantages. For instance, difficulties with velocity-pressure coupling arise in *collocated* arrengments [32, 33]. Oscillations in pressure can also occur. Finally, highly non-uniform pressure fields can act as uniform [33]. Part of the reason is that in order to estimate the pressure gradient at a given node using the central differencing scheme, pressure values will have to be previously interpolated to the CV faces [33]. Information about the non-uniformity of the field is generally lost in the process. A detailed discussion of how this occurs is given in Versteeg Chapter 6 [33]. Collocated arrengements, however, offer some advantages when dealing with complex geometries which use non-orthogonal meshes. It gained popularity when better pressure-velocity coupling algorithms, discussed in Sec. B.7, were developed in the 1980's [32]. Meshes in OpenFOAM employ the *collocated* arrengement.

Alternatively, it is possible, for instance, to store the components of velocity values at the cell face centers instead, while continuing to store the pressure field values at the CV center nodes. Now, to compute the pressure gradient at the velocity nodes, interpolation of pressure is not required [33]. This arrengement is called *staggered* and it offers a great advantadge, which is the strong coupling between the velocity and pressure fields [32,33]. The problem of evening out the non-uniform pressure is eliminated in this configuration. On the other hand, when this arrengement is used to deal with complex non-orthogonal meshes, curvature terms which are troublesome to treat numarically arise [32]. This may cause undesired non-conservative errors.

Hence, this approach lost some popularity in the 80's when the improved pressure-velocity coupling algorithms were developed [32].

B.5.4. Convergence criterion

Here we discuss one final aspect. Naturally, when creating a mesh, the finer the mesh representing a system is, the greater the computational effort required to solve the problem. Additionally, as one would guess, the mesh choice has an influence over the solution [32]. For instance, the solution depends on mesh refinement. The solution on a coarse mesh usually differs from the solution on a finer mesh representing the same problem. However, if the numerical experiments are performed on meshes with successive levels of refinement for a linear problem, one generally finds that the solution converges to a *mesh-independent* solution if the numerical method used the problem is **stable** and **consistent** [32]. Finding the mesh-independent solution is therefore related to this **convergence** problem, being part of the result validation procedure [32]. As a rule, care must be taken, as bad choices of mesh can lead to innacurate results [32, 127].

B.6. Finite Volume Methods

The main trait of finite-volume discretization methods is that they stem from the integral form of the governing equations, see, for instance, Eq. B.8 [31–33]. Although finite volume discretization methods have some drawbacks, they offer two great advantadges over the finite difference and finite element discretization approaches. The first advantadge has to do with the conservation of physical quantities. Naturally, the governing equations are in their differential are conservative. That may not the case for the discretized equations [32]. The FV discretization method, however, is conservative by construction and conservation is generally assured as long as the fluxes through the faces of adjacent CVs are identical [32,90]. The second advantadge refers to coordinate transformations. Generally, employing coordinate transformations is necessary when irregular meshes are used. Again, that is not required when the FV formulation is employed, making straightforward the process of applying FV methods to unstructured meshes consisting of arbitrary polyhedra [90]. Hence, it is not surprising to learn that the OpenFOAM package is able to handle any type of polyhedra, since the discretization method used by it is the FV formulation [89].

In the FV formulation, computational nodes are assigned to the center of each CV in the mesh [32]. The integrals in the governing equations are then computed in terms of the field values at these nodes. Whenever field values at points other than the mesh nodes are needed, they are determined using an interpolation method. Hence, one of the drawbacks of the FV approach is that it has three levels of approximation: differentiation, integration and interpolation [32]. In this section, we discuss, to some

extent, schemes for each of these three levels of approximation. In particular, we present some of the schemes used to solve the flow in the target applications of this work.

B.6.1. Methods for approximating the integrals

As discussed above, in order to solve the integral governing equations numerically, the volume and surface integrals must be approximated in terms of the field values at the mesh nodes. Initially, we discuss the simplest approximation for the surface integral over the faces of a CV in a Cartesian 2D grid, see Fig. B.5. The generalization for 3D non-orthogonal and unstrutured meshes can be encountered in Ferziger [32]. Next, we offer another simple approximation for the volume integral over a CV. The approximation methods presented for both the surface and volume integrals are second-order schemes.

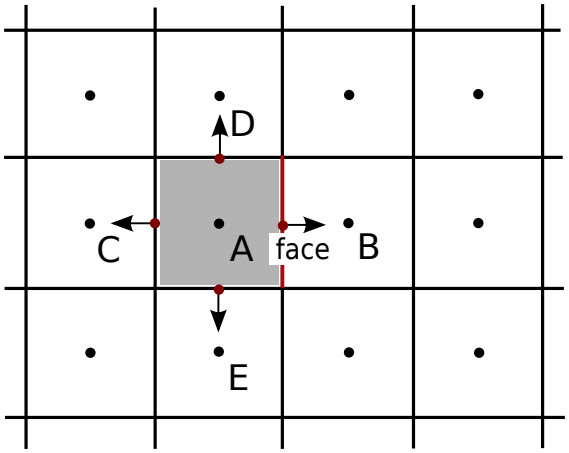

Figure B.5.: 2D Cartesian mesh. The nodes in black store the field values of the CV. Nodes in red correspond to the field values calculated at the faces via interpolation.

A surface integral for a CV contaning N faces can be written as

$$\int_S f dS = \sum_k^N \int_{S_k} f dS = \sum_k^N F_k$$

where f is the component normal to the faces of the CV in the integrands of any surface integrals, such as the advection and stress tensor surface integrals in Eq. B.8. Next, the integrals over each face that appear in the sum must be approximated. The simplest approximation is the midpoint rule. In this approximation, the integral over the face is taken as the product of the face area and the value of the integrand

at the center f_k [32]. Notice that, in fact, it is the mean value of the integrand $\overline{f_k}$ over the face that is being approximated for the value at the face center f_k. This leads to:

$$F_k = \int_{S_k} f dS = \overline{f}_k S_k \approx f_k S_k \tag{B.21}$$

Naturally, the value f_k is not initially known and must be approximated employing some interpolation method which uses the values of f at the CVs center [32]. Observe that the interpolation method must also have second order-accuracy so that the overall approximation may be considered a second-order scheme [32].

The approximation employed to compute the volume integrals over the CV volumes is even simpler. The integral of a source term q over a CV volume ΔV, for instance, is approximated as the product of the source term at the center of the CV and ΔV. Observe that, in this approximation method, the mean value of q over the CV volume is being approximated by the value of q at the center of the CV [32]. Thus,

$$Q = \int_V q dV = \overline{q} \Delta V \approx q_{center} \Delta V$$

where Q is a volume integral term over any CV volume and q_{center} is the value of q at the center of the CV. This time, the values of the source terms q_{center} are already available at the mesh nodes and do not need to be interpolated. In higher-order schemes, however, that is not the case. A discussion on higher-order schemes is of no interest here since they were not employed in our work. The approximations schemes presented here already provide a good compromise between accuracy and efficiency [32]. In particular, they are excellent for complex geometries. Finally, note that only the principles of the approximations schemes are being presented. Here, these schemes are shown in the context of an orthogonal mesh, although they still apply even when *non-orthogonal* or arbitrary unstructured meshes are used. When they are used for such meshes, care must be taken, since there are many additional features which were not described here which are out of the scope of this text. A detailed discussion on this topic, however, may be found in textbooks on CFD [32].

B.6.2. Interpolation methods

Interpolation is necessary when computing field values at points other than the CV centers [32,33,91]. It becomes specially important when computing the contribution to the flux of a surface integral term over a CV face, see Sec. B.6.1. Nevertheless, interpolation may also be necessary depending on the approximation scheme used to compute the volume integrals. In this section, we present two simple interpolation methods. Both were used in this work and are widely employed in CFD.

In order to understand both schemes, imagine we seek the value ϕ_{face} of a scalar field ϕ at the face dividing the CV A from CV B, see Fig. B.5. The field values at

the CV centers ϕ_A and ϕ_B are known here as well as the velocity field $\boldsymbol{v}$ at the face. The first method we analyze is known as the upwind interpolation [32], given by the following equation:

$$\phi_{face} = \begin{cases} \phi_A & if\ (\boldsymbol{v} \cdot \boldsymbol{n})_{face} > 0; \\ \phi_B & if\ (\boldsymbol{v} \cdot \boldsymbol{n})_{face} < 0. \end{cases} \tag{B.22}$$

where $\boldsymbol{n}$ is the unit vector normal to the CV A pointing towards the CV B. Notice that this method approximates the value at the face by the upstream value: if the flow is leaving CV A, ϕ_{face} is approximated to ϕ_A and vice-versa. The value of the velocity field $\boldsymbol{v}$ was assumed known here, but in practice it must be computed as well. In a collocated arrengement, such as the one used by OpenFOAM, each velocity component must be interpolated at the cell center.

The second interpolation method that can be employed to find the field value ϕ_{face} is the linear interpolation. Again the interpolation is performed between values at the center of the CVs which the face connects. In the case of Fig. B.5, for instance, one may compute the value ϕ_{face} using the linear interpolation scheme with the following formula:

$$\phi_{face} = \alpha_{face}\phi_A + (1 - \alpha_{face})\phi_B \tag{B.23}$$

where the coefficient α_{face} is given by:

$$\alpha_{face} = \frac{x_{face} - x_A}{x_B - x_A}$$

Other interpolation methods are discussed thoroughly on the references [32]. Generalizations of the methods discussed above for non-orthogonal meshes are out of the scope of this text but can be found in Ferziger [32].

B.6.3. Truncation and discretization errors

We analyze here the truncation and discretization errors associated with the approximation schemes. The upwind interpolation is a first-order scheme, while the linear interpolation has a second-order accuracy. This is easily demonstrated for the case shown in Fig. B.5. We analyze the upwind interpolation scheme first. To do so, we take the velocity field direction at the face as $(\boldsymbol{v} \cdot \boldsymbol{n})_{face} > 0$ and expand the field ϕ_{face} about the node a using a Taylor series:

$$\phi_{face} = \phi_a + (\Delta x)\left(\frac{\partial \phi}{\partial x}\right)_a + (\Delta x)^2\left(\frac{\partial^2 \phi}{\partial x^2}\right)_a + O((\Delta x)^3) \tag{B.24}$$

where Δx, which, in fact, is $x_{face} - x_a$, can be taken as a measure of the mesh spacing [33]. Moreover, $O\left((\Delta x)^3\right)$ represents terms of higher order. When comparing equations Eq. B.22 and Eq. B.24, we see that the lowest order term we throw

away is the first order term. That is the reason why the upwind scheme is a first order scheme [33]. The terms thrown away, the *truncated* terms, are reponsible for the error associated with the approximation, known as the *truncation error* [32,33]. Naturally, the lower the value of the mesh spacing Δx, that is, the more refined the mesh is, the lower the *truncation error* tends to be if the discretization method is **consistent** [32, 33]. Hence, it is seems wise to refine the mesh as much as possible. On the other hand, hardware limitations impose a restriction on refinement. Ideally, it is desirable to analyze the **convergence** and determine the *mesh-independent* solution [32]. In order to do so, we can estimate the the *discretization errors,* which are the difference between exact solution of the differential and the exact solution difference equation generated for a given mesh [32]. The *discretization errors* tend to zero as the mesh spacing goes to zero if the numerical method is **convergent** [32]. Analizing the **convergence** is part of the validation procedure and, in general, that is done by computing the solution first on a coarser mesh and subsequently refining the mesh [32, 127].

Another more general way to look at the *truncation* and *discretization errors* is in the following. We express, for instance, the transport equation for a scalar field ϕ in the form [32]:

$$\mathcal{L}(\phi) = L_{mesh}(\phi) + \tau_{mesh} = 0 \tag{B.25}$$

Here the operator $\mathcal{L}$ represents the differential equation, L_{mesh} is the operator associated with the discretized equation in a particular mesh, now a *difference* equation, and where τ_{mesh} is the *truncation error.* We see that τ_{mesh} is the difference between the differential and the difference equations. It considers, of course, contributions from all approximations done during the discretization procedure. Notice that the scalar field ϕ here is the exact solution of the *differential* equation. The exact solution of the difference equation ϕ_{mesh}, on the other hand, satisfies the following equation:

$$L_{mesh}(\phi_{mesh}) = 0$$

The relation between ϕ and ϕ_{mesh} can be expressed as:

$$\phi = \phi_{mesh} + \epsilon_{mesh} \tag{B.26}$$

where ϵ_{mesh} is the *discretization error.* More details on *truncation* and *dicretization errors*, as well as the validation procedure may be found in many CFD textbooks [32, 33].

B.6.4. Discretization of the diffusion equation

The general transport equation for the scalar field ϕ in its differential form is given by the following expression [32, 33]:

$$\frac{\partial(\rho\phi)}{\partial t} + \boldsymbol{\nabla}\cdot(\rho\boldsymbol{u}\phi) = \boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi) + q_\phi \tag{B.27}$$

where ρ is the density of the fluid, D is the diffusion coefficient and q_ϕ represents sources and sinks of the scalar ϕ in the CV. The first term on the left side, the one responsible for the time evolution of ϕ, is the transient term, while the second one is the advective term, which accounts for the transport of ϕ along the flow. For the sake of simplicity, following an example from Versteeg [33], we neglect both of these terms and consider the simplest transport process possible: pure diffusion in the steady state. The diffusion process, which is accounted for the first term on the right side, is given by the following equation:

$$\boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi) + q_\phi = 0 \tag{B.28}$$

The equation above is always valid when describing the transport of ϕ at any given point. When working with CFD, however, we discretize the space by generating a mesh. For FV methods, in particular, we are interested in the integral form of the Eq. B.28 over the CVs of the mesh. Equation Eq. B.28 then becomes:

$$\int_{CV} \boldsymbol{\nabla}\cdot(D\boldsymbol{\nabla}\phi)dV + \int_{CV} q_\phi dV = \int_A \boldsymbol{n}\cdot(D\boldsymbol{\nabla}\phi)dS + \int_{CV} q_\phi dV = 0 \tag{B.29}$$

where A is the closed surface of the CV. The problem then consists in obtaining the solution for a set with N of coupled integral equations, where N is the number of CVs in the mesh [33]. Naturally, the boundary conditions must be provided. In this case, the PDE is elliptical, meaning that either Dirichilet or Neumann boundary conditions must be specified along the closed surface encompassing the system, see Tab. 1.2. Finally, in order to solve the integral equations numerically, we can resort to the approximation methods discussed in Sec. B.6.1 and Sec. B.6.2. The gradient at the faces of the mesh is usually approximated using central differencing, which is simply a linear interpolation [33]. For the 1D problem, see Fig. B.6, extracted from Versteeg [33], Eq. B.29 becomes, when applied to the CV P:

$$\int_A \frac{d}{dx}\left(D\frac{d\phi}{dx}\right)dS + \int_{CV} q_\phi dV = \left(DA\frac{d\phi}{dx}\right)_e - \left(DA\frac{d\phi}{dx}\right)_w + \overline{q}\Delta V = 0 \tag{B.30}$$

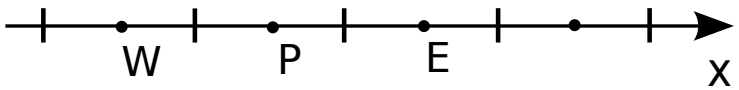

Figure B.6.: 1D mesh utilized to obtain a numerical solution for the 1D diffusion problem.

In case D varies along the mesh, the values of the diffusion coefficient at the faces can be averaged using the values at the nodal points v:

$$D_w = \frac{D_W + D_P}{2}$$

$$D_e = \frac{D_P + D_E}{2}$$

The gradients are evaluated using central differencing as [33]:

$$\left(DA\frac{d\phi}{dx}\right)_e = D_e A_e \left(\frac{\phi_E - \phi_P}{\delta x_{PE}}\right)$$

$$\left(DA\frac{d\phi}{dx}\right)_w = D_w A_w \left(\frac{\phi_P - \phi_W}{\delta x_{WP}}\right)$$

The source term can depend on ϕ in many situations. In these cases, the FV method usually approximates the term by using a source model with the following linear form [33]:

$$\overline{q}\Delta V = q_P + q'_P \phi_P$$

We may then insert these expressions back into Eq. B.30, rearrange the terms and retrieve the following algebraic equation [33]:

$$a_P \phi_P = a_W \phi_W + a_E \phi_E + q_P \tag{B.31}$$

where:

a_W	a_E	a_P
$\frac{D_w}{\delta x_{WP}} A_w$	$\frac{D_e}{\delta x_{PE}} A_e$	$a_W + a_E - q'_P$

We may rewrite equation Eq. B.31 as:

$$a_P \phi_P + \sum_l a_l \phi_l = q_P \tag{B.32}$$

where the sum runs over the neighbor nodal points of P and the coefficients a_l incorporate the minus sign [33]. The node P and its neighbors form the *computational molecule*, which may be generalized to 2 and 3 dimensions [32].

After discretizing all N integral equations, we see that this approach yields a system of N algebraic equations, one for each CV of the mesh. In this case, in which we treat a diffusion problem, the governing equation is linear and the discretization process clearly produces a linear system of algebraic equations, that is, the coefficients a and the source term q do not depend on the unknowns. Here, the unknowns are the nodal values of the scalar field ϕ. The linear sytem may then be written in the matrix form as:

$$\boldsymbol{A\Phi = Q} \tag{B.33}$$

For the one-dimensional structured mesh of the preceeding diffusion problem and its corresponding computational molecule, we may choose to write the matrix $\boldsymbol{A}$ in a tridiagonal form [32] whereas for the problems in 2 or 3 dimensions, which require the use of some type of $2D$ or $3D$ computational molecule, $\boldsymbol{A}$ adopts other forms. For structured meshes $\boldsymbol{A}$ has a banded structure [32] while for unstructured meshes, which are used to represent complex geometries, that is not the case. In all cases, $\boldsymbol{A}$ is sparse and techniques which exploit this trait are generally employed in order to save computer memory [32].

Finally, we stress that most of the work in CFD revolves around solving equations of form given in equations Eq. B.32 and Eq. B.33. There are many methods which can be used to solve a linear system of equations [32]. Solvers with either a direct or iterative approach may be employed [32]. When dealing with a more general non-linear problem, such as the one described by the NS-equations, only iterative approaches are employed, as the coeffiecients and source terms may depend on the unknown variables [32]. Moreover, when using unstructured meshes, different iterative solvers must be used. Later in this appendix chapter, we will sketch out the main concepts behind iterative methods.

As a last remark, we stress that the approximation techniques used in this section to discretize the diffusion equation, the midpoint rule, the central difference approximation and linear interpolation are often the best option when dealing with unstructured meshes with CVs of an arbitrary number of faces, as they offer a good compromise between accuracy and efficiency, generality and simplicity [32]. Hence, these discretization schemes have been selected to solve the transport in the geometries of this work. OpenFOAM allows the user to chose between the different discretization schemes in the *fvSchemes* dictinary located in the *system* folder of the case [89].

B.6.5. Explicit vs. Implicit methods and Stability

Up until now, we have only analyzed the *steady* problem governed by Eq. B.28. In this subsection, we will briefly inspect fundamental traits of *unsteady* problems which will be important when discussing the algorithms used to solve the momentum equations. Moreover, we will offer a very concise introduction to **stability** analysis. Again, if a more thorough explanation is sought, it can be found in some of the CFD textbooks used as references. We also declare that much of the upcoming discussion was extracted from Ferziger [32].

We start the discussion with a simple first order ordinary differential equation with respect to time. The initial condition is also provided. Hence,

$$\frac{d\phi(t)}{dt} = f(t, \phi(t)) \,; \qquad \phi(t_0) = \phi^0 \tag{B.34}$$

where our problem is that we seek the solution $\phi(t)$ for a time greater than t_0. Naturally, in order to solve the equation numerically, time discretization is necessary.

Therefore, we discretize time using a very short uniform spacing Δt, the time step. So as to simplify the following discussion, we introduce the notation for future discrete times $t_1 = t_0 + \Delta t$, $t_2 = t_1 + \Delta t$ and so on, as well as the respective solutions at those times ϕ^1, ϕ^2 and etc.

Integrating Eq. B.34 in time from t_n to t_{n+1}, we obtain the following exact relation:

$$\int_{t_n}^{t_{n+1}} \frac{d\phi}{dt} dt = \phi^{n+1} - \phi^n = \int_{t_n}^{t_{n+1}} f(t, \phi(t)) dt$$

The second integral above could be exactly evaluated if the correct choice of time τ was made, yielding $f(\tau, \phi(\tau))\Delta t$ as a result. That is guaranteed by the mean value theorem of calculus. Unfortunately, τ is not known and, therefore, it must be approximated. If it is approximated by t_n, we obtain the following result:

$$\phi^{n+1} = \phi^n + f(t_n, \phi^n)\Delta t \tag{B.35}$$

The equation above offers a method to obtain the solution at the next time step, allowing for time advancement. The method illustrated in Eq. B.35 is known as the *forward* or *explicit Euler* method [32]. *Explicit methods* are characterized by allowing one to determine the solution at the next time step, based only on the solution and on variables values from the *previous* time step. Here, ϕ^{n+1} is evaluated using the ϕ^n and t^n values. This usually makes time advancement a straightforward process.

On the other hand, if τ is approximated using t^{n+1} instead, we have:

$$\phi^{n+1} = \phi^n + f(t_{n+1}, \phi^{n+1})\Delta t \tag{B.36}$$

Equation Eq. B.36 provides a different method to determine ϕ^{n+1} called *backward* or *implicit Euler* [32]. *Implicit* methods, as opposed to *explicit* ones, require that the solution is computed in terms of itself an the variables at the current time step. Notice that in Eq. B.36, ϕ^{n+1} is computed in terms of ϕ^n, ϕ^{n+1} and t_{n+1}. As one would guess, computing ϕ^{n+1} using *implicit* methods is usually a much more cumbersome process, requiring ϕ^{n+1} at all nodes of the mesh to be solved simultaneously in the best case scenario [31]. Nevertheless, this extra trouble sometimes pays off, as *implicit* methods often allow large time steps to be used without *instability*, a property that can reduce the computational cost of the numerical method considerably, specially when solving *steady* problems. That is because the implicit Euler method is *unconditionally stable* [32].

There are many other different approximations methods available depending on the choice of τ. They will not be discussed here, as they are out of the scope of this text, which only aims to offer a succint introduction.

Observe that both the *explicit* and *implicit Euler* method produce good results when Δt is small [32]. When a larger Δt is chosen, the error associated with the discretization increases in the same manner, as both are first order methods. If the errors

at a given time step are not magnified as the computation of ϕ at the next time steps proceeds, the numerical method is called **stable** [32]. Naturally, the stability depends on the whole discretization process, including mesh resolution and the equations governing the flow problem. In particular, the stability of complicated, non-linear problems are difficult to analyze. Therefore, one usually investigates the stability of the methods for linear problems with constant coefficients and no boundary conditions and then extrapolates the analysis to the more complex problems [32]. More ofttern than not, this approach does work [32]. In fact, one of the most widely employed methods of stability analysis proposed by *John von Neumann* includes some of these ideas. For instance, *Neumann* argued that boundary conditions are seldom the source of stability problems and that ignoring them so as to simplify the analysis is justified [32].

Next, in order illustrate this discussion, we will analyze the stability of the one dimensional transport equation with constant velocity and fluid properties and without any sources nor sinks:

$$\frac{\partial \phi}{\partial t} = -u\frac{\partial \phi}{\partial x} + \frac{D}{\rho}\frac{\partial^2 \phi}{\partial x^2} \tag{B.37}$$

Discretizing the above equation using the *upwind differencing scheme* (UDS) for the spatial derivatives, which is an approximation for the first derivative is analogous to the upwind interpolation method presented in Sec. B.6.2, and the *explicit Euler* scheme for the time derivative, we obtain the following difference equation:

$$\phi_i^{n+1} = \phi_i^n + \left[-u\frac{\phi_i^n - \phi_{i-1}^n}{\Delta x} + \frac{D}{\rho}\frac{\phi_{i+1}^n + \phi_{i-1}^n - 2\phi_i^n}{(\Delta x)^2}\right]\Delta t \tag{B.38}$$

where i is the index of the mesh node being considered. The equation may be rewritten in the following manner:

$$\phi_i^{n+1} = (1 - 2d - Co)\,\phi_i^n + d\phi_{i+1}^n + (d + Co)\,\phi_{i-1}^n \tag{B.39}$$

where the dimensionless parameters d and Co are:

$$d = \frac{D\Delta t}{\rho\,(\Delta x)^2} \tag{B.40}$$

$$Co = \frac{u\Delta t}{\Delta x} \tag{B.41}$$

where d is the ratio between the time step and the characteristic diffusion time and where Co, the *Courant number,* an important parameter in CFD, is the ratio between the time step and the characteristic advective time [32]. In general, if the

method is stable, a quantity such as the measure given below must decrease with time:

$$\epsilon = \sqrt{\sum_i \left(\phi_i^n - \phi_i^{n-1}\right)^2} \tag{B.42}$$

In this context, the norm must *decrease* through dissipation due to the form of the differential equation Eq. B.37. It follows that if one of the coefficients in Eq. B.39 is negative, the norm could potentially *increase* as time goes on. Hence, it is desirable that $1 - 2d - Co > 0$. Therefore, the condition below must be satisfied:

$$\Delta t < \frac{1}{\frac{2D}{\rho(\Delta x)^2} + \frac{u}{\Delta x}} \tag{B.43}$$

When there is no advection, the restriction on the time step for stability becomes:

$$\Delta t < \frac{\rho\,(\Delta x)^2}{2D} \tag{B.44}$$

In contrast, when diffusion is negligible, the stability condition is:

$$Co < 1 \tag{B.45}$$

Or simply:

$$\Delta t < \frac{\Delta x}{u} \tag{B.46}$$

When employing the actual *von Neumann* stability analysis similar conclusions are reached [32]. Equation Eq. B.45 is an important result. In OpenFOAM, the user may actually monitor the *Courant number* during the solving process so as to observe whether the stability condition is being met [89]. The condition may also be interpreted as a restriction on the movement of a fluid parcel. If $Co > 1$, it will move more than one mesh spacing Δx per time step [32]. As information could be lost in the process, the condition seems very reasonable, but unfortunately it is restrictive. In OpenFOAM, there is an algorithm which does not require Eq. B.45 to be satisfied [96]. The reasons will be discussed in Sec. B.7.3. Finally, we stress that the stability analysis for other implicit methods can be found in many CFD textbooks [31, 32, 90] and are out of the scope of this text.

B.6.6. Solving the algebraic system of equations

In the Sec. B.6.4, we saw that one of the central problems in CFD is to solve equations of the type Eq. B.33 numerically. To that end, either direct or iterative methods

can be employed. Direct methods yield *exact* solutions to a system of linear algebraic equations, while iterative methods offer an *approximate* solution, which is obtained by taking an initial guess and improving the guessed solution using the governing equations [32]. The sources of error when using the direct approach are the *modeling* (see Sec. B.2) and *discretization errors* (see Sec. B.6.3). The drawback when employing direct methods is that they are generally more expensive computationally. Iterative methods, on the other hand, introduce another source of error, in addition to the other two: the *iteration errors* [32]. In pratice, a solution with *iteration* errors smaller than the *discretization* errors may often be reached with a lower computational cost. Whenever that is possible, there is no reason to solve the algebraic system exactly [32]: the *discretization errors* present in the solution offset the possible benefits of obtaining an exact solution to an already approximated algebraic system. Moreover, when dealing with a non-linear systems, employing iterative methods is a must [32]. Hence, as iterative methods are generally more cost-effective, being applicable to non-linear problems, they can be found in virtually all CFD packages, OpenFOAM being no exception. In the following discussion, we provide a basic of analysis of iterative methods. These can be applied not only to more relatively simple problems such as the linear transport equation for a scalar field, but may be extented to the much more complex problem of solving the non-linear incompressible Navier-Stokes equations to obtain the pressure and the velocity fields [32]. Later, in this chapter, we will go into the details of how to solve the incompressible NS-equations using common pressure-corretion algorithms, such as the SIMPLE and PISO algorithms. Finally, we will present the algorithm employed in this work, the PIMPLE algorithm, a mixture of both the SIMPLE and PISO, which is additionally a standard solver implemented in the OpenFOAM toolbox.

We start the discussion of the iterative methods where we left off a previous subsection: with the algebraic system of equations in the matrix form,

$$\boldsymbol{A\Phi} = \boldsymbol{Q} \tag{B.47}$$

As stated, when employing iterative methods, an initial solution guess is made and then improved upon every iteration. After n iterations, we obtain an approximate solution $\boldsymbol{\Phi^n}$, which does not satisfy the system of algebraic equations exactly. In addition to $\boldsymbol{Q}$, a leftover term $\boldsymbol{\rho^n}$, which is a non-zero *residual*, arises when we apply $\boldsymbol{A}$ to $\boldsymbol{\Phi^n}$. Mathematically, we can express this statement as:

$$\boldsymbol{A\Phi^n} = \boldsymbol{Q} - \boldsymbol{\rho^n} \tag{B.48}$$

Subtracting Eq. B.48 from Eq. B.47, we obtain:

$$\boldsymbol{A}\left(\boldsymbol{\Phi} - \boldsymbol{\Phi^n}\right) = \boldsymbol{\rho^n} \tag{B.49}$$

where the difference between the *exact* solution to the system of algebraic equations $\boldsymbol{\Phi}$ and the *approximate* solution $\boldsymbol{\Phi^n}$ is the *iteration error* $\boldsymbol{\epsilon^n}$:

$$\boldsymbol{\epsilon^n} = \boldsymbol{\Phi} - \boldsymbol{\Phi^n}$$

From Eq. B.49, we see that, in principle, when the residual $\boldsymbol{\rho^n}$ tends zero, so does $\boldsymbol{\epsilon^n}$. In fact, driving the residuals to zero is the main goal of the iteration process. The following iteration scheme for a linear system allows us to see how the process works [32]:

$$M\boldsymbol{\Phi^{n+1}} = N\boldsymbol{\Phi^n} + \boldsymbol{B} \tag{B.50}$$

where $\boldsymbol{\Phi^{n+1}}$ is the solution at $n+1$th iteration step, and $\boldsymbol{\Phi^n}$, naturally, is the approximate solution at nth iteration step. At convergence, $\boldsymbol{\Phi^{n+1}} = \boldsymbol{\Phi^n} = \boldsymbol{\Phi}$. Thus, since the converged solution must satisfy Eq. B.47, we obtain the following relations [32]:

$$M - N = \boldsymbol{A}$$

$$\boldsymbol{B} = \boldsymbol{Q}$$

It is also possible to use a *pre-conditioning matrix* P and rewrite the equations above:

$$M - N = P\boldsymbol{A}$$

$$\boldsymbol{B} = P\boldsymbol{Q}$$

Moreover, if we subtract $M\boldsymbol{\Phi^n}$ from both sides of Eq. B.50, we obtain the following relation [32]:

$$M(\boldsymbol{\Phi^{n+1}} - \boldsymbol{\Phi^n}) = \boldsymbol{B} - (M - N)\boldsymbol{\Phi^n} \tag{B.51}$$

The difference $\boldsymbol{\Phi^{n+1}} - \boldsymbol{\Phi^n}$ is referred to as the *correction* $\boldsymbol{\delta^n}$. If we use the relations obtained for $M - N$ and $\boldsymbol{B}$ and compare Eq. B.51 with relation Eq. B.49, we reach [32]:

$$M\boldsymbol{\delta^n} = \boldsymbol{\rho^n} \tag{B.52}$$

The *correction* $\boldsymbol{\delta^n}$, which is an approximation for the *iteration error,* will be discussed later on when we analyze the SIMPLE and PISO algorithms. For a more thorough explanation on the criteria for convergence, please refer to Ferziger chapter 5 [32], from which most of the previous discussion was extracted. For now, it suffices to say that one of the main goals of iteration methods is to reach convergence fast, since the more iteration steps used to reach the solution, the more inefficient the method becomes.

Finally, we highlight that it is possible to monitor and control many of the parameters discussed in this section when employing an iterative method to solve a CFD case in OpenFOAM [89]. For instance, it is possible to plot the residual $\boldsymbol{\rho^n}$ on the fly with progams such as *gnuplot*. Moreover, the convergence criteria must

be defined in the *fvSolution* dictionary under the *system* folder [89]. That is accomplished by defining a threshold value below which the *residuals* must go for the *iterative solver* to stop [89]. The threshold value is set on the *tolerance* keyword under the subdictionary of each field [89]. There are also a number of different options for *solvers* and *Pre-conditioners* which the user may draw from. For instance, a popular iterative solver, which is implemented in OpenFOAM, is the Gauss-Seidel method. A detailed analysis on this and other *solvers* is out of the scope of this text. An in-depth look at the Gauss-Seidel method, some its enhancements, such as the *successive over-relaxation* (SOR), and *splitting* methods can be found in many CFD textbooks, including Fezinger, Versteeg, Lomax and others [31–33, 90].

B.6.7. Coupled Equations, Sequential solution and Under-relaxation

Up until now we have mostly discussed how to obtain a solution for a scalar field ϕ, governed by a linear equation, such as diffusion equation in Sec. B.6.4. Naturally, this discussion also extends to non-linear equations and vector-fields as well. Solving the flow governed by continuity equation and the non-linear incompressible Navier-Stokes equations, however, possess an additional feature: the equations are coupled [32, 33]. Each component of the velocity vector field $\boldsymbol{v}$ and the pressure field p are unknown. To make matters worse, the pressure field p does not even appear in the continuity equation for an incompressible fluid, see Eq. B.6 [32, 33]. In Sec. B.7, we will discuss in detail how to circumvent this problem. For now, we briefly present two approaches to solve coupled equations: the simultaneous and the sequential method.

As the name suggests, simultaneous solutions are obtained by solving all variables *simultaneously*. Simultaneous solutions with iterative solvers have been developed by many authors and are better applied to problems with linear and tightly coupled equations [32]. Using the simultaneous methods with complex, non-linear equations, such as the ones we face in this work, is too expensive computationally. The preferred approach, in these cases, is to solve each equation as if there was only a single unknown, taking all of the other variables as known [32]. The best values for each variable at a given iteration are used. The equations are solved *sequentially,* forming cycles that stop only when all equations are satisfied. The iterations on each equation are called *inner iterations.* However, it must be noted that after the cycle of iterations performed on each equation of the system, the coefficients a_l and the sources q_l must be updated [32], see Eq. B.32. That is due to the dependence of the coefficients on the unknown variables, which occurs specially in the case of non-linear equations that have been *linearized* [32]. A new cycle of iterations is then executed after the update on these values. The process is repeated until convergence is reached. The cycles are called *outer iterations* [32]. The choice of *inner* per *outer* iterations ratio may optimize performance [32].

Another property that must be controlled in order to stabilize the solution are the *under-relaxation factors* [32]. *Under-relaxation* is employed to limit the change in the variables from one *outer* iteration to the next. Mathematically, this can be performed in the following fashion [32]:

$$\phi^n = \phi^{n-1} + \alpha_\phi \left(\phi^{new} - \phi^{n-1}\right) \tag{B.53}$$

where n is the *index* of the *outer* iteration, ϕ^{new} is the solution for the variable ϕ determined at the end of an *outer* iteration and α_ϕ is the *under-relaxation* factor, whose value must be between 0 and 1. The closer α_ϕ is to 1, the more the newly determined ϕ^{new} influences ϕ^n. Notice that at convergece, $\phi^{n-1} = \phi^n = \phi^{new} = \phi$ and, therefore, the choice of α_ϕ does not influence the solution in that limit, since the term involving it becomes null [32]. An optimal choice of *under-relaxation factors* may speed up convergence greatly, but, alas, the optimal number is hard to predict [32]. A bad choice, on the other hand, may even prevent convergence [32]. *Under-relaxation* factors may be set in OpenFOAM in the $fvSolution$ dictionary under the *system* folder for each field separately [89], see Fig. B.2. More on *under-relaxation* may be found in the CFD textbooks mentioned earlier [32, 96]. Next, we discuss in more detail common algorithms used to solve the Navier-Stokes equations.

B.7. Solving the coupled Pressure-Velocity equations

In this section, we present some of the most common algorithms utilized to obtain the velocity and pressure fields from the coupled pressure-velocity equations. Of great importance to this discussion are the conservation laws. By guaranteeing that the flows satisfy them, we ensure the physical validity of the results. Conversation of momentum is enforced by making the variables satisfy the momentum equations, the discretized incompressible Navier-Stokes equations, while mass conservation is enforced by making the variables satisfy the continuity equation. Conservation of energy for isothermal flows and angular momentum, on the other hand, are consequences of the momentum equations and the choice of discretization [32]. They *cannot* be enforced independently [32]. Hence, special care must be taken when selecting a discretization method. A thorough discussion on energy and angular momentum conservation is out of the scope of this text. More details can be found in Ferziger chapter 7 [32].

When using a sequential iterative method to obtain the solution, the role of the momentum equations are clear: determining the components of the velocity field $\boldsymbol{u}$. After all they are the dominant variables in their respective equations [31, 32]. That leaves as with the continuity equation to determine the pressure p. The problem is that the pressure does not appear in the continuity equation for incompressible flows:

$$\boldsymbol{\nabla} \cdot \boldsymbol{u} = 0 \tag{B.54}$$

In order to circumvent this, we take the divergence of the momentum equations [31, 32]:

$$\boldsymbol{\nabla}\cdot\boldsymbol{\nabla}p = \boldsymbol{\nabla}\cdot\left[-\rho\frac{\partial \boldsymbol{u}}{\partial t} - \rho\boldsymbol{u}\cdot\boldsymbol{\nabla}\boldsymbol{u} + \mu\boldsymbol{\nabla}^{\mathbf{2}}\boldsymbol{u} + \rho\boldsymbol{g}\right]$$

Using the assumptions that both the density ρ and the viscosity μ are constant, already made back in Eq. B.7, and the continuity equation, we obtain [31, 32]:

$$\frac{1}{\rho}\boldsymbol{\nabla}^{\mathbf{2}}p = -\boldsymbol{\nabla}\cdot(\boldsymbol{\nabla}\cdot\boldsymbol{u}\boldsymbol{u}) + \boldsymbol{\nabla}\cdot\boldsymbol{g} \tag{B.55}$$

where we have used the following vector calculus identities to discard the viscous term: $\boldsymbol{\nabla}^{\mathbf{2}}\boldsymbol{A} = \boldsymbol{\nabla}(\boldsymbol{\nabla}\cdot\boldsymbol{A}) - \boldsymbol{\nabla}\times(\boldsymbol{\nabla}\times\boldsymbol{A})$ and $\boldsymbol{\nabla}\cdot(\boldsymbol{\nabla}\times\boldsymbol{A}) = 0$. Using the Einstein notation and neglecting the body force term we may rewrite the equation above as:

$$\partial_i(\partial_i(p)) = -\partial_i(\partial_j(u_i u_j)) \tag{B.56}$$

This is a Poisson equation for the pressure and it is the starting point of the algorithms we will discuss next [31, 32]. By making the pressure satisfy the equation above, we enforce the mass conservation [32]. Equation Eq. B.56, in addition to the momentum equations, see Eq. B.7, the appropriate boundary and initial conditions, allows us to determine the flow of an incompressible fluid through a system.

B.7.1. SIMPLE algorithm

The SIMPLE method is an algorithm developed by Patankar and Spalding in 1972 [130], whose name stands for Semi-Implicit Method for Pressure-Linked Equations. It is essentially a guess-and-correct procedure used to determine the pressure and velocity field, by making them satisfy both the mass and momentum constraints [32, 33]. It can be constructed to solve both *steady* and *unsteady* flows. In OpenFOAM, the SIMPLE algorithm is implemented to solve *steady* problems [89]. Many standard *solvers* in OpenFOAM make use of the SIMPLE algorithm. Parameters of the algorithm can be set in the SIMPLE *subdictionary,* which, in turn, can be found in the $fvSolution$ dictionary under the *system* folder of the case, see Fig. B.2.

B.7.1.1. Explicit time advancement

Before examining the algorithm itself, we use, as our starting point to the algorithm introduction, the semi-discretized momentum equations [32], which are discrete in space but not time:

$$\frac{\partial(u_i)}{\partial t} = -\frac{\delta(u_i u_j)}{\delta x_j} - \frac{1}{\rho}\frac{\delta p}{\delta x_i} + \frac{1}{\rho}\frac{\delta \sigma_{ij}}{\delta x_j}$$

where the terms with $\delta/\delta x$ represent some discretization scheme and where we have considered the densisty ρ of the fluid as a constant. A non-uniform density, however, could have been easily be accounted for [32]. When using an explicit method for time advancement such as the explicit Euler method, we obtain the following equations:

$$u_i^{n+1} - u_i^n = \Delta t \left(-\frac{\delta (u_i u_j)^n}{\delta x_j} - \frac{1}{\rho}\frac{\delta p^n}{\delta x_i} + \frac{1}{\rho}\frac{\delta \sigma_{ij}^n}{\delta x_j} \right) \tag{B.57}$$

Notice that the velocities at the $n+1$th time step can be computed in terms of the known values of the variables at the nth time step. The computed velocities usually do not satisfy the continuity equation. In order to guarantee that the velocity components satisfy the mass conservation constraint, we force the pressure p^n to satisfy the Poisson equation of the type shown in Eq. B.56 before computing the velocities at the new time step [32]. Hence:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta p^n}{\delta x_i}\right) = -\rho \frac{\delta}{\delta x_i}\left(\frac{\delta (u_i u_j)^n}{\delta x_j}\right)$$

B.7.1.2. Implicit time advancement

Now, instead of using an explicit scheme for time advancement, we could have employed some implicit method. Implicit methods have the benefit of allowing the use of large time steps without instability [32]. For the implicit Euler method, we have:

$$u_i^{n+1} - u_i^n = \Delta t \left(-\frac{\delta (u_i u_j)^{n+1}}{\delta x_j} - \frac{1}{\rho}\frac{\delta p^{n+1}}{\delta x_i} + \frac{1}{\rho}\frac{\delta \sigma_{ij}^{n+1}}{\delta x_j} \right) \tag{B.58}$$

With the corresponding Poisson equation for the pressure:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta p^{n+1}}{\delta x_i}\right) = -\rho \frac{\delta}{\delta x_i}\left(\frac{\delta (u_i u_j)^{n+1}}{\delta x_j}\right) \tag{B.59}$$

Obtaining the velocities at the new time step $(n+1)$ from equations Eq. B.58 and Eq. B.59, this time, is a much more difficult problem. The pressures at the new time step can only be computed when the velocities are known and vice-versa. Hence, the equations must be solved simultaneously via an iteration procedure. Additionally, even if the pressures p^{n+1} were known, equations Eq. B.58 are still a large non-linear system of equations. In order solve them, one can either linearize the solutions about the preceeding time step to obtain a linear system, or use the converged solutions from the previous time step as input to an iteration method which will converge the solution at the new time step [32]. If one chooses to linearize the solutions, then:

$$u_i^{n+1} = u_i^n + \Delta u_i$$

$$p^{n+1} = p^n + \Delta p$$

By inserting the expressions above back into equations Eq. B.58, neglecting the second order term $\Delta u_i \Delta u_j$, we obtain:

$$\Delta u_i = \Delta t \left(-\frac{\delta (u_i u_j)^n}{\delta x_j} - \frac{\delta (u_i^n \Delta u_j)}{\delta x_j} - \frac{\delta (\Delta u_i u_j^n)}{\delta x_j} - \frac{1}{\rho}\frac{\delta p^n}{\delta x_i} - \right.$$

$$\left. -\frac{1}{\rho}\frac{\delta \Delta p}{\delta x_i} + \frac{1}{\rho}\frac{\delta \sigma_{ij}^n}{\delta x_j} + \frac{1}{\rho}\frac{\delta \Delta \sigma_{ij}}{\delta x_j} \right) \tag{B.60}$$

The system is now linear with respect to the velocity correction terms. A reasonable approach that can be used to solve the system of equations and obtain the fields at the new time step is to first determine a velocity field u_i^* which does not include the pressure-correction term from Eq. B.60 [32]. The alternating direction implicit (ADI) method, which splits the equations tranforming them into a one dimensional problem may be used to compute u_i^* [32]. Naturally, the velocity field u_i^* does not satisfy the continuity equation. After u_i^* has been computed, we can determine u_i^{n+1} . First, however, observe that u_i^* can be expressed as:

$$u_i^* = u_i^n + \Delta u_i + \frac{\Delta t}{\rho}\frac{\delta \Delta p}{\delta x_i} \tag{B.61}$$

Hence:

$$u_i^{n+1} = u_i^* - \frac{\Delta t}{\rho}\frac{\delta \Delta p}{\delta x_i} \tag{B.62}$$

By taking the divergence of Eq. B.62, we obtain the following Poisson equation for the pressure-correction, knowing the velocity u_i^{n+1} does satisfy the mass conservation constraint [32]:

$$\frac{\delta}{\delta x_i}\left(\frac{\delta \Delta p}{\delta x_i}\right) = \frac{\rho}{\Delta t}\frac{\delta u_i^*}{\delta x_i} \tag{B.63}$$

where we again stress that the fluid density ρ taken as constant. Wrapping up, in order to determine u_i^{n+1} using the implicit time advancement scheme, we first compute u_i^*, then we determine the pressure-correction gradient from Eq. B.63 and finally use Eq. B.62 to calculate u_i^{n+1} [32]. Computation of the variables at the new time steps then proceeds. The error made in linearizing the problem and using large time steps is unfortunately not negligible. Hence, when computing *steady* state flows, where the common approach is to solve the *unsteady* problem until a *steady* solution is reached, algorithms such as SIMPLE and PISO are preferred [32].

B.7.1.3. The Algorithm

The SIMPLE algorithm holds some resemblance to the method from the previous section. To observe this, our starting point is the semi-discretized momentum equations about the CV nodal point P, with an implicit time advance scheme [32]:

$$a_P^{u_i} u_{i,P}^{n+1} + \sum_l a_l^{u_i} u_{i,l}^{n+1} = q_{u_i}^{n+1} - \left(\frac{\delta p^{n+1}}{\delta x_i} \right)_P \tag{B.64}$$

where the sum with the index l is about the neighbor nodal points, where the source term $q_{u_i}^{n+1}$ incorporates all terms that contain the velocities u_i^n and, finally, where other linearized terms that may depend on u_i^{n+1}. The coefficients $a_l^{u_i}$ also depend on the solution u_i^{n+1}. Again, the term with $\delta/\delta x$ represents a discretization choice for the pressure gradient at the new time step. The coefficients a and possibly the source q depend on u_i^{n+1} due to the non-linearity of the momentum equations, see, for instance, Eq. B.58. Moreover, notice the similarity between Eq. B.64 and Eq. B.32. The exact values of the coefficients and source term depend on the discretization and interpolation choices, but can always be written in this manner [32].

The *iterative* procedure used for solving Eq. B.64 is a *sequential* method. Coefficients are considered fixed for *inner* iterations, while being updated for *outer* iterations, see Sec. B.6.7. The equations solved on *outer* iterations are as follows:

$$a_P^{u_i} u_{i,P}^{m*} + \sum_l a_l^{u_i} u_{i,l}^{m*} = q_{u_i}^{m-1} - \left(\frac{\delta p^{m-1}}{\delta x_i} \right)_P \tag{B.65}$$

where we replaced the index $n+1$ with m and where u_i^{m*} is an estimate for u_i^m which still does not satisfy the mass conservation. Again, this somewhat resembles the situation from Sec. B.7.1.2, see Eq. B.61. Next, we find a Poisson equation for the pressure p^m and then enforce the mass conservation on the velocities by subtracting the pressure gradient term in a fashion similar to the previous subsection, see Sec. B.7.1.2. This time, however, after the gradient has been subtracted from the estimated velocity, the momentum conservation is no longer satisfied [32]. We then proceed to the next *outer* iteration, until the pressure and velocity fields satisfy both the momentum and mass conservation equations. It is important to stress that when computing *unsteady flows*, the fields must satisfy the conservation equations to a narrow tolerance in a given time step [32]. For *steady* flows, the tolerance can be more generous [32]. In the SIMPLE algorithm, after determining provisional values of the velocity field from Eq. B.65, instead of proceeding with a Poisson equation for the actual pressure, correction terms are used alternatively, as given below [32, 33]:

$$u_i^m = u_i^{m*} + u'$$

$$p^m = p^{m-1} + p'$$

where u' and p' are the corrections that must be added to the fields u_i^{m*} and p^{m-1} so that they satisfy the mass conservation constraint. Replacing these back into equation Eq. B.65, yields the following:

$$u'_{i,P} = \tilde{u}'_{i,P} - \frac{1}{a_P^{u_i}} \left(\frac{\delta p'}{\delta x_i} \right)_P \tag{B.66}$$

where $\tilde{u}'_i$ is given by:

$$\tilde{u}'_{i,P} = -\frac{\sum_l a_l^{u_i} u'_{i,l}}{a_P^{u_i}} \tag{B.67}$$

The Poisson equation for the pressure-correction derived from Eq. B.66 and the continuity equation applied to the velocity field u_i^m is [32]:

$$\frac{\delta}{\delta x_i} \left[\frac{1}{a_P^{u_i}} \left(\frac{\delta p'}{\delta x_i} \right) \right]_P = \left[\frac{\delta \left(u_i^{m*} \right)}{\delta x_i} \right]_P + \left[\frac{\delta \left(\tilde{u}'_i \right)}{\delta x_i} \right]_P \tag{B.68}$$

In the SIMPLE algorithm, the $\tilde{u}'_i$ part of the velocity corrections are neglected since they are unknown. That is the **main approximation** of the algorithm and probably the main reason why the solution is not even faster to converge [32,33]. The solving procedure is similar to the one described in the beginning of this section, which uses a sequential approach. The fields then alternate satisfying either the momentum or mass conservation constraint until both are satisfied to narrow range [32]. Finally, one proceeds to the next time step. In OpenFOAM, the SIMPLE algorithm is implemented to solve *steady* problems, employing under-relaxation to ensure stability and a faster convergence [89, 96]. Therefore, the time steps, which are, in fact, the *outer* iterations, do not correspond to actual time lengths, as under-relaxation makes the solution converge faster, by losing the time consistency in the process [96]. All that matters is the solution at convergence, which is when it reaches the *steady* state, see Sec. B.6.7. When running a solver which uses the SIMPLE algorithm in OpenFOAM, such as simpleFOAM, the time step size is a dummy parameter: the important aspect is the number of iterations [96]. A diagram illustrating the basic steps of the algorithm can be found in Versteeg Chapter 6 [33].

B.7.2. PISO algorithm

The anacronym PISO stands for Pressure Implicit Split-Operator. The PISO algorithm serves the same purpose the SIMPLE algorithm does, that is, it solves the momentum and continuity equations [131]. In OpenFOAM, it is implemented as an algorithm for *transient* problems [89, 96]. Similarly, it is used by many of the standard OpenFOAM *solvers.* Parameter changes can be made in the PISO *subdictionary* under the *fvSolution* dictionary of the case, see Fig. B.2.

The PISO algorithm follows the exact same steps the SIMPLE algorithm does to solve the momentum equations Eq. B.65 sequentially and then, just as in SIMPLE, a correction step is performed neglecting the $\tilde{u}_i'$ part of the velocity correction in equations Eq. B.66 and Eq. B.68. At this point, in the PISO method, before carrying on to the next *outer* iteration, additional correction steps are performed [32, 33]. In OpenFOAM, the *outer* iterations for the PISO algorithm are the time steps [96]. Hence, in a fashion similar to Eq. B.66, the second correction to the velocity u'' is given by:

$$u''_{i,P} = \tilde{u}'_{i,P} - \frac{1}{a_P^{u_i}} \left(\frac{\delta p''}{\delta x_i} \right)_P \tag{B.69}$$

where $\tilde{u}'_{i,P}$ is calculated using the Eq. B.67 and where second pressure correction p'' is determined by the following Poisson equation:

$$\frac{\delta}{\delta x_i} \left[\frac{1}{a_P^{u_i}} \left(\frac{\delta p''}{\delta x_i} \right) \right]_P = \left[\frac{\delta \left(\tilde{u}_i' \right)}{\delta x_i} \right]_P \tag{B.70}$$

More correction steps can be employed in a manner analagous to the one showcased in equations Eq. B.69 and Eq. B.70 [32]. In OpenFOAM, typically the number of correction steps chosen is not greater than 4 [96]. This number can be set in the PISO subdictionary under the *nCorrectors* keyword [89, 96]. In case the number chosen is 1, the implemented PISO algorithm would in theory work in a manner similar to the SIMPLE algorithm described in the previous section. In practice, it does not, since the SIMPLE algorithm is implemented to solve steady-state problems and to make use of under-relaxation factors, while PISO is designed for transient problems, and makes no use of explicit under-relaxation factors. An additional requirement of PISO is that the *Courant* number should be less than one for the solution to be stable [96]. A diagram outlining the main steps of the PISO algorithm can be found in Versteeg Chapter 6 [33].

Finally, in OpenFOAM, there is an additional type of correction that accounts for the *non-orthogonality* of the meshes [89]. A detailed discussion on how this is performed is out of the scope of this text. The number of non-orthogonal correction steps can be specified under the *nNonOrthogonalCorrectors* under the SIMPLE, PISO or PIMPLE subdictionary, which can be found, in turn, in the *fvSolution* dictionary of the case, see Fig. B.2.

B.7.3. Merged PISO-SIMPLE - the PIMPLE algorithm

In OpenFOAM, the PIMPLE algorithm merges aspects from both the implemented SIMPLE and PISO algorithms, yielding a robust method to solve *transient* problems quickly by employing large time steps [89, 96]. The feature which the PIMPLE algorithm shares with the SIMPLE algorithm implemented in OpenFOAM is that

both of them make use of under-relaxation to reach convergence faster [96]. *Outer* iterations in PIMPLE do not correspond to time steps, as in the SIMPLE algorithm. In PIMPLE, *outer* iterations occur *within* the time steps and can be set next to the *nOuterCorrectors* keyword in the PIMPLE subdictionary in the $fvSolution$ file of the case, see Fig. B.2. In PIMPLE, as in SIMPLE, the solution is always under-relaxed when proceeding from an *outer* iteration to next, see Sec. B.6.7, except for the last *outer* iteration within a time step [96]. This ensures time consistency between time steps, making PIMPLE suitable for *transient* problems [96]. Aside from this aspects, the PIMPLE resembles the PISO algorithm at everything else. For instance, additional pressure-correction steps can be set with *nCorrectors* keyword and both are suitable for solving transient flow problems. In fact, if one sets *nOuterCorrectors* to 1 without including under-relaxation, the PIMPLE algorithm operates in PISO mode [96].

In many situations, a great advantage of employing solver applications which make use of the PIMPLE algorithm, e.g., pimpleFOAM, is that PIMPLE can handle *Courant* numbers of magnitude greater than one, see Sec. B.6.5 [96]. For instance, PIMPLE greatly outperfoms PISO when complex geometries are used, for in those cases the stability condition for PISO ($Co < 1$) would require a very small time step to be used, considerably increasing the computational effort necessary to solve the problem [96]. In that case, if one uses a large time step with PISO regarless, the solution becomes unstable and may diverge [96]. Moreover, the solver will often crash. Meanwhile, the PIMPLE algorithm allows for much larger time steps to be used without any stability issues, tremendously reducing the computational cost [96]. That can be accomplished by setting a high number of outer iterations ranging from 50 to 200. Between 1 to 4 pressure-correction steps are also set in conjunction with the appropriate convergence criteria in the $fvSolution$ dictionary. The computational cost reduction is the main reason why the PIMPLE algorithm was used to solve the flow problems in this work [96].

The other reason for employing PIMPLE is that PIMPLE-based solvers, such as pimpleFOAM, allow for the inclusion of porous regions by adding a source term to the momentum equations, which is, in fact, the Darcy term [129]. As discussed in Sec. B.4, that is done by including a porosity subdictionary defining the properties of the porous medium in the $fvOptions$ file under the *system* folder, see Fig. B.2.

Bibliography

[1] M. Bernot, V. Caselles, and J.M. Morel. *Optimal Transportation Networks: Models and Theory*. Number no. 1955 in Lecture Notes in Mathematics. Springer, 2009.

[2] Eleni Katifori, Gergely J. Szöllősi, and Marcelo O. Magnasco. Damage and fluctuations induce loops in optimal transport networks. *Phys. Rev. Lett.*, 104:048704, Jan 2010.

[3] Oliver Tamm, Christian Hermsmeyer, and Allen M Rush. Eco-sustainable system and network architectures for future transport networks. *Bell Labs Technical Journal*, 14(4):311–327, 2010.

[4] Robertus Van Nes. *Design of multimodal transport networks: A hierarchical approach*. TU Delft, Delft University of Technology, 2002.

[5] Kenneth J Button. *Economics of transport networks. In: Handbook of transport systems and traffic control*. 2001.

[6] Raul P Lejano. Optimizing the layout and design of branched pipeline water distribution systems. *Irrigation and Drainage Systems*, 20(1):125–137, 2006.

[7] Antonio Heitor Reis, Antonio Ferreira Miguel, and M Aydin. Constructal theory of flow architecture of the lungs. *Medical physics*, 31(5):1135–1140, 2004.

[8] Geoffrey B West, James H Brown, and Brian J Enquist. A general model for the origin of allometric scaling laws in biology. *Science*, 276(5309):122–126, 1997.

[9] JS Andrade Jr, AM Alencar, MP Almeida, J Mendes Filho, SV Buldyrev, S Zapperi, HE Stanley, and B Suki. Asymmetric flow in symmetric branched structures. *Physical review letters*, 81(4):926, 1998.

[10] Adriano M Alencar, Zoltán Hantos, Ferenc Peták, József Tolnai, Tibor Asztalos, Stefano Zapperi, José S Andrade Jr, Sergey V Buldyrev, H Eugene Stanley, and Béla Suki. Scaling behavior in crackle sound during lung inflation. *Physical Review E*, 60(4):4659, 1999.

[11] Adriano M Alencar, Sergey V Buldyrev, Arnab Majumdar, H Eugene Stanley, and Béla Suki. Avalanche dynamics of crackle sound in the lung. *Physical review letters*, 87(8):088101, 2001.

[12] Hiroko Kitaoka, Ryuji Takaki, and Béla Suki. A three-dimensional model of the human airway tree. *Journal of Applied Physiology*, 87(6):2207–2217, 1999.

[13] Adrian Bejan. Constructal tree network for fluid flow between a finite-size volume and one source or sink. *Revue generale de thermique*, 36(8):592–604, 1997.

[14] Daniel P Bebber, Juliet Hynes, Peter R Darrah, Lynne Boddy, and Mark D Fricker. Biological solutions to transport network design. *Proceedings of the Royal Society of London B: Biological Sciences*, 274(1623):2307–2315, 2007.

[15] Robert W Barber and David R Emerson. Optimal design of microfluidic networks using biologically inspired principles. *Microfluidics and Nanofluidics*, 4(3):179–191, 2008.

[16] Willard I Zangwill. Minimum concave cost flows in certain networks. *Management Science*, 14(7):429–450, 1968.

[17] Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8(3):399–404, 1956.

[18] Nannan Guo, Ming C. Leu, and Umit O. Koylu. Bio-inspired flow field designs for polymer electrolyte membrane fuel cells. *International Journal of Hydrogen Energy*, 39(36):21185 – 21195, 2014.

[19] Anita Roth-Nebelsick, Dieter Uhl, Volker Mosbrugger, and Hans Kerp. Evolution and function of leaf venation architecture: a review. *Annals of Botany*, 87(5):553–566, 2001.

[20] AA Jeje. The flow and dispersion of water in the vascular network of dicotyledonous leaves. *Biorheology*, 22(4):285–302, 1984.

[21] Park S Nobel. *Physicochemical and environmental plant physiology*. Academic press, 1999.

[22] K. Kalyanasundaram. *Dye-sensitized Solar Cells*. Fundamental Sciences: Chemistry. EFPL Press, 2010.

[23] Janne Halme. Dye-sensitized nanostructured and organic photovoltaic cells: technical review and preliminary tests. Master's thesis, Helsinki University of Technology, 2002.

[24] Hyung-Jun Koo and Orlin D. Velev. Regenerable Photovoltaic Devices with a Hydrogel-Embedded Microvascular Network. *SCIENTIFIC REPORTS*, 3, AUG 5 2013.

[25] F. Barbir. *PEM Fuel Cells: Theory and Practice*. Energy-Engineering. Academic Press, 2013.

[26] Ulrich Eberle, Bernd Müller, and Rittmar von Helmolt. Fuel cell electric vehicles and hydrogen infrastructure: status 2012. *Energy & Environmental Science*, 5(10):8780–8798, 2012.

[27] Jason P. Kloess, Xia Wang, Joan Liu, Zhongying Shi, and Laila Guessous. Investigation of bio-inspired flow channel designs for bipolar plates in proton exchange membrane fuel cells. *Journal of Power Sources*, 188(1):132 – 140, 2009.

[28] BH Lim, EH Majlan, WRW Daud, Teuku Husaini, and Masli Irwan Rosli. Effects of flow field design on water management and reactant distribution in pemfc: a review. *Ionics*, 22(3):301–316, 2016.

[29] K Tüber, A Oedegaard, M Hermann, and C Hebling. Investigation of fractal flow-fields in portable proton exchange membrane and direct methanol fuel cells. *Journal of Power Sources*, 131(1):175–181, 2004.

[30] George B Arfken and Hans J Weber. *Mathematical methods for physicists international student edition*. Academic press, 2005.

[31] T. Cebeci, J.P. Shao, F. Kafyeke, and E. Laurendeau. *Computational Fluid Dynamics for Engineers: From Panel to Navier-Stokes Methods with Computer Programs*. Springer Berlin Heidelberg, 2005.

[32] J.H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2001.

[33] H.K. Versteeg and W. Malalasekera. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*. Pearson Education Limited, 2007.

[34] CAJ Fletchet. Computational techniques for fluid dynamics; vol 1. 1991.

[35] D.W. Peaceman. *Fundamentals of Numerical Reservoir Simulation*. Developments in Petroleum Science. Elsevier Science, 2000.

[36] B.J. Winer, D.R. Brown, and K.M. Michels. *Statistical Principles in Experimental Design*. McGraw-Hill series in psychology. McGraw-Hill, 1991.

[37] R.L. Mason, R.F. Gunst, and J.L. Hess. *Statistical Design and Analysis of Experiments: With Applications to Engineering and Science*. Wiley Series in Probability and Statistics. Wiley, 2003.

[38] J. Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Elsevier Science, 2013.

[39] Robert McGill, John W Tukey, and Wayne A Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.

[40] Bernard L Welch. The generalization ofstudent's' problem when several different population variances are involved. *Biometrika*, 34(1/2):28–35, 1947.

[41] M.J. Crawley. *Statistics: An Introduction Using R*. Wiley, 2014.

[42] A Chapman and I Mellor. Development of biomimetic flow field plates for pem fuel cells. In *Eighth grove fuel cell symposium*, 2003.

[43] Xianguo Li and Imran Sabir. Review of bipolar plates in pem fuel cells: Flow-field designs. *International Journal of Hydrogen Energy*, 30(4):359–371, 2005.

[44] Adam Runions, Martin Fuhrer, Brendan Lane, Pavol Federl, Anne-Gaëlle Rolland-Lagan, and Przemyslaw Prusinkiewicz. Modeling and visualization of leaf venation patterns. *ACM Trans. Graph.*, 24(3):702–711, July 2005.

[45] R Melville. The terminology of leaf architecture. *Taxon*, pages 549–561, 1976.

[46] Leo J Hickey. Classification of the architecture of dicotyledonous leaves. *American journal of botany*, pages 17–33, 1973.

[47] Irving W Bailey and Edmund W Sinnott. The climatic distribution of certain types of angiosperm leaves. *American journal of botany*, pages 24–39, 1916.

[48] Lawren Sack and Christine Scoffoni. Leaf venation: structure, function, development, evolution, ecology and applications in the past, present and future. *New Phytologist*, 198(4):983–1000, 2013.

[49] Katherine Esau. Anatomy of seed plants. 1977.

[50] Lawren Sack, Christine Scoffoni, Athena D McKown, Kristen Frole, Michael Rawls, J Christopher Havran, Huy Tran, and Thusuong Tran. Developmentally based scaling of leaf venation architecture explains global ecological patterns. *Nature Communications*, 3:837, 2012.

[51] Adam Runions, Richard S Smith, and Przemyslaw Prusinkiewicz. Computational models of auxin-driven development. In *Auxin and its role in plant development*, pages 315–357. Springer, 2014.

[52] Norman MacDonald. Trees and networks in biological models. 1983.

[53] Gilbert Strang and LB Freund. Introduction to applied mathematics. *Journal of Applied Mechanics*, 53:480, 1986.

[54] Robert Sedgewick and Kevin Wayne. Algorithms and data structures. *Princeton University, COS*, 226, 2007.

[55] P.S. Addison. *Fractals and Chaos: An illustrated course*. Taylor & Francis, 1997.

[56] Benoıt B Mandelbrot. How long is the coast of britain. *Science*, 156(3775):636–638, 1967.

[57] Jens Feder. *Fractals*. Springer Science & Business Media, 2013.

[58] Astrid Herbig and Ulrich Kull. Leaves and ramification. 1992.

[59] V Mosbrugger. Constructional morphology as a useful approach in fossil plant biology. *Cour. Forsch.-Inst. Senckenberg*, 147:19–29, 1992.

[60] Odemir Martinez Bruno, Rodrigo de Oliveira Plotze, Mauricio Falvo, and Mário de Castro. Fractal dimension applied to plant identification. *Information Sciences*, 178(12):2722–2733, 2008.

[61] Tsvi Sachs. The control of the patterned differentiation of vascular tissues. volume 9 of *Advances in Botanical Research*, pages 151 – 262. Academic Press, 1981.

[62] Tsvi Sachs. Collective specification of cellular development. *BioEssays*, 25(9):897–903, 2003.

[63] Sang-Woo Lee, Francois Gabriel Feugier, and Yoshihiro Morishita. Canalization-based vein formation in a growing leaf. *Journal of theoretical biology*, 353:104–120, 2014.

[64] René Benjamins and Ben Scheres. Auxin: the looping star in plant development. *Annu. Rev. Plant Biol.*, 59:443–465, 2008.

[65] Anne-Gaëlle Rolland-Lagan and Przemyslaw Prusinkiewicz. Reviewing models of auxin canalization in the context of leaf vein pattern formation in arabidopsis. *The Plant Journal*, 44(5):854–865, 2005.

[66] Enrico Scarpella, Danielle Marcos, Jiří Friml, and Thomas Berleth. Control of leaf vascular patterning by polar auxin transport. *Genes & development*, 20(8):1015–1027, 2006.

[67] Roni Aloni, Katja Schwalm, Markus Langhans, and Cornelia I Ullrich. Gradual shifts in sites of free-auxin production during leaf-primordium development and their role in vascular differentiation and leaf morphogenesis in arabidopsis. *Planta*, 216(5):841–853, 2003.

[68] Enrico Scarpella, Michalis Barkoulas, and Miltos Tsiantis. Control of leaf and vein development by auxin. *Cold Spring Harbor Perspectives in Biology*, 2(1):a001511, 2010.

[69] Katherine Esau et al. Plant anatomy. *Plant anatomy*, (2nd Edition), 1965.

[70] Michael LaBarbera. Principles of design of fluid transport systems in zoology. *Science*, 249(4972):992–1000, 1990.

[71] Charles A Price, Scott Wing, and Joshua S Weitz. Scaling and structure of dicotyledonous leaf venation networks. *Ecology letters*, 15(2):87–95, 2012.

[72] Charles A Price, Sarah-Jane C Knox, and Tim J Brodribb. The influence of branch order on optimal leaf vein geometries: Murray's law and area preserving branching. *PloS one*, 8(12):e85420, 2013.

[73] Cecil D. Murray. The physiological principle of minimum work: I. the vascular system and the cost of blood volume. *Proceedings of the National Academy of Sciences*, 12(3):207–214, 1926.

[74] GJ Mitchison. A model for vein formation in higher plants. *Proceedings of the Royal Society of London B: Biological Sciences*, 207(1166):79–109, 1980.

[75] Marc E Gottlieb. Angiogenesis and vascular networks: complex anatomies from deterministic non-linear physiologies. In *Growth patterns in physical sciences and biology*, pages 267–276. Springer, 1993.

[76] Kenneth H Rosen and Kamala Krithivasan. *Discrete mathematics and its applications*, volume 6. McGraw-Hill New York, 1995.

[77] Adam Drozdek. *Data Structures and algorithms in C++*. Cengage Learning, 2012.

[78] David Dobkin and Richard J Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5(2):181–186, 1976.

[79] Steven Fortune. A sweepline algorithm for voronoi diagrams. *Algorithmica*, 2(1-4):153–174, 1987.

[80] Godfried T Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

[81] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[82] Franco P Preparata and Michael Shamos. *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

[83] RB Urquhart. Algorithms for computation of relative neighbourhood graph. *Electronics Letters*, 14(16):556–557, 1980.

[84] Diogo Vieira Andrade and Luiz Henrique de Figueiredo. Good approximations for the relative neighbourhood graph. In *CCCG*, pages 25–28, 2001.

[85] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[86] Mason Woo, Jackie Neider, Tom Davis, Dave Shreiner, et al. Opengl programming guide, 1999.

[87] Christophe Geuzaine. Gl2ps: an opengl to postscript printing library, 2006.

[88] Wikibooks. Openscad user manual/using openscad in a command line environment — wikibooks, the free textbook project, 2016. [Online; accessed 5-December-2016].

[89] CJ Greenshields. Openfoam user guide, 2015.

[90] H. Lomax, T.H. Pulliam, and D.W. Zingg. *Fundamentals of Computational Fluid Dynamics*. Scientific Computation. Springer Berlin Heidelberg, 2013.

[91] C. Pozrikidis. *Fluid Dynamics: Theory, Computation, and Numerical Simulation*. Springer US, 2009.

[92] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia. Meshlab: an open-source 3d mesh processing system. *Ercim news*, 73(45-46):6, 2008.

[93] Ryan Schmidt and Nobuyuki Umetani. Branching support structures for 3d printing. In *ACM SIGGRAPH 2014 Studio*, page 9. ACM, 2014.

[94] Kudo3D Titan. Printing guide. 2014.

[95] C Peralta, H Nugusse, SP Kokilavani, J Schmidt, and B Stoevesandt. Validation of the simplefoam (rans) solver for the atmospheric boundary layer in complex terrain. In *ITM Web of Conferences*, volume 2. EDP Sciences, 2014.

[96] Tobias Holzmann. *Mathematics, Numerics, Derivations and OpenFOAMÂ®*. Holzmann CFD, 2016.

[97] Jan Graffelman. Linear-angle correlation plots: new graphs for revealing correlation structure. *Journal of Computational and Graphical Statistics*, 22(1):92–106, 2013.

[98] Tyler Finnes. High definition 3d printing–comparing sla and fdm printing technologies. *The Journal of Undergraduate Research*, 13(1):3, 2015.

[99] Glyn O Phillips and Peter A Williams. *Handbook of hydrocolloids*. Elsevier, 2009.

[100] Janaky Narayanan, Jun-Ying Xiong, and Xiang-Yang Liu. Determination of agarose gel pore size: Absorbance measurements vis a vis other techniques. In *Journal of Physics: Conference Series*, volume 28, page 83. IOP Publishing, 2006.

[101] Mark Robert. Kemp. *The enhancement of mass transfer in foods by alternating electric fields*. PhD thesis, University of Birmingham, 2000.

[102] K. Samprovalaki, P.T. Robbins, and P.J. Fryer. Investigation of the diffusion of dyes in agar gels. *Journal of Food Engineering*, 111(4):537 – 545, 2012.

[103] Annemarie Nadort, Koen Kalkman, Ton G van Leeuwen, and Dirk J Faber. Quantitative blood flow velocity imaging using laser speckle flowmetry. *Scientific reports*, 6, 2016.

[104] Jungwoo Lee, Meghan J Cuddihy, and Nicholas A Kotov. Three-dimensional cell culture matrices: state of the art. *Tissue Engineering Part B: Reviews*, 14(1):61–86, 2008.

[105] Francesco Pampaloni, Emmanuel G Reynaud, and Ernst HK Stelzer. The third dimension bridges the gap between cell culture and live tissue. *Nature reviews Molecular cell biology*, 8(10):839–845, 2007.

[106] Luiz E Bertassoni, Martina Cecconi, Vijayan Manoharan, Mehdi Nikkhah, Jesper Hjortnaes, Ana Luiza Cristino, Giada Barabaschi, Danilo Demarchi, Mehmet R Dokmeci, Yunzhi Yang, et al. Hydrogel bioprinted microchannel networks for vascularization of tissue engineering constructs. *Lab on a Chip*, 14(13):2202–2211, 2014.

[107] Nitish Peela, Feba S Sam, Wayne Christenson, Danh Truong, Adam W Watson, Ghassan Mouneimne, Robert Ros, and Mehdi Nikkhah. A three dimensional micropatterned tumor model for breast cancer cell migration studies. *Biomaterials*, 81:72–83, 2016.

[108] Der-Tsai Lee. Two-dimensional voronoi diagrams in the l p-metric. *Journal of the ACM (JACM)*, 27(4):604–618, 1980.

[109] Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*, volume 501. John Wiley & Sons, 2009.

[110] MacGayver da S Castro. Efeito mecÃ¢nico da exposiÃ§Ã£o aguda de componentes encontrados na poluiÃ§Ã£o atmosfÃ©rica em amostras de mÃºsculo liso. Master's thesis, Universidade de SÃ£o Paulo, 2012.

[111] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[112] Nicholas Nethercote and Julian Seward. Valgrind: A program supervision framework. *Electronic notes in theoretical computer science*, 89(2):44–66, 2003.

[113] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[114] Richard J Trudeau. *Introduction to graph theory*. Courier Corporation, 2013.

[115] Raimund Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Computational Geometry*, 1(1):51–64, 1991.

[116] Cyprien Soulaine. On the origin of darcy's law.

[117] K. Vafai. *Handbook of Porous Media, Third Edition*. CRC Press, 2015.

[118] Hrvoje Jasak, Aleksandar Jemcov, Zeljko Tukovic, et al. Openfoam: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20. IUC Dubrovnik, Croatia, 2007.

[119] Hrvoje Jasak. Openfoam: open source cfd in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, 1(2):89–94, 2009.

[120] Martin Beaudoin and Hrvoje Jasak. Development of a generalized grid interface for turbomachinery simulations with openfoam. In *Open source CFD International conference*, volume 2, 2008.

[121] Pablo Higuera, Javier L Lara, and Inigo J Losada. Realistic wave generation and active wave absorption for navier–stokes models: Application to openfoam®. *Coastal Engineering*, 71:102–118, 2013.

[122] TÃ¢nia TomÃ© and MÃ¡rio J de Oliveira. *DinÃ¢mica estocÃ¡stica e irreversibilidade*. Edusp, 2014.

[123] Songmiao Liang, Jian Xu, Lihui Weng, Hongjun Dai, Xiaoli Zhang, and Lina Zhang. Protein diffusion in agarose hydrogel in situ measured by improved refractive index method. *Journal of controlled release*, 115(2):189–196, 2006.

[124] Veerle Cnudde and Matthieu Nicolaas Boone. High-resolution x-ray computed tomography in geosciences: A review of the current technology and applications. *Earth-Science Reviews*, 123:1–17, 2013.

[125] HC Brinkman. A calculation of the viscous force exerted by a flowing fluid on a dense swarm of particles. *Applied Scientific Research*, 1(1):27–34, 1949.

[126] R Allan Freeze. Henry darcy and the fountains of dijon. *Ground Water*, 32(1):23–30, 1994.

[127] C Soulaine. Direct numerical simulation in fully saturated porous media.

[128] Pierre Adler. *Porous media: geometry and transports*. Elsevier, 2013.

[129] Haukur Elvar Hafsteinsson. Porous media in openfoam. *Chalmers University of Technology, Gothenburg*, 2009.

[130] Suhas V Patankar and D Brian Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International journal of heat and mass transfer*, 15(10):1787–1806, 1972.

[131] R. I. Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of Computational Physics*, 62:40–65, January 1986.