ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta jaderná a fyzikálně inženýrská

Optimální tvar stěn idealizovaného kavopulmonálního spojení

Optimal wall geometry of an idealized total cavopulmonary connection

Master's thesis

Author:	**Bc. Jan Bureš**
Supervisor:	**doc. Ing. Radek Fučík, Ph.D.**
Consultant:	**MUDr. Mgr. Radomír Chabiniok, Ph.D.**
Academic year:	2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Bureš**	Jméno: **Jan**	Osobní číslo: **494688**

Fakulta/ústav: **Fakulta jaderná a fyzikálně inženýrská**

Zadávající katedra/ústav: **Katedra matematiky**

Studijní program: **Matematické inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimální tvar stěn idealizovaného úplného kavopulmonálního spojení.

Název diplomové práce anglicky:

Optimal wall geometry of an idealized total cavopulmonary connection.

Pokyny pro vypracování:

1. S použitím dostupné literatury a konzultací s odborníky sestavte matematický model cévního proudění v problematice úplného kavopulmonálního spojení (TCPC) založený na mřížkové Boltzmannově metodě (LBM) a navrhněte vhodně parametrizovanou testovací úlohu, která aproximuje charakteristiky a funkčnost systému TCPC.
2. Formulujte optimalizační úlohu zaměřenou na hledání optimálního tvaru stěn s využitím parametricky popsané geometrie testovací úlohy z bodu 1. Pokuste se navrhnout vhodnou účelovou funkci použitelnou v rámci studované problematiky.
3. Proveďte rešerši optimalizačních metod vhodných pro řešení problémů charakterizovaných zvýšenou časovou náročností potřebnou pro vyhodnocení účelové funkce.
4. Pro simulaci proudění vhodně upravte výpočetní kód LBM vyvíjený na KM FJFI ČVUT v Praze. Navrhněte a implementujte obecný optimalizační rámec, jehož součástí bude výpočetní kód LBM. Využijte volně dostupný software nebo sám implementujte vybrané metody matematické optimalizace z bodu 3.
5. Aplikujte metody matematické optimalizace k hledání optimálního řešení pro uvažovanou optimalizační úlohu z bodu 2. Diskutujte získané výsledky a výpočetní náročnost jejich získání.

Seznam doporučené literatury:

[1] T. Krüger, et al., The lattice Boltzmann method: Principles and Practice. Springer International Publishing, 2017.
[2] Z. Guo, S. Chang, Lattice Boltzmann method and its application in engineering. World Scientific, 2013.
[3] J. D. Anderson, Computational Fluid Dynamics. McGraw-Hill series in mechanical engineering. McGraw-Hill Professional, 1995.
[4] F. M. Rijnberg, et al., Energetics of blood flow in cardiovascular disease. Circulation,137(22), 2018, 2393–2407.
[5] S. Boyd, L. Vandenberghe, Convex optimization. Cambridge University Press, 2004.
[6] C. Audet a W. Hare. Derivative-free and blackbox optimization. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Cham, Switzerland, 1.edice, 2017.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Radek Fučík, Ph.D. katedra matematiky FJFI

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

Mudr. Mgr. Radomír Chabiniok, Ph.D. University of Texas Southwestern Medical Center, USA

Datum zadání diplomové práce: **31.10.2023** Termín odevzdání diplomové práce: **10.05.2024**

Platnost zadání diplomové práce: **31.10.2025**

[illegible]	Masáková	[illegible]
doc. Ing. Radek Fučík, Ph.D.	prof. Ing. Zuzana Masáková, Ph.D.	doc. Ing. Václav Čuba, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

9. 11. 2023	[illegible]
Datum převzetí zadání	Podpis studenta

Acknowledgment:
I would like to express my gratitude to my supervisor, doc. Ing. Radek Fučík, Ph.D., whose guidance, expertise, and support were pivotal throughout this work. I am also deeply thankful to my consultant, MUDr. Mgr. Radomír Chabiniok, Ph.D., for his valuable insights on the medical aspects of this thesis.

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS23/188/OHK4/3T/14.

Author's declaration:
I declare that this thesis is entirely my own work and I have listed all the used sources in the bibliography.

I further declare that the available AI tools were used solely for grammar checking and improving sentence formulations, in full accordance with the guidelines set out by CTU in Prague.

Prague, January 5, 2025 Jan Bureš

[illegible]

Název práce: **Optimální tvar stěn idealizovaného kavopulmonálního spojení**

Autor: Bc. Jan Bureš

Program: Matematické inženýrství

Druh práce: Diplomová práce

Vedoucí práce: doc. Ing. Radek Fučík, Ph.D., Katedra matematiky a katedra softwarového inženýrství, FJFI ČVUT v Praze Trojanova 13, 120 00 Praha

Konzultant: MUDr. Mgr. Radomír Chabiniok, Ph.D., University of Texas Southwestern Medical Center, USA

Abstrakt: Tato práce se zabývá optimalizací tvaru idealizovaného modelu úplného kavopulmonálního spojení (TCPC)–chirurgického zákroku prováděného u pacientů s vrozenou srdeční vadou zahrnující funkčně jedinou komoru. Byl vyvinut optimalizační rámec, který zahrnuje generování geometrie v jazyce Python, simulaci pomocí mřížkové Boltzmannovy metody (LBM) a bezgradientní optimalizační algoritmy – Nelderovu-Meadovu metodu a sít'ové adaptivní přímé vyhledávání. Vyvinutý rámec automatizuje proces generování parametrizovaných 3D geometrií, simulace proudění nestlačitelné newtonovské tekutiny a vyhodnocování účelových funkcí. Navržený rámec byl testován na zjednodušených modelech TCPC s různými optimalizačními parametry. Výsledky prokázaly funkčnost a účinnost navrženého přístupu. Přestože se studie zaměřuje pouze na idealizované geometrie se zjednodušenými předpoklady, jako je rigidita stěn cév, newtonovské proudění a ustálené proudění, poskytuje základ pro rozšíření na data specifická pro pacienty a pro zahrnutí složitějších fyziologických podmínek. Tato práce představuje krok vpřed v aplikaci optimalizace v kardiochirurgii s potenciálem zlepšit klinické výsledky a plánování léčby specifické pro jednotivé pacienty.

Klíčová slova: bezgradientní optimalizace, modelování proudění krve, mřížková Boltzmannova metoda, optimalizace tvarů, úplné kavopulmonární spojení

Title: **Optimal wall geometry of an idealized total cavopulmonary connection**

Author: Bc. Jan Bureš

Abstract: This thesis addresses the optimization of wall geometry for an idealized total cavopulmonary connection (TCPC), a surgical procedure used to treat congenital heart defects involving a single functional ventricle. A custom optimization framework was developed, integrating Python-based geometry generation, simulation using the lattice Boltzmann method (LBM), and gradient-free optimization algorithms–Nelder-Mead and Mesh adaptive direct search methods. The framework automates the process of generating parameterized 3D geometries, simulating flow of incompressible Newtonian fluid, and evaluating objective functions. The proposed framework was tested on simplified TCPC models with varying optimization parameters. Results demonstrated the feasibility and effectiveness of the approach. While the study focuses on idealized geometries with simplified assumptions, such as rigid vessel walls, Newtonian flow, and steady flow, it provides a robust foundation for extending the framework to patient-specific data and more complex physiological conditions. This work represents a step forward in applying computational optimization to cardiovascular surgery, with the potential to enhance clinical outcomes and patient-specific treatment planning.

Keywords: gradient-free optimization, lattice Boltzmann method, modeling of blood flow, shape optimization, total cavopulmonary connection

Contents

Introduction

This work focuses on the mathematical modeling of fluid flow, with an emphasis on optimizing the shape of walls of the idealized total cavopulmonary connection (TCPC). In many engineering fields such as the automotive or aerospace industries, incorporating optimization processes to find the optimal shape of the studied object is a standard practice. In clinical medicine, however, the use of optimization techniques is less common due to various challenges. Accurate modeling of blood flow, often through complex geometries, requires careful validation against experimental data, which can be difficult to obtain – *in vivo*[1] experiments are naturally often infeasible or ethically constrained, complicating the process of validation.

Nevertheless, the process of shape optimization can find a great potential especially in cardiac and vascular surgery [1, 2, 3, 4, 5]. Developing a systematic optimization framework for medical applications would provide clinicians with an *in silico*[2] tool to assess intervential procedures within patient-specific geometries. Designing and implanting objects such as stents or artificial valves could then be directly tailored to the patient's anatomy. This approach could lead to improved clinical outcomes, reduced risk of postoperative complications, and a general improvement in the patient's quality of life [3, 6].

An example of a specific surgical procedure, where the process of wall shape optimization can find its application, is the total cavopulmonary connection. TCPC is performed on children diagnosed with a congenital heart defect reffered to as functionally single ventricle, where the heart can effectively utilize only one functional ventricle for the blood circulation [7]. In this procedure, the superior vena cava (*vena cava superior*, labeled *d* in Figure 1) is surgically connected to the pulmonary artery. The inferior vena cava (*vena cava inferior*, labeled *b* in Figure 1) is also connected directly to the pulmonary artery (*arteria pulmonalis*, labeled *a* in Figure 1), typically using an extracardiac conduit (labeled *c* in Figure 1), which is a vascular prosthesis [8, 9].

While TCPC enables functional blood circulation, the resulting circulatory system is highly sensitive to energy losses caused by factors such as turbulent flow and flow collisions. These inefficiencies can gradually lead to system failure. The shape of the extracardiac conduit connection, aiming to minimize energy losses or tissue stress, is therefore well-suited to be a subject of optimization [7, 10, 11]. Numerous studies address the issue of optimal conduit connection, however, many explore and examine only a small number of geometry variations using a "trial-and-error” approach rather than a rigorous optimization process [5, 6, 12, 13].

To create an optimization framework usable in the context of blood flow simulations, it is necessary to use an efficient and reliable numerical method to compute the values of the objective function, which is subject to minimization. In this work, the lattice Boltzmann method (LBM) is used for numerical computations. Note that an in-house implementation of LBM developed at the Department of Mathematics of Faculty of Nuclear Sciences and Physical Engineering, Czech Technical University (FNSPE CTU) in Prague was used and modified according to the needs of this work. The main advantage of LBM is the

[1]Performed in a living organism.

[2]Performed on a computer.

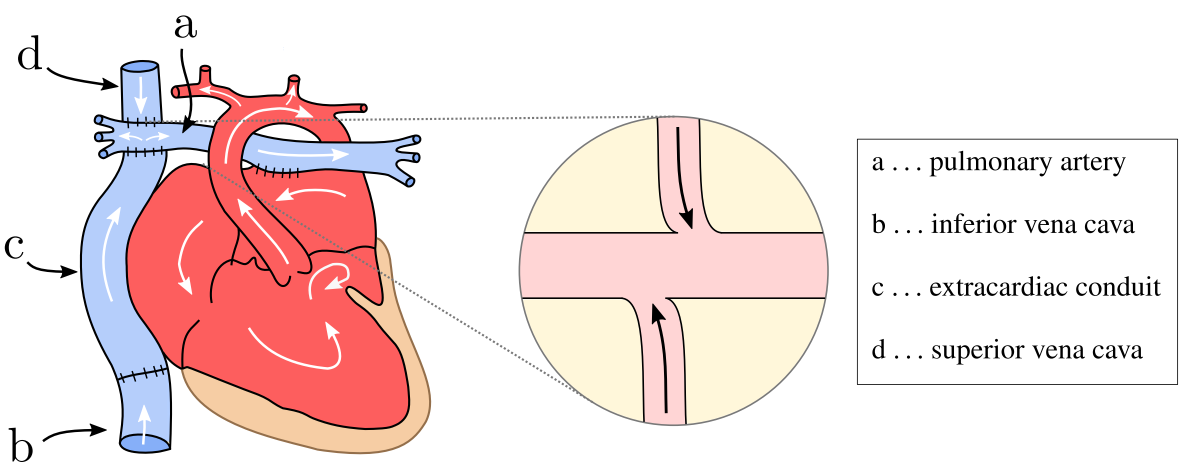

a
d
c
b
a . . . pulmonary artery
b . . . inferior vena cava
c . . . extracardiac conduit
d . . . superior vena cava

Figure 1: Diagram of the total cavopulmonary connection. The enlarged region shows the extracardiac conduit connection.

possibility of massive parallelization on GPUs (graphics processing units), which can potentially significantly reduce the computational time compared to other numerical methods for fluid flow modeling [14, 15] such as the finite difference method [16], the finite volume method [17], or the finite element method [18].

However, LBM also has a notable limitation in the context of geometry representation. The method relies on a regular lattice for spatial discretization, which results in a stair-step approximation of boundaries, which can pose challenges when modeling complex geometries such as blood vessels. To address this challenge, our previous work [19, 20] studied the implementation of interpolated boundary conditions, enabling more accurate representation of boundary shapes within LBM. Furthermore, [20] explored the coupling of both gradient-based and gradient-free optimization methods with LBM. The results from [20] indicate that gradient-free optimization methods are more suitable in the context of this work, making the implementation of interpolated boundary conditions unnecessary for the optimization process to proceed effectively.

The structure of this work is organized as follows. The first chapter introduces the mathematical model of fluid flow, with a focus on modeling of blood flow in vessels. The second chapter discusses the numerical method employed, namely the lattice Boltzmann method. The third chapter details the custom tool developed for parameterization and automatic generation of 3D geometries, tailored to the requirements of both the optimization process and the numerical solver. The fourth chapter details the optimization methods used in this work. Additionally, this chapter introduces the proposed optimization framework used for solving the optimization problems. The final chapter presents the results of the numerical experiments. First, the optimization framework is validated on a simplified test problem involving a cylinder junction with a single optimization parameter, representing an idealized 3D model of TCPC. Then, the optimization is applied to a more complex problem based on an extended 3D idealized model of TCPC, incorporating multiple optimization parameters. Among other aspects, the influence of different optimization methods and objective functions on the performance and outcomes of the optimization framework is investigated.

Chapter 1

Mathematical model

In this chapter, we introduce the governing equations for fluid dynamics, along with an overview of turbulence modeling and Reynolds decomposition. Furthermore, the chapter addresses specific considerations relevant to blood flow in vessels and summarizes the assumptions made in the mathematical model adopted in this work.

1.1 Fluid dynamics

In this work, the fluid is modeled as a continuum. The governing equations for continuum dynamics derive from the fundamental conservation laws of mass, momentum, and energy, which are inherently tied to the physical properties of the fluid and its environment.

Assuming an isolated, isothermal system, the evolution of fluid flow in three-dimensional space is governed by the following system of partial differential equations:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{u}) = 0, \tag{1.1a}$$

$$\frac{\partial (\rho \boldsymbol{u})}{\partial t} + \nabla \cdot (\rho \boldsymbol{u} \otimes \boldsymbol{u}) = \nabla \cdot \mathbf{T} + \rho \boldsymbol{g}, \tag{1.1b}$$

where $\otimes$ denotes the outer product, defined component-wise as $(\boldsymbol{u} \otimes \boldsymbol{u})_{ij} = u_i u_j,\ i, j \in \{1, 2, 3\}$ [21]. The quantities in Eqs. (1.1) are defined as follows:

ρ [kg m^{-3}]	density of the fluid,
$\boldsymbol{u}$ [m s^{-1}]	macroscopic velocity vector,
$\mathbf{T}$ [kg m^{-1} s^{-2}]	(total) stress tensor,
$\boldsymbol{g}$ [m s^{-2}]	external force acceleration vector.

Each of these quantities is generally a function of time t [s] and spatial position $\boldsymbol{x}$ [m].

Assuming an ideal gas in an isothermal system, the equation of state is expressed as:

$$p = c_s^2 \rho, \tag{1.2}$$

where p [Pa] represents the pressure, and c_s [m s^{-1}] is the speed of sound in the fluid [22]. Together, Eqs. (1.1) and (1.2) form a closed system, eliminating the need to explicitly consider the conservation of energy.

1.1.1 Stress tensor

We denote the dynamic stress tensor as $\mathbf{T}_\mu = (\sigma^\mu_{ij})$ [kg m^{-1} s^{-2}], where $i, j \in \{1, 2, 3\}$. For Newtonian fluids, each component of the dynamic stress tensor linearly depends on the spatial velocity derivatives. Fluids that do not satisfy this linear relationship are classified as non-Newtonian.

For Newtonian fluids, the dynamic stress tensor components are given by [23]:

$$\sigma^\mu_{ii} = \lambda \nabla \cdot \boldsymbol{u} + 2\mu \frac{\partial u_i}{\partial x_i}, \quad i \in \{1, 2, 3\}, \tag{1.3a}$$

$$\sigma^\mu_{ij} = \sigma^\mu_{ji} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right), \quad i, j \in \{1, 2, 3\},\ i \neq j, \tag{1.3b}$$

where μ [kg m^{-1} s^{-1}] is the dynamic viscosity, and λ [kg m^{-1} s^{-1}] represents the second viscosity coefficient [24]. For Newtonian fluids, $\mathbf{T}_\mu$ is symmetric.

By introducing the strain rate tensor $\mathbf{D}$ [s^{-1}], defined as

$$\mathbf{D} = \frac{1}{2} \left[\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right], \tag{1.4}$$

the dynamic stress tensor can be equivalently expressed as

$$\mathbf{T}_\mu = 2\mu \mathbf{D} + \left(\lambda + \frac{2}{3}\mu \right) (\nabla \cdot \boldsymbol{u}) \mathbf{I}, \tag{1.5}$$

where $\mathbf{I}$ denotes the identity tensor in three-dimensional space. Stokes' hypothesis, which assumes $\lambda = -\frac{2}{3}\mu$ [21], simplifies the expression to

$$\mathbf{T}_\mu = 2\mu \mathbf{D}. \tag{1.6}$$

The stress tensor for a Newtonian fluid is then given by

$$\mathbf{T} = -p\mathbf{I} + \mathbf{T}_\mu, \tag{1.7}$$

where $\mathbf{I}$ remains the identity tensor in three-dimensional space and p [Pa] represents the pressure [24].

Furthermore, the strain rate tensor is used to the define the shear rate $\dot{\gamma}$ [s^{-1}] [24] given by

$$\dot{\gamma} = \sqrt{2} \|\mathbf{D}\|_F, \tag{1.8}$$

where $\| \cdot \|_F$ denotes the Frobenius norm, defined as

$$\|\mathbf{A}\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}, \tag{1.9}$$

where $\mathbf{A}$ is a matrix of dimensions $m \times n$ with elements a_{ij}.

1.1.2 Turbulence and Reynolds decomposition

Turbulent flow consists of a highly irregular, fluctuating motion [23]. Turbulence typically occurs at high values of the Reynolds number, a dimensionless parameter defined by

$$\mathrm{Re} = \frac{l_0 u_0}{\nu} = \frac{l_0^2}{t_0 \nu}, \tag{1.10}$$

where ν represents the kinematic viscosity [$\mathrm{m^2\,s^{-1}}$], and l_0 [m], t_0 [s], and u_0 [$\mathrm{m\,s^{-1}}$] denote the characteristic length, time, and velocity relevant to the specific flow [25].

Given the highly complex nature of turbulent flow, achieving an exact description is challenging. A common approach is to employ Reynolds time-averaging method, where the instantaneous value of a flow variable ψ is decomposed into its mean value $\overline{\psi}$ and a fluctuating component ψ':

$$\psi = \overline{\psi} + \psi'. \tag{1.11}$$

The fluctuation ψ' is defined such that its time-averaged value over a sufficiently long period is zero [23].

In this decomposition, ψ can represent any scalar quantity, such as velocity components u_i, pressure p, or density ρ [26]. The averaging period should be significantly larger than the time scale of the turbulent fluctuations being studied [26].

When applying the Reynolds decomposition to the velocity field $\boldsymbol{u}(\boldsymbol{x}, t)$ with respect to time averaging, the (specific) turbulent kinetic energy T_{turb} [$\mathrm{m^2\,s^{-2}}$] is defined by

$$T_{\mathrm{turb}} = \frac{1}{2}\left(\overline{(u_1')^2} + \overline{(u_2')^2} + \overline{(u_3')^2}\right) = \frac{1}{2}\overline{\left(u_1'^2 + u_2'^2 + u_3'^2\right)}, \tag{1.12}$$

where we utilize the arithmetic properties of Reynolds decomposition to simplify this expression [26].

1.2 Vascular flow modeling

Due to the presence of numerous physical, chemical, and physiological processes, vascular flow represents a highly complex process. In many cases, however, a simplified model that neglects some characteristics suffices for its description [27]. In this section, we briefly discuss key physiological considerations in blood flow modeling.

1.2.1 Vessel compliance

The interaction between blood flow and elastic vessel walls can be modeled using methods such as the immersed boundary method [28]. In many cases, wall elasticity can be neglected without significantly impacting results. Particularly in smaller vessels, neglecting the elasticity of the vascular walls has not shown a significant effect on the outcome [29]. However, this is not always the case, as significant non-negligible effects on the overall error of the results have been observed in the area of the aorta when considering rigid geometry [30]. Generally, neglecting elasticity in larger vessels can introduce notable errors.

Note that including the elasticity of vascular walls in the considered model requires appropriately defining the fluid-structure interaction, which is generally a difficult task in vascular flow, often depending heavily on accurate *in vivo* measurements. Additionally, in diseases such as arteriosclerosis[1], vessels lose their elasticity, thus including elasticity in the model may not necessarily reflect the physiological state of the vessels correctly [27].

1.2.2 Blood viscosity

Blood is often modeled as a Newtonian fluid. However, there are situations where the Newtonian behavior of blood is compromised. In cases where the flow velocity of blood is very low, red blood cells can aggregate, which leads to increased the viscosity of blood. Areas with such low flow velocities can be

[1] Arteriosclerosis is a disease in which the arterial walls thicken and subsequently lose their elasticity [31].

found, e.g., in regions of aneurysmal dilation[2]. Conversely, the viscosity significantly decreases in areas where blood flows through very narrow vessels, particularly through vessels at the scale of arterioles or capillaries [27].

Multiple viscosity models have been proposed to more accurately reflect the physiological behavior of blood viscosity [27, 33, 34]. Examples include the Power law model [35], the Casson model [34], the Cross model [35], and the Carreau-Yasuda model [34].

1.2.3 Turbulent flow

Blood flow is mostly laminar under normal physiological conditions [35]. However, turbulent flow can occur in specific situations [36]. One notable region where turbulence is frequently observed is behind a vascular stenosis[3] [38]. The flow velocity of blood in the narrowed region can significantly increase. Although the flow typically remains laminar within the narrowed part of the vessel, the increased velocity downstream can lead to the formation of vortices and changes in the flow regime [27, 35, 39].

The occurrence of turbulent flow is physiologically undesirable, as it is often results in significant dissipation of kinetic energy, increasing the strain on the circulatory system. Furthermore, regions with turbulent flow show increased stress on the vessel walls, which can contribute to tissue damage and increased risk of developing conditions such as arteriosclerosis [27, 40].

1.3 Assumptions and simplifications of the model in this work

In this section, we summarize the additional assumptions applied to our system to simplify the governing equations (1.1) and focus on the modeling of the TCPC.

Specifically, the model assumes an isothermal system (i.e., constant temperature over time) with no external forces acting on it. Blood is treated as an incompressible, Newtonian fluid with constant dynamic viscosity. This assumption is justified as the model of TCPC involves relatively large vessels where the non-Newtonian effects of blood, such as viscosity variations, are expected to be negligible.

The vessel walls are assumed to be rigid, ignoring their elasticity. This simplification reduces the model's physiological fidelity. Studies such as [41] and [42] have shown that neglecting compliance in larger vessels can impact the accuracy of results, particularly in terms of power loss and flow distribution. However, correctly modeling fluid-structure interactions to incorporate wall compliance would require detailed *in vivo* data and significantly increase the computational demand.

Additionally, we assume a constant, steady inflow profile, neglecting pulsatile flow typically observed in physiological settings. While pulsatility can influence flow patterns and energy dissipation, its omission simplifies the boundary conditions. This assumption is suited to studying time-averaged metrics, which are often of primary interest in TCPC modeling.

We consider a cuboidal domain $\Omega \subset \mathbb{R}^3$, defined by $\Omega = (0, L_1) \times (0, L_2) \times (0, L_3)$, where L_1, L_2, and L_3 [m] represent the domain dimensions along each spatial axis. The temporal interval is denoted by $\mathcal{I} = [0, T]$, where $T > 0$.

With these assumptions, the governing equations Eqs. (1.1) in $\Omega \times \mathcal{I}$ simplify to [23]:

$$\nabla \cdot \boldsymbol{u} = 0, \tag{1.13a}$$

$$\rho \frac{\mathrm{D}\boldsymbol{u}}{\mathrm{D}t} = -\nabla p + \mu \Delta \boldsymbol{u}, \tag{1.13b}$$

[2]Aneurysmal dilation is a condition characterized by localized enlargement of a blood vessel [32].

[3]Vascular stenosis is a condition characterized by localized narrowing of a blood vessel [37].

where $\frac{\mathrm{D}}{\mathrm{D}t}$ is the material derivative operator

$$\frac{\mathrm{D}}{\mathrm{D}t} := \frac{\partial}{\partial t} + \boldsymbol{u} \cdot \nabla. \tag{1.14}$$

The boundary of Ω is split into parts as follows: $\partial\Omega_{\mathrm{in}}$ denotes the inflow boundary where inlet conditions are specified; $\partial\Omega_{\mathrm{out}}$ represents the outflow boundary; and $\partial\Omega_{\mathrm{w}}$ corresponds to the wall boundaries, where the no-slip condition is imposed. The system of Eqs. (1.13) is supplemented by the following initial and boundary conditions:

$$\begin{aligned}
&\boldsymbol{u} = \boldsymbol{u}_{\mathrm{ini}}, & p &= p_{\mathrm{ini}} && \text{in } \Omega \times \mathcal{I}, && \text{(1.15a)}\\
&(\nabla p - \nu \Delta \boldsymbol{u}) \cdot \boldsymbol{n} = 0, & \boldsymbol{u} &= \boldsymbol{u}_{\mathrm{in}} && \text{in } \partial\Omega_{\mathrm{in}} \times \mathcal{I}, && \text{(1.15b)}\\
&\boldsymbol{u} = \boldsymbol{0}, & \nabla p \cdot \boldsymbol{n} &= 0 && \text{in } \partial\Omega_{\mathrm{w}} \times \mathcal{I}, && \text{(1.15c)}\\
&p = p_{\mathrm{out}}, & \nabla u_i \cdot \boldsymbol{n} &= 0 && \text{in } \partial\Omega_{\mathrm{out}} \times \mathcal{I}, \quad i = 1, 2, 3, && \text{(1.15d)}
\end{aligned}$$

where $\boldsymbol{n}$ is the outward-pointing normal unit vector to the boundary $\partial\Omega$. Here, $\boldsymbol{u}_{\mathrm{ini}}$ [$\mathrm{m\,s^{-1}}$] and p_{ini} [$\mathrm{kg\,m^{-1}\,s^{-2}}$] denote the initial velocity and pressure, respectively, $\boldsymbol{u}_{\mathrm{in}}$ [$\mathrm{m\,s^{-1}}$] is the prescribed inflow velocity at $\partial\Omega_{\mathrm{in}}$, and p_{out} [$\mathrm{kg\,m^{-1}\,s^{-2}}$] is the prescribed outflow pressure at $\partial\Omega_{\mathrm{out}}$. Initial and boundary conditions are further discussed in Section 2.3.

Chapter 2

Lattice Boltzmann method

A fluid can be described as a continuum and using a macroscopic perspective, where it is treated as a whole, and its state is characterized by macroscopic quantities such as density, flow velocity, or pressure. In this case the governing equations are the Navier-Stokes equations from Eqs. (1.1). A fluid's behavior can also be described on a microscopic scale by tracking the dynamics of all individual particles. A significant drawback of this approach is its evident computational complexity, which is directly proportional to the number of particles involved.

A compromise between these two approaches is the description of a fluid on a mesoscopic scale [14], which is based on the kinetic theory. The fluid is described using one-particle density function $\varphi(\boldsymbol{x}, \boldsymbol{\xi}, t)$ [$\mathrm{kg\,s^3\,m^{-6}}$], which describes the system in the space of positions $\boldsymbol{x}$ and microscopic velocities $\boldsymbol{\xi}$, and time t. The density functions represent the density of particles at a position $\boldsymbol{x}$, with microscopic velocity $\boldsymbol{\xi}$, at time t.

The one-particle density functions satisfy the Boltzmann transport equation [15]

$$\frac{\partial \varphi}{\partial t} + \sum_{i=1}^{3} \xi_i \frac{\partial \varphi}{\partial x_i} + \sum_{i=1}^{3} g_i \frac{\partial \varphi}{\partial \xi_i} = C(\varphi), \tag{2.1}$$

where $\boldsymbol{g}$ [$\mathrm{m\,s^{-2}}$] is the acceleration vector of external forces, and $C(f)$ [$\mathrm{kg\,s^2\,m^{-6}}$] is the collision operator.

Using the distribution functions φ, certain macroscopic quantities can be expressed as statistical moments [15], such as

$$\rho(\boldsymbol{x}, t) = \int_{\mathbb{R}^3} \varphi(\boldsymbol{x}, \boldsymbol{\xi}, t)\, \mathrm{d}\boldsymbol{\xi}, \tag{2.2a}$$

$$\rho(\boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{x}, t) = \int_{\mathbb{R}^3} \varphi(\boldsymbol{x}, \boldsymbol{\xi}, t)\, \boldsymbol{\xi}\, \mathrm{d}\boldsymbol{\xi}. \tag{2.2b}$$

The lattice Boltzmann method (LBM) is a numerical method based on the mesoscopic description of fluids. The numerical scheme of LBM can be derived by discretizing Eq. (2.1). In LBM, the spatial discretization is performed using a discrete equidistant lattice, and the discretization of velocity space utilizes a finite set of discrete microscopic velocities. These discrete velocity sets are described using velocity models denoted as DdQq, where d specifies the spatial dimension and q the number of different velocity directions at each lattice node (called sites).

In this thesis, we use the D3Q27 velocity model illustrated in Figure 2.1.

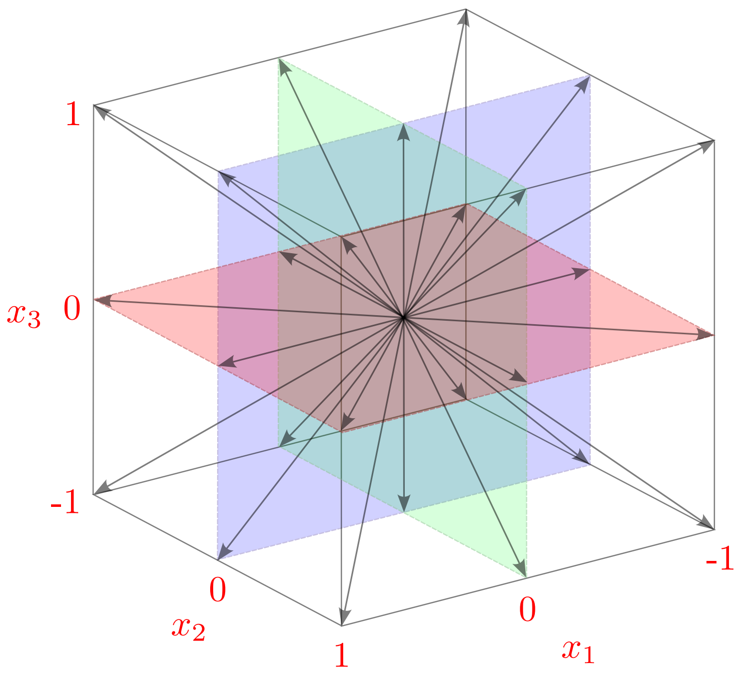


Figure 2.1: Geometric representation of the D3Q27 velocity model.

In the case of the D3Q27 model, the velocity space is discretized by a finite set of microscopic velocities defined as

$$(\boldsymbol{\xi}_k)_{k=1}^{27} = \left(\begin{matrix} \begin{pmatrix}0\\0\\0\end{pmatrix}, \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix}, \begin{pmatrix}-1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\-1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\-1\end{pmatrix}, \begin{pmatrix}0\\1\\1\end{pmatrix}, \begin{pmatrix}0\\1\\-1\end{pmatrix}, \begin{pmatrix}0\\-1\\1\end{pmatrix}, \begin{pmatrix}0\\-1\\-1\end{pmatrix}, \begin{pmatrix}1\\1\\0\end{pmatrix}, \begin{pmatrix}1\\-1\\0\end{pmatrix}, \begin{pmatrix}-1\\1\\0\end{pmatrix}, \\ \begin{pmatrix}-1\\-1\\0\end{pmatrix}, \begin{pmatrix}1\\0\\1\end{pmatrix}, \begin{pmatrix}1\\0\\-1\end{pmatrix}, \begin{pmatrix}-1\\0\\1\end{pmatrix}, \begin{pmatrix}-1\\0\\-1\end{pmatrix}, \begin{pmatrix}1\\1\\1\end{pmatrix}, \begin{pmatrix}1\\1\\-1\end{pmatrix}, \begin{pmatrix}1\\-1\\1\end{pmatrix}, \begin{pmatrix}1\\-1\\-1\end{pmatrix}, \begin{pmatrix}-1\\1\\1\end{pmatrix}, \begin{pmatrix}-1\\1\\-1\end{pmatrix}, \begin{pmatrix}-1\\-1\\1\end{pmatrix}, \begin{pmatrix}-1\\-1\\-1\end{pmatrix} \end{matrix} \right). \tag{2.3}$$

2.1 Discretization and non-dimensionalization

2.1.1 Spatial and temporal discretization

The computational domain $\Omega = (0; L_1) \times (0; L_2) \times (0; L_3)$, where L_i [m], $i = 1, 2, 3$, represent its dimensions, is discretized using a uniform grid with a spatial step size δ_x [m] constant in all coordinate directions. The computational domain and its closure are discretized as follows:

$$\hat{\Omega} = \left\{ \boldsymbol{x}_{i,j,k} = (i\delta_x,\ j\delta_x,\ k\delta_x)^T \,\middle|\, i \in \{1, \dots, N_1 - 1\}, j \in \{1, \dots, N_2 - 1\}, k \in \{1, \dots, N_3 - 1\} \right\}, \tag{2.4a}$$

$$\overline{\hat{\Omega}} = \left\{ \boldsymbol{x}_{i,j,k} = (i\delta_x,\ j\delta_x,\ k\delta_x)^T \,\middle|\, i \in \{0, \dots, N_1\}, j \in \{0, \dots, N_2\}, k \in \{0, \dots, N_3\} \right\}, \tag{2.4b}$$

where N_i [-] denotes the number of grid points in the x_i-direction, $i = 1, 2, 3$.

Since the physical domain boundaries are effectively located between the outermost grid nodes and the first interior nodes in the LBM discretization considered here [15], the physical domain is discretized by uniform $N_i - 1$ intervals in the x_i-direction, $i = 1, 2, 3$. Hence, it follows that $\delta_x = \frac{L_i}{N_i - 1}$, $i = 1, 2, 3$.

The boundary of the domain, $\partial\Omega$, is the closure of the union of disjoint parts given by

$$\partial\Omega = \overline{\partial\Omega_{\mathrm{W}} \cup \partial\Omega_{\mathrm{E}} \cup \partial\Omega_{\mathrm{N}} \cup \partial\Omega_{\mathrm{S}} \cup \partial\Omega_{\mathrm{F}} \cup \partial\Omega_{\mathrm{B}}}, \tag{2.5}$$

where $\partial\Omega_{\mathrm{W}}, \partial\Omega_{\mathrm{E}}, \partial\Omega_{\mathrm{N}}, \partial\Omega_{\mathrm{S}}, \partial\Omega_{\mathrm{F}}$, and $\partial\Omega_{\mathrm{B}}$ represent the west, east, north, south, front, and back boundaries, respectively, as illustrated in Figure 2.2. The discretized boundary of the computational domain is given by

$$\partial\hat{\Omega} = \overline{\hat{\Omega}} \setminus \hat{\Omega}, \tag{2.6}$$

with its corresponding parts denoted by $\partial\hat{\Omega}_{\mathrm{W}}, \partial\hat{\Omega}_{\mathrm{E}}, \partial\hat{\Omega}_{\mathrm{N}}, \partial\hat{\Omega}_{\mathrm{S}}, \partial\hat{\Omega}_{\mathrm{F}}$, and $\partial\hat{\Omega}_{\mathrm{B}}$.

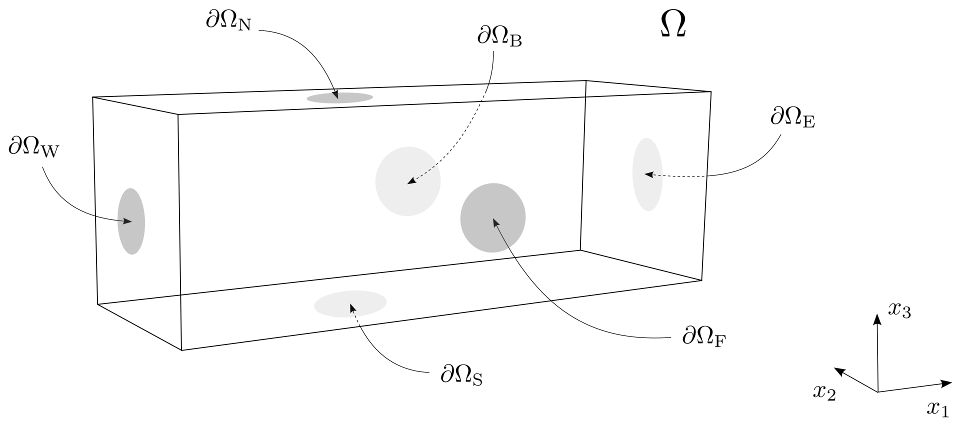

$\partial\Omega_{\mathrm{N}}$
$\partial\Omega_{\mathrm{B}}$
Ω
$\partial\Omega_{\mathrm{E}}$
$\partial\Omega_{\mathrm{W}}$
$\partial\Omega_{\mathrm{F}}$
$\partial\Omega_{\mathrm{S}}$
x_3
x_2
x_1

Figure 2.2: Schematic representation of the computational domain Ω and its boundary $\partial\Omega$.

The time interval of interest $\mathcal{I} = [0, T]$, with $T > 0$ [s], is discretized by the set

$$\hat{\mathcal{I}} = \{i\delta_t \mid i \in \{0, \ldots, N_t\}\}, \tag{2.7}$$

where N_t represents the number of discrete time steps and the time step δ_t [s] satisfies $\delta_t = \frac{T}{N_t}$.

2.1.2 Non-dimensionalization using lattice units

In LBM it is common to work with non-dimensional quantities referred to as lattice units. In the following relations and derivations, all quantities in lattice units are marked with the superscript L.

Assume that l_0 [m] is the characteristic length for a given problem and l_0^L [−] is its counterpart in lattice units. Suppose that l_0 is divided into N_l intervals of spatial step Δl. We can write

$$\Delta l = \frac{l_0}{N_l}, \tag{2.8a}$$

$$\Delta l^L = \frac{l_0^L}{N_l}, \tag{2.8b}$$

where Δl^L [−] is the spatial step in lattice units. From Eqs. (2.8) it follows that

$$\Delta l^L = \frac{l_0^L}{l_0}\Delta l. \tag{2.9}$$

Similarly, if we assume that t_0 [s] is the characteristic time for a given problem and t_0^L [−] is its counterpart in lattice units, we can derive the relation for the conversion of the time step given by

$$\Delta t^L = \frac{t_0^L}{t_0}\Delta t. \tag{2.10}$$

For simplicity, in this work, we assume that the spatial step Δl^L and the time step Δt^L in lattice units are $\Delta l^L = \Delta t^L = 1$. All subsequent derivations rely on these assumptions.

Furthermore, one can derive the conversion relation for velocity given by

$$\boldsymbol{u}^L = \frac{\Delta t}{\Delta l}\boldsymbol{u}. \tag{2.11}$$

Once transitioned to lattice units the uniform grid $\hat{\Omega}$ becomes a discrete equidistant lattice which is used in LBM:

$$\widetilde{\Omega} = \left\{ \tilde{\boldsymbol{x}}_{i,j,k} = (i,\ j,\ k)^T \;\middle|\; i \in \{1,\dots,N_1 - 1\}, j \in \{1,\dots,N_2 - 1\}, k \in \{1,\dots,N_3 - 1\} \right\}, \tag{2.12a}$$

$$\overline{\widetilde{\Omega}} = \left\{ \tilde{\boldsymbol{x}}_{i,j,k} = (i,\ j,\ k)^T \;\middle|\; i \in \{0,\dots,N_1\}, j \in \{0,\dots,N_2\}, k \in \{0,\dots,N_3\} \right\}. \tag{2.12b}$$

Moreover, the time interval in lattice units is discretized by

$$\widetilde{\mathcal{I}} = \{\, i \mid i \in \{0,\dots,N_t\}\} . \tag{2.13}$$

For other important quantities the following relations hold [15]:

$$\text{kinematic viscosity:} \quad \nu^L = \frac{t_0 (l_0^L)^2}{{l_0}^2 t_0^L}\nu = \frac{\Delta t}{\Delta l^2}\nu, \tag{2.14a}$$

$$\text{density:} \quad \rho^L = \frac{\rho_0^L}{\rho_0}\rho, \tag{2.14b}$$

$$\text{distribution function:} \quad \varphi^L = \frac{1}{\rho_0}\left(\frac{l_0}{l_0^L}\right)^3 \left(\frac{t_0^L}{t_0}\right)^3 \varphi. \tag{2.14c}$$

Note that characteristic length (l_0 [m]), time (t_0 [s]), and density (ρ_0 [kg m^{-3}]) values are chosen based on the given physical problem. The computational domain's largest dimension or the size of an obstacle within the flow is typically chosen as l_0. Detailed derivations of these relations can be found in [15]. From Eq. (2.14a), it can be seen that for a given spatial step Δl, the time step Δt is linked to the value of ν^L.

Throughout the remainder of this chapter, lattice units will be used exclusively. For brevity, the superscript L is omitted, although all quantities remain non-dimensional. In subsequent chapters, the discussion will revert to physical units, with the superscript L reintroduced where necessary to distinguish between lattice units and physical quantities.

2.1.3 Discrete Boltzmann transport equation

When using the D3Q27 model, we work with a set of distribution functions

$$\{\varphi_k(\boldsymbol{x}, t) \mid k = 1,\dots,27\}, \ \forall \boldsymbol{x} \in \widetilde{\Omega},\ \forall t \in \widetilde{\mathcal{I}}, \tag{2.15}$$

where the indices correspond to the directions of microscopic velocities from Eq. (2.3).

It can be shown that by discretizing Eq. (2.1), we obtain the form

$$\varphi_k\left(\boldsymbol{x} + \Delta t \boldsymbol{\xi}_k, t + \Delta t\right) = \varphi_k(\boldsymbol{x}, t) + \mathcal{C}_k(\boldsymbol{x}, t) + \mathcal{S}_k(\boldsymbol{x}, t), \quad k \in \{1,\dots,27\}, \ \forall \boldsymbol{x} \in \widetilde{\Omega},\ \forall t \in \widetilde{\mathcal{I}}, \tag{2.16}$$

where C_k represents the discrete collision operator, and $\mathcal{S}_k$ is the discrete forcing term. Details of the derivation can be found in [15].

The choice of the discrete collision operator C_k in Eq. (2.16) defines the specific variant of LBM. Several choices for C_k exist, including single relaxation time (SRT-LBM) [43], multiple relaxation time (MRT-LBM) [44], central moment (CLBM) [45], entropic (ELBM) [46], and cumulant-based (CuLBM) [43]. In this work, we use the cumulant collision operator, as detailed in [43].

To simplify the notation of the discrete equation, we define the post-collision distribution functions φ_k^* as

$$\varphi_k^*(\boldsymbol{x}, t) = \varphi_k(\boldsymbol{x}, t) + C_k(\boldsymbol{x}, t) + \mathcal{S}_k(\boldsymbol{x}, t), \quad k \in \{1, \dots, 27\},\ \forall \boldsymbol{x} \in \widetilde{\Omega},\ \forall t \in \widetilde{\mathcal{I}}. \tag{2.17}$$

Using φ_k^*, we can express Eq. (2.16) in the form

$$\varphi_k\left(\boldsymbol{x} + \Delta t \boldsymbol{\xi}_k, t + \Delta t\right) = \varphi_k^*(\boldsymbol{x}, t), \quad k \in \{1, \dots, 27\},\ \forall \boldsymbol{x} \in \widetilde{\Omega},\ \forall t \in \widetilde{\mathcal{I}}. \tag{2.18}$$

This formulation allows for an explicit prescription of calculating the distribution functions.

2.1.4 Macroscopic quantities

The relations for calculating key macroscopic quantities, including density, momentum, and pressure, can be derived from Eqs. (2.2) through discretization [15] as follows:

$$\rho = \sum_{k=1}^{27} \varphi_k, \tag{2.19a}$$

$$\rho \boldsymbol{u} = \sum_{k=1}^{27} \varphi_k \boldsymbol{\xi_k} + \rho \frac{\Delta t}{2} \boldsymbol{g}, \tag{2.19b}$$

$$p = p_0 + c_s^2 (\rho - \rho_0), \tag{2.19c}$$

where p_0 [−] is the non-dimensional reference pressure, c_s [−] is the non-dimensional (lattice) speed of sound, and ρ_0 is the non-dimensional reference density. For the D3Q27 model, the lattice speed of sound is given by $c_s = \frac{1}{\sqrt{3}}$. Furthermore, we consider $\rho_0 = 1$. A detailed description of the calculation of macroscopic quantities is provided in [15].

2.2 LBM algorithm

The LBM algorithm can be summarized in the following steps:

1. **Initialization** of the initial conditions on the grid, as described in Section 2.3.
2. **Iterative process** ending after a predefined termination condition is met.
   (a) **Streaming** of post-collision distribution functions φ_k^* in the respective directions $\boldsymbol{\xi_k}$.
   (b) **Macroscopic quantities computation** using the Eqs. (2.19).
   (c) **Collision**, where the post-collision state of the distribution function is calculated using Eq. (2.18), and **application of the boundary conditions**, discussed in Section 2.3.
3. **End of the algorithm.**

A flowchart of the LBM algorithm is shown in Figure 2.3.

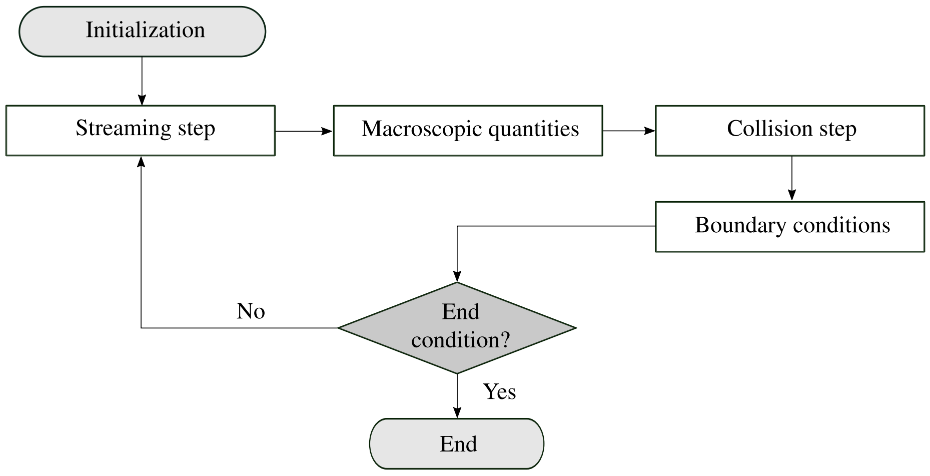


Figure 2.3: Flowchart of the LBM algorithm.

2.3 Initial and boundary conditions

The choice of initial and boundary conditions is an integral part of the lattice Boltzmann method to ensure consistency the mesoscopic description. Therefore, the chosen initial and boundary conditions are described in more detail in this section.

2.3.1 Initial condition

In this work, the equilibrium distribution function φ^{eq} is used to set the initial condition. The second order approximation of the equilibrium distribution function is used in the form

$$\varphi_k^{\mathrm{eq}} = \rho w_k \left(1 + \frac{\boldsymbol{\xi_k} \cdot \boldsymbol{u}}{c_s^2} + \frac{(\boldsymbol{\xi_k} \cdot \boldsymbol{u})^2}{2c_s^4} - \frac{\boldsymbol{u} \cdot \boldsymbol{u}}{2c_s^2}\right), \quad k \in \{1, \dots, 27\}, \tag{2.20}$$

where w_k are the weights specific to the chosen velocity model. For the D3Q27 model, these weights are defined by [15]

$$w_k = \begin{cases} \frac{8}{27}, & k = 1, \\ \frac{2}{27}, & k = 2, 3, \dots, 7, \\ \frac{1}{54}, & k = 8, 9, \dots, 19, \\ \frac{1}{216}, & k = 20, 21, \dots, 27. \end{cases} \tag{2.21}$$

The initial density ρ and the velocity $\boldsymbol{u}$, are denoted by ρ_{ini} and $\boldsymbol{u}_{\mathrm{ini}}$, respectively. At each lattice site $\boldsymbol{x} \in \widetilde{\Omega}$ at time $t = 0$, the distribution functions are initialized as

$$\varphi_k(\boldsymbol{x}, 0) = \varphi_k^{\mathrm{eq}}(\rho_{\mathrm{ini}}(\boldsymbol{x}), \boldsymbol{u}_{\mathrm{ini}}(\boldsymbol{x})), \quad k \in \{1, \dots 27\}. \tag{2.22}$$

This approach assumes that the non-equilibrium component of the distribution functions, defined by $\varphi_k^{\mathrm{neq}} = \varphi_k - \varphi_k^{\mathrm{eq}}$, can be neglected, and the distribution functions can be approximated by their equilibrium part. A significant advantage of this choice of initial condition approximation is its easy implementation. While more advanced initialization methods exist [14], the equilibrium-based approach is used in this work.

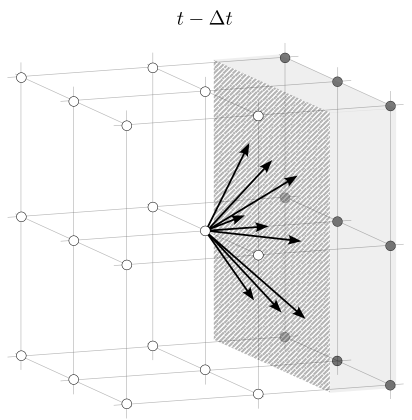
2.3.2 Boundary conditions

Bounce-back boundary condition

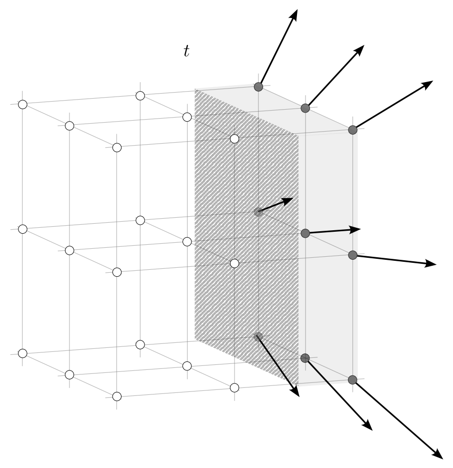
The first boundary condition discussed is the bounce-back boundary condition, specifically its *full-way* variant [15]. The bounce-back boundary condition is typically used for modeling the interface between a fluid and a solid. Its advantage is that it satisfies the no-slip condition at the fluid-solid interface while remaining straightforward to implement. The principle of the bounce-back boundary condition is that at the interface, the distribution functions corresponding to particles with microscopic velocity $\boldsymbol{\xi}_{\boldsymbol{k}}$ are reflected back into the directions from which they arrived at the node, with velocity $\boldsymbol{\xi}_{\bar{\boldsymbol{k}}} = -\boldsymbol{\xi}_{\boldsymbol{k}}$.

When using this boundary condition, the fluid-solid interface is located halfway between the fluid and solid nodes. For curved boundaries that are not parallel to the grid, the bounce-back method leads to a "staircase" shape of the boundary, which can limit the accuracy of the simulation.

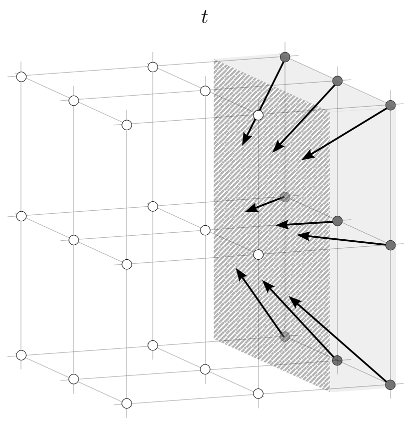
In this approach, particles are reflected over two time steps. During this time, the particles reach the solid nodes, where their direction is reversed and they are streamed back, as schematically shown in Figure 2.4.



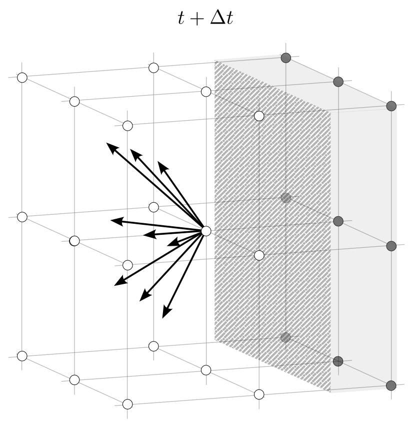
(a) Step before the streaming at time $t - \Delta t$.



(b) Step after streaming at time t.



(c) The distribution functions are reversed.



(d) Step after the streaming at time $t + \Delta t$.

Figure 2.4: Schematic representation of the fullway bounce-back boundary condition for the D3Q27 velocity model. White points represent fluid sites, and gray points represent wall sites. The wall is represented by a gray plane.

An alternative to the fullway method is the *halfway* variant of the bounce-back boundary condition, which completes the reflection process within a single time step. Details can be found in [15]. In this work, however, we limit ourselves to the fullway variant.

Free outflow boundary condition

When using the free outflow boundary condition at the outlets, each distribution function for directions pointing out of the domain is set equal to the value it had at the adjacent node inside the computational domain in the previous timestep. While this boundary condition is numerically stable, its usage can cause unphysical behavior of pressure and velocity field near the outflow boundary. Details on the free outflow boundary condition can be found in [14].

Constant inflow boundary condition

The constant inflow boundary condition imposes a steady velocity vector $\boldsymbol{u} = (u_1, u_2, u_3)^T$ at the inflow boundaries of the computational domain. This type of Dirichlet boundary condition ensures a constant mass flux entering the computational domain, representing a simplified inlet profile. The inflow boundary condition is realized by modifying the distribution functions φ_k to reflect the prescribed macroscopic velocity. One of the main advantages of the constant inflow boundary condition its low difficulty of implementation. Details on the constant inflow boundary condition can be found in [15].

2.4 Notes on the implementation

As mentioned in the introduction, the numerical solution using LBM was based on a code developed at the Department of Mathematics of FNSPE, CTU in Prague, which is used to solve the Navier-Stokes equations for a Newtonian incompressible fluid. The program is implemented in C++ using the TNL library [47, 48] and employs parallelization on a GPU using the CUDA platform. The variant of the lattice Boltzmann method used, CuLBM, is implemented in the code for the D3Q27 model.

For the purposes of this work, the previously developed code presented in [20] was extended and enhanced to support three-dimensional simulations. The most significant modifications include the implementation of stress tensor calculations using finite differences and the calculation of other monitored quantities (objective functions used in the optimization process) with their subsequent output to files for further analysis.

Additionally, the code has been designed with a templated structure, allowing flexibility in adapting to the optimization process. The computational domain, including its dimensions and the classification of nodes as fluid or wall, is dynamically defined based on the current parameters provided by the optimization framework.

Notable examples of newly introduced implementations can be found in Appendix A.

Chapter 3

Geometry generation

In this chapter, the process of geometry generation for the optimization workflow is described. The generated geometry serves as the foundation for simulation using the lattice Boltzmann method described in Chapter 2 and is directly defined by the optimization parameters that arise from the used optimization algorithm.

These parameters are used to dynamically populate predefined geometry templates implemented in Gmsh, a mesh generation software supporting parametric definition of geometries (associated with the file extension `.geo`) [49]. The filled template is used to generate a stereolitography file (STL) [50] associated with the `.stl` file extension. This generated STL file is then loaded into the Trimesh [51] Python package, which voxelizes the geometry based on the specified resolution for the LBM simulation. The voxelized geometry is stored as a 3D NumPy array [52], with options for exporting it in either plain text format (`.txt`) or the TNL format (`.tnl`) native to the TNL library. The TNL format offers performance advantages when used with TNL-based LBM implementations.

An overview of the entire process illustrating key steps from parameter definition to simulation-ready output files is presented in Figure 3.1.

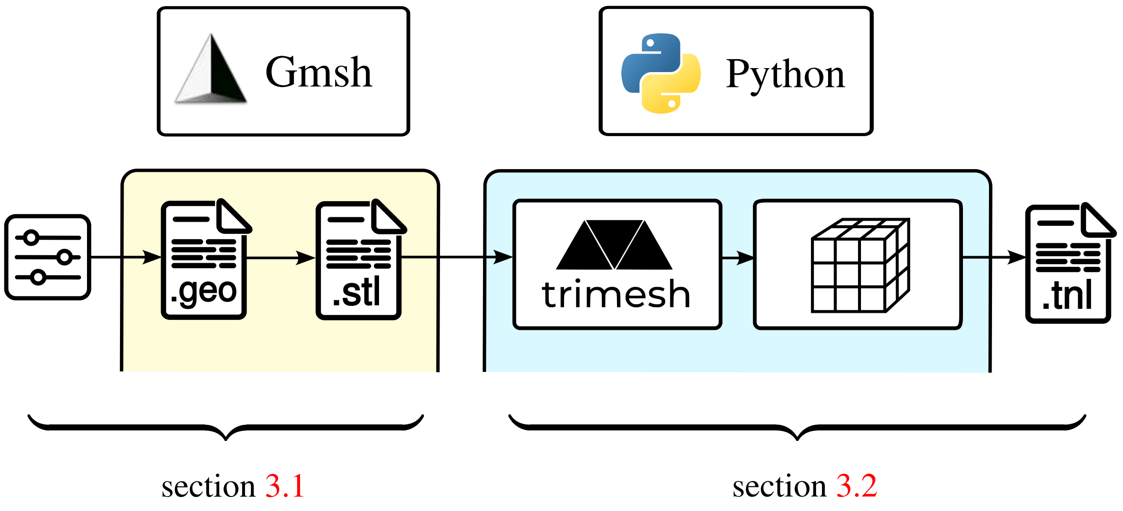


Figure 3.1: Overview of the geometry generation process. The optimization parameters are used to fill in a Gmsh .geo template file, which is used to generate an `.stl` geometry. This geometry is subsequently loaded by the Trimesh Python package and is voxelized and exported either as a `.tnl` object or a plain `.txt` file.

A custom installable Python package, named `meshgen`, was implemented to encapsulate the entire process of geometry generation, from defining the geometry templates to preparing the output files for simulations. Its structure reflects the key steps in this process. The Python modules within the package are designed to handle specific tasks, such as loading and processing templates, voxelizing geometries, and providing utility functions. The modules and its specific roles in the process are discussed in detail in the following sections.

Figure 3.2 provides an overview of the package's structure. The package is available upon request on Github at `https://github.com/buresjan/meshgen`.

```
meshgen
├─ geo_templates .......... // gmsh template files
├─ stl_models ............. // temporary .stl files
├─ output ................. // .tnl or .txt file with mesh
├─ mesher.py .............. // loads Gmsh template file and generates .stl
├─ voxels.py .............. // voxelizing the .stl file
├─ utilities.py ........... // utility functions
└─ geometry.py ............ // Geometry class definition providing the user
                               API for generating and managing voxelized
                               geometries
```

Figure 3.2: Overview of the `meshgen` package structure. Each module has its specific role in the geometry generation process.

3.1 Template-based geometry generation

The geometry generation process begins with predefined Gmsh template files. These templates, which are created by the user, serve as the foundation for defining the problem-specific geometry. The template itself defines the problem being solved. A full example of a Gmsh template file used in this workflow can be found in Appendix B.

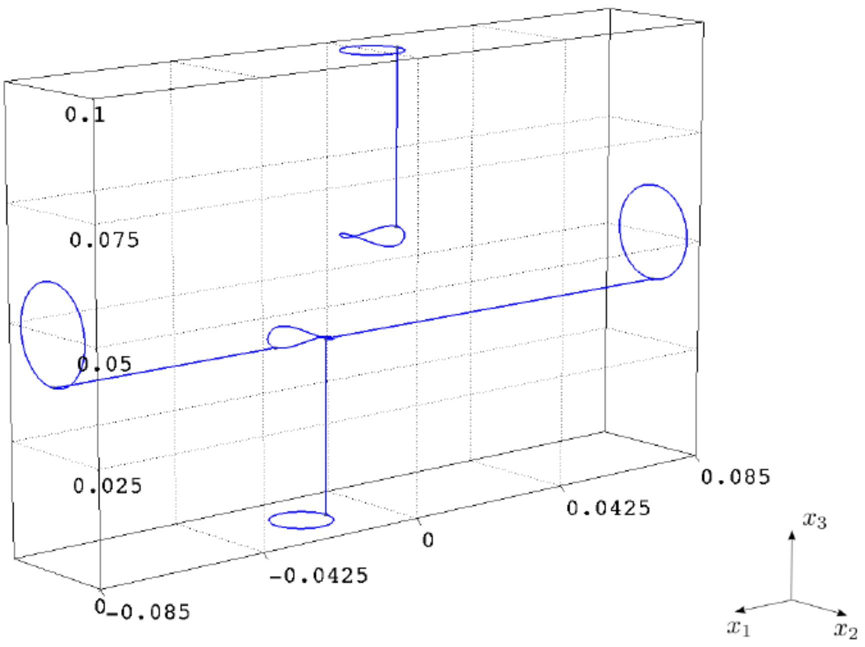
Each Gmsh template file incorporates placeholders for variable parameters, marked with the "DEFINE_" string prefix (e.g., `DEFINE_OFFSET`, `DEFINE_ANGLE`). These placeholders are replaced programmatically with specific values provided by the optimization algorithm. Importantly, the number of placeholders in the template must correspond to the dimension of the optimization problem being solved, ensuring that each optimization parameter is appropriately mapped to a specific aspect of the geometry.

The Python function responsible for this substitution process is shown in Listing 3.1. The function reads the template file, replaces the placeholders with the parameter values, and writes the modified geometry definition to a new file.

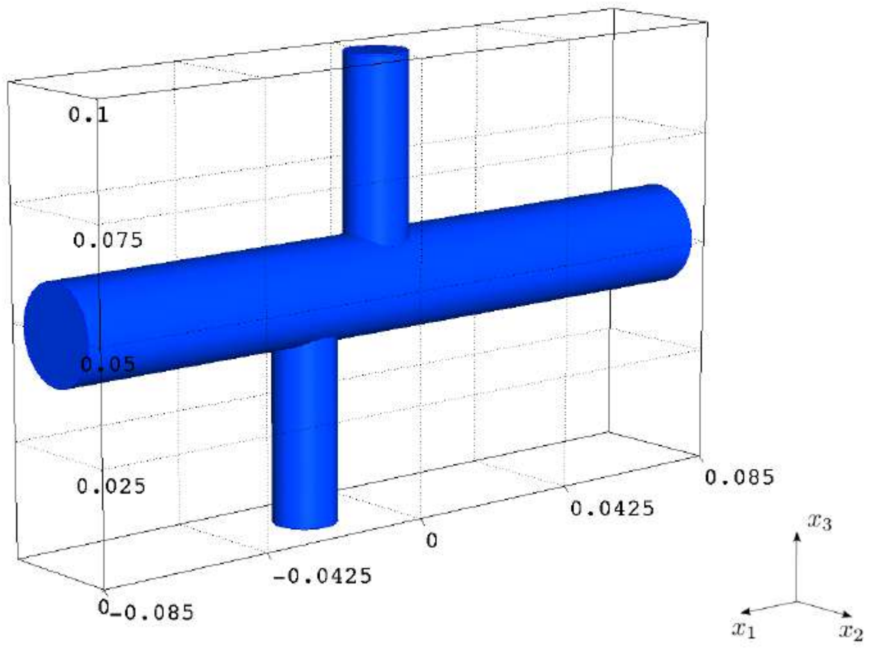
```python
def modify_geo_file(input_file_path, output_file_path, **kwargs):
    # Open and read the content of the original GEO file
    with open(input_file_path, "r") as file:
        file_data = file.read()

    # Iterate through each keyword argument to replace placeholders
    for variable, value in kwargs.items():
        # Construct the placeholder string
        placeholder_variable = "DEFINE_" + variable.upper()
        replace_string = str(value)

    # Replace the placeholder with the actual value
    file_data = file_data.replace(placeholder_variable, replace_string)

    # Write the modified data to the new GEO file
    with open(output_file_path, "w") as file:
        file.write(file_data)
```

Code Listing 3.1: Function for modifying Gmsh template files.

The modified Gmsh file is then processed to generate the desired geometry. Figure 3.3a illustrates the geometry outlines produced by the template example from Appendix B. These outlines serve as the basis for further processing. Once the geometry outlines are defined, Gmsh is used to export the geometry as an `.stl` file. The `.stl` file captures the full 3D structure. Figure 3.3b shows the `.stl` file generated from the template example in Appendix B.



(a) Geometry outlines generated using the Gmsh template.



(b) 3D geometry exported to an `.stl` file. The blue volume represents the fluid sites later projected onto the LBM mesh.

Figure 3.3: 3D geometry exported to an `.stl` file. The blue volume represents the fluid later projected onto the LBM lattice.

3.2 Mesh processing

The mesh processing pipeline is designed to take the generated `.stl` files and convert them into a voxelized representation suitable for numerical simulations. This section describes the main steps involved in the mesh processing pipeline as illustrated in Figure 3.4.

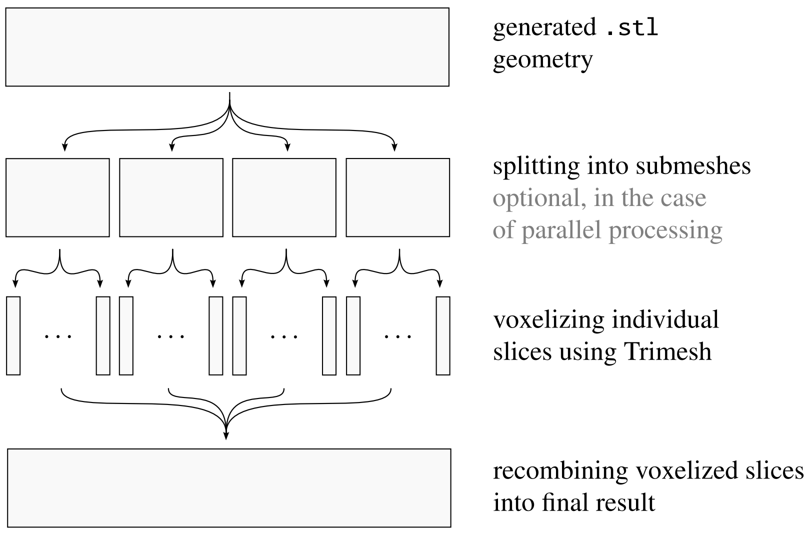


Figure 3.4: Overview of the mesh processing pipeline, including optional splitting into submeshes, voxelization of slices, and recombination into the final voxelized geometry.

While the Trimesh package can load and voxelize an `.stl` file in a single step, this approach becomes computationally expensive for large-scale geometries such as those used in this work. To address this issue, the mesh is split into smaller individual slices along the leading direction, which are then voxelized and recombined. On top of that, the process supports parallelization by grouping the slices into multiple independent groups.

3.2.1 Parallel processing of submeshes

The `.stl` file can be first loaded as a mesh using the Trimesh package and then divided into smaller submeshes along its leading direction. This division reduces the computational cost by distributing the workload across multiple CPU cores, hence significantly reducing processing time.

The parallelization is achieved by using Python's `ProcessPoolExecutor` available in the built-in `concurrent.futures` module [53], as shown in Listing 3.2. The `ProcessPoolExecutor` allows tasks to be executed in parallel each running independently on separate CPU cores [53].

```python
# Parallel processing of submeshes
with ProcessPoolExecutor(max_workers=num_processes) as executor:
    # Prepare arguments for each submesh processing
    futures = [
        executor.submit(
        process_submesh,
        submsh,
        margin,
        voxel_size,
        leading_direction
        ) for submsh, margin in zip(submeshes, margins)
    ]

    # Collecting results
    components = [
        future.result() for future in
        tqdm(futures, total=len(submeshes), desc="Voxelizing")
    ]
```

Code Listing 3.2: Parallel processing of submeshes for voxelization.

3.2.2 Voxelization and recombination of slices

The mesh, whether processed as a whole or divided into submeshes (if parallelization is used), is split into individual slices based on their thickness that is defined by the user. Each slice is then converted into a voxelized representation using the Trimesh package, which provides a tool for voxelizing 3D geometries, as shown in Listing 3.3.

```python
def voxelize_elementary(mesh, voxel_size):
    # Voxelize mesh with the specified voxel size
    # In this case, the mesh represents an elementary slice
    return mesh.voxelized(voxel_size).matrix
```

Code Listing 3.3: Voxelization of a mesh slice using the Trimesh package [51].

Finally, the voxelized slices are recombined using the `numpy.concatenate` function [52] along the leading direction to form the final geometry ready for simulation.

3.2.3 The `Geometry` class: a user-friendly interface

The `Geometry` class serves as a high-level interface that encapsulates the entire process of mesh generation. It provides users with control over the voxelization process, including options to define the slice thickness and distribute the workload into parallel tasks. The class also supports visualization and file export in multiple formats.

The `Geometry` class supports the following key functionalities:

- **Voxelization**: Converts the geometry into a voxelized representation based on a Gmsh `.geo` template as discussed in 3.2.2.
- **Parallel Processing**: Allows users to split the geometry into slices paramater) and process them in parallel across multiple CPU cores as discussed in 3.2.1.

  The `split` parameter determines the number of slices into which the mesh is divided along its leading direction. The `num_processes` parameter specifies the number of groups into which the slices are distributed, with each group processed on a separate CPU core.
- **Mesh Exporting**: Exports the voxelized mesh as binary NumPy object (`.npy`), plain text file (`.txt`), or LBM simulation-compatible `.tnl` format. Exporting in the `.tnl` format is achieved using the PyTNL package [54], which provides Python bindings for TNL.
- **Visualization**: Visualizes the voxelized geometry, primarily for debugging purposes. Visualization is implemented using the Mayavi Python package [55], which allows inspecting geometries in an interactive 3D environment. An example visualization, generated using the `.geo` template file from Appendix B, is shown in Figure 3.5.

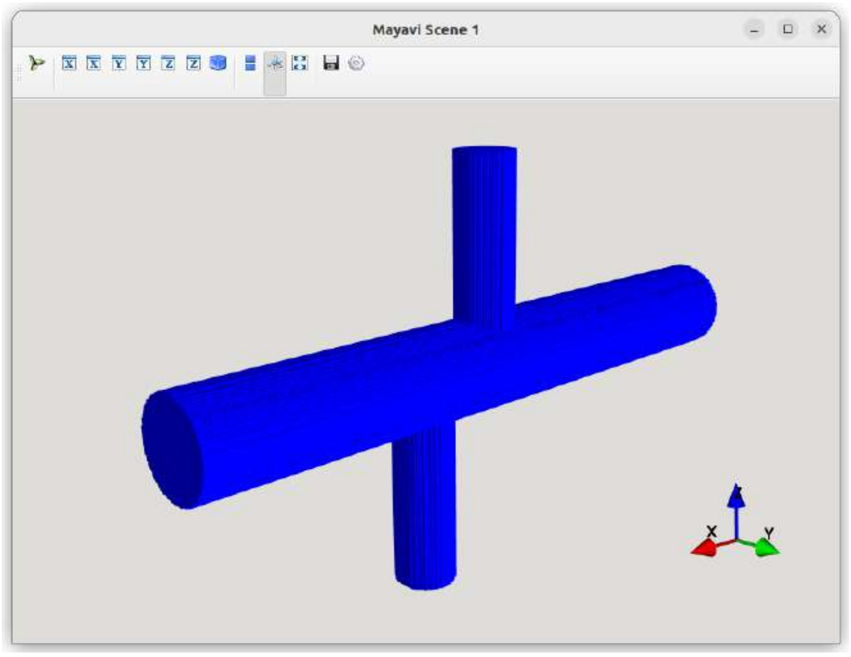


Figure 3.5: Visualization of the voxelized geometry using the Mayavi package [55].

Finally, the following code demonstrates the usage of the `Geometry` class, from defining the object to exporting the voxelized geometry in multiple formats and visualizing it.

```python
from meshgen.geometry import Geometry

# Initialize a Geometry instance with parameters
geom = Geometry(
    name="junction_1d",  # name of the .geo template file
    resolution=8,        # desired resolution for the LBM simulation
    split=1024,          # slice thickness; mesh is split into 1024
                         # slices
    num_processes=8,     # distribute slices into 8 groups, each
                         # processed on a separate CPU core
    offset=0.02,         # any optimization parameters specific to
                         # the problem
    expected_in_outs={"W", "E", "S", "N"} # list of the parts of the
                                          # computational domain that
                                          # are expected to include
                                          # the inlets and outlets;
                                          # for practical purposes
)

# Generate the voxel mesh
geom.generate_voxel_mesh()

# Save the voxel mesh in various formats
geom.save_voxel_mesh("voxel_mesh.npy")
geom.save_voxel_mesh_to_text("voxel_mesh.txt")
geom.save_voxel_mesh_to_tnl("voxel_mesh.tnl")

# Visualize the voxelized geometry
geom.visualize()
```

Code Listing 3.4: Example usage of the `Geometry` class.

Chapter 4

Mathematical optimization

Mathematical optimization, also known as mathematical programming, is a broad field that covers a wide range of methods and problem types, including linear and nonlinear optimization, convex programming, and integer programming [56]. Selecting the right optimization method for a given problem is a crucial step, as factors such as nonlinearity, the presence of constraints, and the computational cost of evaluating the objective function can significantly influence performance. In many cases, classic gradient-based algorithms—such as the Davidon-Fletcher-Powell [57] or Broyden-Fletcher-Goldfarb-Shanno [58] methods—can be applied efficiently. However, these methods become less practical when gradients are difficult to obtain analytically and must instead be computed numerically. Moreover, numerical gradient estimation can be slow, computationally expensive, and prone to inaccuracies.

In this work, we address a constrained, nonlinear optimization problem in which the objective function is evaluated numerically and may take several hours per evaluation. The discipline that deals with problems of these characteristics—where the objective function (or constraints) is difficult to evaluate and is given by a so-called black-box[1]—is called black-box optimization (hereafter referred to as BBO).

Our previous works [19, 20] examined how various optimization algorithms, including gradient-based methods, could be coupled with such numerical simulations. Those studies showed that gradient-based approaches are not well-suited to the black-box setting addressed in this work. In contrast, gradient-free methods avoid the costs and complexities of gradient approximation, making them a more suitable option in these scenarios. This naturally guides us toward employing gradient-free optimization methods and black-box optimization techniques in this work.

When dealing with constraints, standard approaches such as penalty and barrier methods offer systematic ways to ensure feasibility [60]. However, they often require multiple runs of a modified, unconstrained subproblem [60], which can become prohibitively time-consuming in our context. Therefore, we adopt a simplified approach to constraint handling known as the extreme barrier function [59], which lets us incorporate constraints directly without incurring extensive computational costs.

This chapter is structured as follows: First, we formally define the general optimization problem under consideration. Next, we discuss black-box optimization methods and their relevance to our scenario. Finally, we present the proposed optimization framework in detail, highlighting its components and illustrating how they are interconnected.

[1]In programming, a black-box refers to a system whose internal mechanisms are unknown to the user. This means that the user generally has access only to the system's input and output [59].

4.1 General optimization problem

Let $m, n, q \in \mathbb{N}$. Define the continuous functions $f : \mathbf{D} \to \mathbb{R}$, $\boldsymbol{g} : \mathbf{D} \to \mathbb{R}^m$, $\boldsymbol{h} : \mathbf{D} \to \mathbb{R}^q$, where $\mathbf{D} = \mathrm{Dom}\,(f) \cap \mathrm{Dom}\,(\boldsymbol{g}) \cap \mathrm{Dom}\,(\boldsymbol{h})$, i.e., $\mathbf{D}$ is the intersection of the domains of the given functions. Next, define the set

$$\mathbf{X} = \{\boldsymbol{x} \in \mathbf{D} \subseteq \mathbb{R}^n \mid \boldsymbol{g}(\boldsymbol{x}) \le \mathbf{0} \wedge \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}\}, \tag{4.1}$$

where the inequality $\boldsymbol{g} \le \mathbf{0}$ and equality $\boldsymbol{h} = \mathbf{0}$ are understood component-wise. The general goal of mathematical optimization is to solve the problem

$$\min_{\boldsymbol{x} \in \mathbf{X}} f(\boldsymbol{x}). \tag{4.2}$$

The function f being minimized is called the objective function, $\mathbf{D}$ is referred to as the domain of the problem, and $\mathbf{X}$ is called the set of feasible solutions of the problem [60].

When classifying optimization problems, we refer to what are known as constraints. These are determined by the definition of the set $\mathbf{X}$, i.e., the equality and inequality conditions for the functions $\boldsymbol{g}$ and $\boldsymbol{h}$, and by the domain $\mathbf{D}$. Constraints defined by $\boldsymbol{g}(\boldsymbol{x}) \le \mathbf{0} \wedge \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}$ are called explicit constraints, while those determined by the domain $\mathbf{D}$ are called implicit constraints. The optimal solution of the problem given by Eq. (4.2) is denoted by $\boldsymbol{x}^\star \in \mathbf{X}$ and is defined as

$$\boldsymbol{x}^\star = \operatorname*{argmin}_{\boldsymbol{x} \in \mathbf{X}} f(\boldsymbol{x}). \tag{4.3}$$

Note that the optimal solution may not be unique, and we refer to the set of all optimal solutions as the optimal set. It is also important to note that the problem of finding an optimal solution can equivalently be formulated as finding the maximum of the function $-f$ over the same set $\mathbf{X}$, enabling the use of the same techniques for solving maximization problems [60, 61].

4.2 Black-box optimization

In BBO, it is typically not assumed that the objective function is continuous or differentiable [59, 62, 63]. It is worth noting that BBO is often confused with derivative-free optimization (DFO), which encompasses methods and techniques for objective functions whose derivatives are unknown or difficult to compute [59, 62, 64]. These two disciplines share many common characteristics, but they differ primarily in that, within DFO, the formula for calculating the derivative of the objective function may still be known. Furthermore, DFO focuses mainly on methods that can be reliably analyzed mathematically in terms of convergence and stopping criteria, which is often not possible for BBO methods [59]. Therefore, although the terms BBO and DFO are often used interchangeably, in this work, we will treat them as two distinct disciplines [59].

Additionally, it should be noted that various classifications of methods within BBO can be found in the literature. In this work, we use the classification presented in [59], distinguishing between heuristic methods, direct search methods, and methods based on surrogate models. Each of these classes are briefly described in this section.

4.2.1 Heuristic methods

Heuristic optimization methods often rely on different predefined rules or even trial-and-error when seeking the solution of an optimization problem. These methods usually do not guarantee optimal solutions, but they are often effective for finding near-optimal results in a reasonable amount of time. Heuristic methods include genetic algorithms, detailed in [59], along with various other heuristic approaches.

In this section, however, we focus on a different widely used heuristic method, the Nelder-Mead method, also known as the simplex method[2,3][66].

The Nelder-Mead method finds a solution to an optimization problem by iteratively constructing simplexes. The process begins by initializing a starting simplex. The objective function is then evaluated at each vertex of this simplex. In each subsequent iteration, the simplex is transformed in order to move closer to the position of the sought stationary point of the objective function. The transformation of the simplex involves manipulating its points using predefined operations – expansion, reflection, contraction (inner and outer), and shrinking, which are schematically illustrated in Figure 4.1.

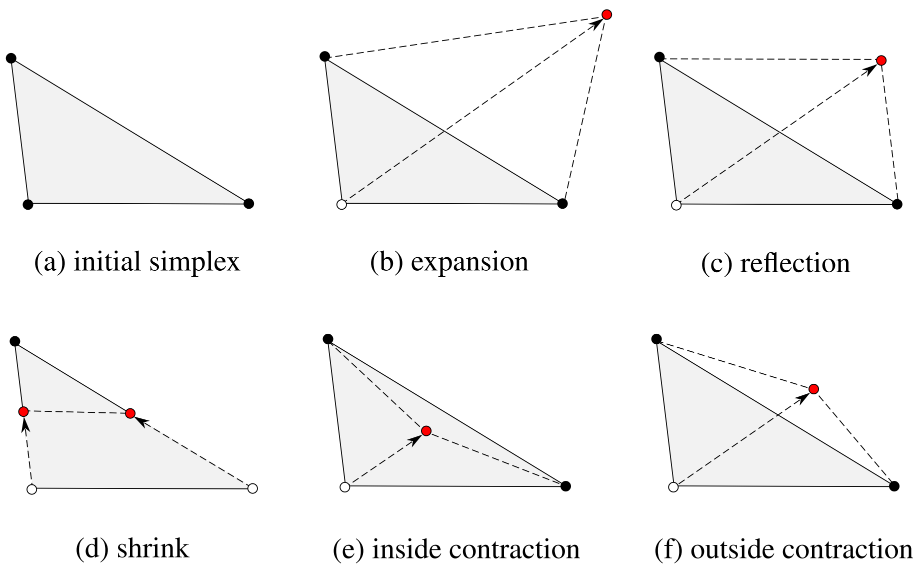


Figure 4.1: A schematic representation of the operations used to transform simplexes in the Nelder-Mead method. The vertices generated by applying each operation are shown in red. For clarity, the operations are depicted in $\mathbb{R}^2$.

The transformations performed during each iteration are determined by comparing the function values at the vertices of the simplex. The newly formed simplex shares either exactly one vertex or exactly n vertices with the simplex from the preceding iteration. The algorithm continues to iteratively transform the simplex until a stopping condition (specified by the user) is met [59]. Details of the Nelder-Mead method, including the algorithm's description and the choice of stopping condition, are discussed in [59, 62, 66]. The full Nelder-Mead method algorithm is presented in Algorithm 1.

The heuristic nature of the Nelder-Mead method stems from the fact that its principle is based on a somewhat random search of the space using predefined rules. Several iterations of space exploration using simplexes, for a specific choice of initial simplex and a specific function, are shown in Figure 4.2. While the convergence of this method has been proven, it is not guaranteed that the method will always converge to a global optimum [59]. It should be noted that the Nelder-Mead method was primarily developed for unconstrained optimization problems, but it can be adapted for constrained optimization problems [59].

[2]A simplex in $\mathbb{R}^n$ is defined as a bounded convex polytope (a generalization of a polyhedron to any dimension) with a non-empty interior and exactly $n + 1$ vertices [59].

[3]The term "simplex method" more commonly refers to the algorithm used to find the optimal solution in linear programming. This algorithm was developed by George Dantzig [65].

Algorithm 1 Nelder-Mead algorithm

```
Require: Initial simplex $S^0 = \{s^0, s^1, \ldots, s^n\}$, function $f : \mathbb{R}^n \to \mathbb{R}$, parameters $\delta^e$, $\delta^{oc}$, $\delta^{ic}$, $\gamma$, iteration
    counter $k \leftarrow 0$
Ensure: Approximate optimal solution $\boldsymbol{x}^\star$ for function $f$
 1: procedure NELDER-MEAD($S^0$)
 2:     Reorder $S^k$ so that $f(s^0) \leq f(s^1) \leq \cdots \leq f(s^n)$
 3:     Set $f^k_{\text{best}} = f(s^0)$
 4:     while stopping condition not met do
Reflection
 5:         Compute centroid $x^c = \frac{1}{n} \sum_{i=0}^{n-1} s^i$
 6:         Set reflection point $x^r = x^c + (x^c - s^n)$ and compute $f^r = f(x^r)$
 7:         if $f^k_{\text{best}} \leq f^r < f(s^{n-1})$ then
 8:             Set $S^{k+1} = \{s^0, s^1, \ldots, s^{n-1}, x^r\}$
 9:             Increment $k \leftarrow k + 1$ and continue
10:         end if
Expansion
11:         if $f^r < f^k_{\text{best}}$ then
12:             Set expansion point $x^e = x^c + \delta^e(x^r - x^c)$ and compute $f^e = f(x^e)$
13:             if $f^e < f^r$ then
14:                 Set $S^{k+1} = \{s^0, s^1, \ldots, s^{n-1}, x^e\}$
15:             else
16:                 Set $S^{k+1} = \{s^0, s^1, \ldots, s^{n-1}, x^r\}$
17:             end if
18:             Increment $k \leftarrow k + 1$ and continue
19:         end if
Contraction
20:         if $f^r \geq f(s^n)$ then
21:             Outside Contraction: Compute $x^{oc} = x^c + \delta^{oc}(x^c - s^n)$ and $f^{oc} = f(x^{oc})$
22:             if $f^{oc} < f(s^n)$ then
23:                 Set $S^{k+1} = \{s^0, s^1, \ldots, s^{n-1}, x^{oc}\}$
24:             else
25:                 Shrink: Set $S^{k+1} = \{s^0, s^0 + \gamma(s^1 - s^0), \ldots, s^0 + \gamma(s^n - s^0)\}$
26:             end if
27:             Increment $k \leftarrow k + 1$ and continue
28:         else
29:             Inside Contraction: Compute $x^{ic} = x^c + \delta^{ic}(x^c - s^n)$ and $f^{ic} = f(x^{ic})$
30:             if $f^{ic} < f(s^n)$ then
31:                 Set $S^{k+1} = \{s^0, s^1, \ldots, s^{n-1}, x^{ic}\}$
32:             else
33:                 Set $S^{k+1} = \{s^0, s^0 + \gamma(s^1 - s^0), \ldots, s^0 + \gamma(s^n - s^0)\}$
34:             end if
35:             Increment $k \leftarrow k + 1$ and continue
36:         end if
37:     end while
38: end procedure
```

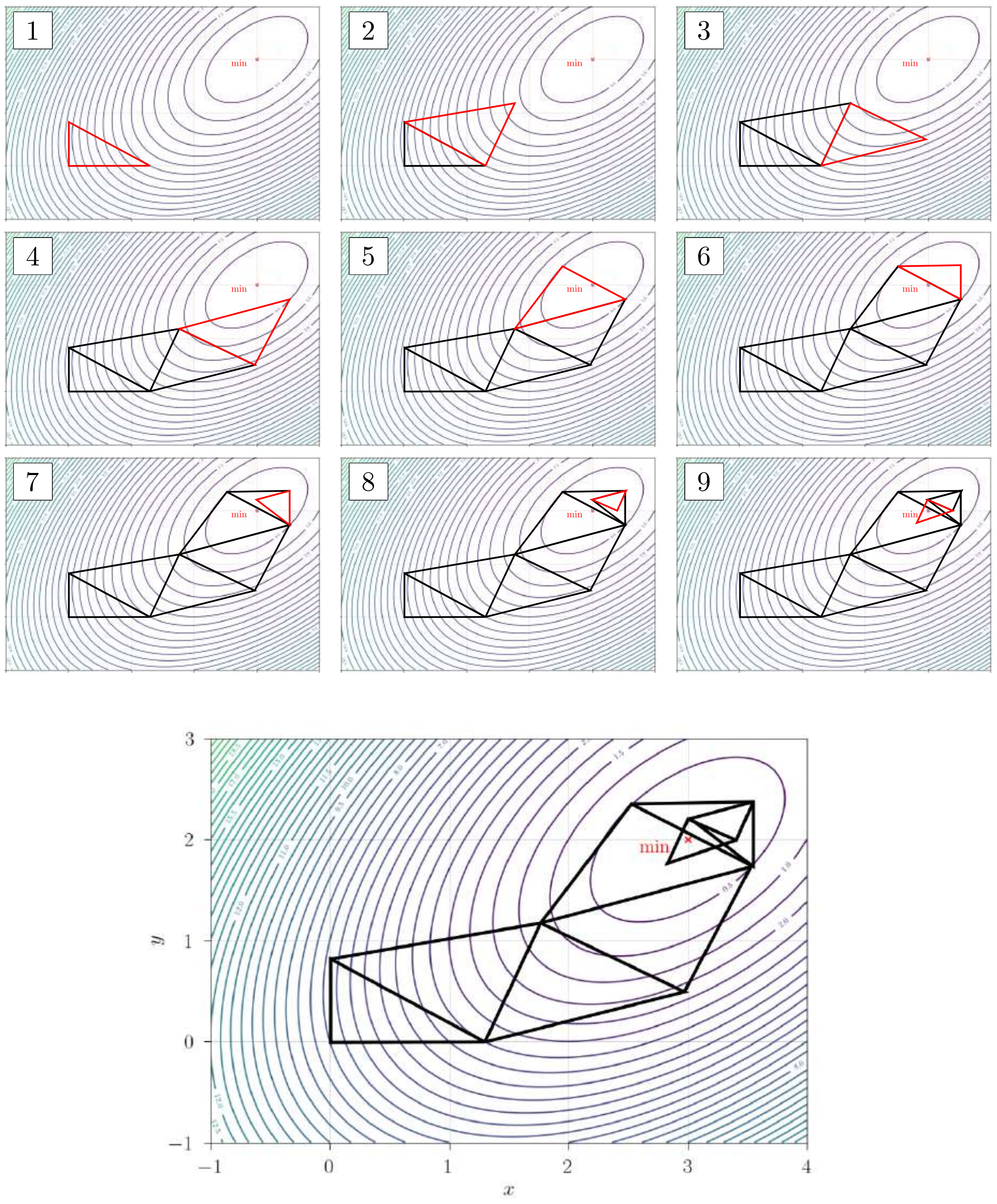


Figure 4.2: Several iterations of the Nelder-Mead method for a specific choice of the initial simplex when minimizing the function $x^2 - 4x + y^2 - y - xy + 7$, with the minimum at the point (3,2) marked by a red cross.

4.2.2 Direct search methods

In this part, we describe the *generalized pattern search* (hereafter GPS) method [67] and the *mesh adaptive direct search* (hereafter MADS) method [68].

To describe the GPS algorithm, it is necessary to define a mesh, which is used to describe the search sets within the GPS algorithm. Let $\mathbf{G} \in \mathbb{R}^{n\times n}$ be invertible and $\mathbf{Z} \in \mathbb{Z}^{n\times p}$, $n, p \in \mathbb{N}$. Assume that every vector from $\mathbb{R}^n$ can be expressed as a linear combination of the columns of matrix $\mathbf{Z}$ (treated as vectors), such that all the coefficients in this linear combination are non-negative. Furthermore, let $\mathbf{D} = \mathbf{GZ}$. The mesh $\mathbf{M}$ generated by $\mathbf{D}$ centered at point $\boldsymbol{x}$ is defined as

$$\mathbf{M} = \{\boldsymbol{x} + \delta\,\mathbf{D}y \,|\, y \in \mathbb{N}^p\}\,, \tag{4.4}$$

where δ is called the mesh size parameter [59, 67]. In each iteration of the GPS algorithm, the shape of the mesh generally changes, as it is always centered at the point representing the best estimate in that iteration, and the size of the mesh step also changes. Let $\boldsymbol{x}_k$ and δ_k represent the estimate of the solution and the mesh size in the k-th iteration, respectively. We then define the mesh in the k-th iteration, denoted as $\mathbf{M}_k$, as

$$\mathbf{M}_k = \{\boldsymbol{x}_k + \delta_k\,\mathbf{D}y \,|\, y \in \mathbb{N}^p\}\,. \tag{4.5}$$

Note that the columns of matrix $\mathbf{D}$, as defined above, can be interpreted as the possible directions in which the GPS algorithm searches the optimization space [59, 67]. Examples of search directions and meshes generated by different matrices $\mathbf{G}$ and $\mathbf{Z}$ are presented in Figure 4.3.

$$\mathbf{G} = \mathbf{I}, \quad \mathbf{Z} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} \sqrt{3} & 0 \\ -1 & 2 \end{pmatrix}, \quad \mathbf{Z} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \end{pmatrix}$$

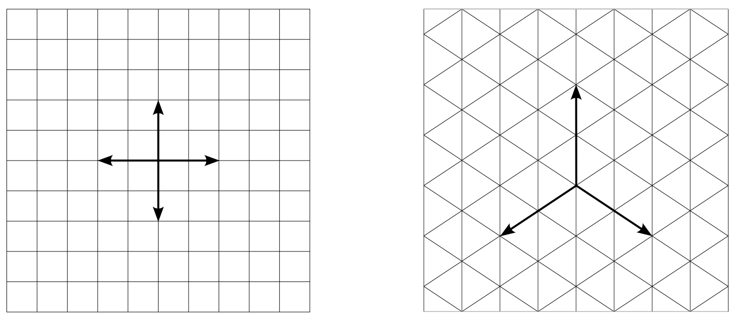


Figure 4.3: Examples of search directions and meshes in $\mathbb{R}^2$ with obtained by different choices of $\mathbf{G}$ and $\mathbf{Z}$. The mesh points are at the intersections of the lines, the arrows represent possible search directions.

After initializing the necessary starting parameters, each iteration of the GPS algorithm is divided into two main steps. The first step is called the search step. During the search step, a finite set S_k of candidate mesh points, selected according to a strategy specified by the user, is evaluated by computing the objective function at each one of the points. If none of the evaluated points represents an improvement over the value $f(\boldsymbol{x}_k)$, the poll step follows. In the poll step, the objective function is evaluated at all neighboring mesh points of $\boldsymbol{x}_k$. The poll set in k-th iteration is defined as $P_k = \{x_k + \delta_k d \,|\, d \in \mathbf{D}\}$. If none of the evaluated points represents an improvement over the value $f(\boldsymbol{x}_k)$, we set $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ and decrease the mesh size, i.e., $\delta_{k+1} < \delta_k$. However, if a point that improves the estimate of the solution is found in either the search or the poll step, this point is set as $\boldsymbol{x}_{k+1}$, and the mesh size is increased, i.e., $\delta_{k+1} > \delta_k$ [59, 67].

The changes described above define a new mesh $\mathbf{M}_k$ in each iteration. The mesh changes throughout the GPS algorithm. The algorithm terminates when $\delta_{k+1} < \varepsilon$ for some user-specified $\varepsilon > 0$. It can be shown that the mesh step converges to zero, and under appropriate assumptions, the solution estimates converge to a stationary point of the objective function. Details can be found in [59]. It should be noted that the convergence of GPS has been proven for unconstrained problems [59]. The full GPS algorithm is presented in Algorithm 2.

Algorithm 2 Generalized Pattern Search (GPS) for unconstrained optimization

```
Require: Function $f : \mathbb{R}^n \to \mathbb{R}$, initial point $x_0$, initial mesh size parameter $\delta_0$, positive spanning matrix
    $\mathbf{D}$, mesh size adjustment parameter $\tau \in (0; 1)$, stopping tolerance $\epsilon_{\text{stop}}$, iteration counter $k \leftarrow 0$
Ensure: Approximate optimal solution $\boldsymbol{x}^\star$ for function $f$
 1: procedure GPS($x_0$)
 2:     while $\delta_k > \epsilon_{\text{stop}}$ do
1. Search
 3:         Define a finite subset $S_k$ of the mesh $\mathbf{M}_k$
 4:         if $f(t) < f(x_k)$ for some $t \in S_k$ then
 5:             Set $x_{k+1} \leftarrow t$ and $\delta_{k+1} \leftarrow \tau^{-1}\delta_k$
 6:             continue
 7:         else
 8:             Go to Poll step
 9:         end if
2. Poll
10:         Select a positive spanning set $\mathbf{D}_k \subseteq \mathbf{D}$
11:         Define $P_k = \{x_k + \delta_k d : d \in \mathbf{D}_k\}$
12:         if $f(t) < f(x_k)$ for some $t \in P_k$ then
13:             Set $x_{k+1} \leftarrow t$ and $\delta_{k+1} \leftarrow \tau^{-1}\delta_k$
14:         else
15:             $x_k$ is a mesh local optimizer
16:             Set $x_{k+1} \leftarrow x_k$ and $\delta_{k+1} \leftarrow \tau\delta_k$
17:         end if
3. Termination
18:         if $\delta_{k+1} \leq \epsilon_{\text{stop}}$ then
19:             terminate
20:         else
21:             Increment $k \leftarrow k + 1$ and continue
22:         end if
23:     end while
24: end procedure
```

We turn our attention to the MADS algorithm, which generalizes the GPS algorithm by allowing for a different set of polling directions. The key difference is that MADS introduces the concept of a frame, which allows the poll directions to form a dense subset in $\mathbb{R}^n$ as the algorithm progresses [59, 62]. The frame $\mathbf{F}_k$ at iteration k is defined as

$$\mathbf{F}_k = \{\boldsymbol{x} \in \mathbf{M}_k \mid \|\boldsymbol{x} - \boldsymbol{x}_k\|_\infty \leq \Delta_k b\}, \tag{4.6}$$

where M_k represents the current mesh, Δ_k is the frame size parameter, which satisfies $\delta_k \leq \Delta_k$, and $b = \max\{\|d\|_\infty \mid d \in \mathbf{D}\}$. The extent of the frame is determined by the parameter Δ_k, and the polling directions are taken from this frame.

Each iteration of the MADS algorithm begins similarly to GPS, with a search step where a finite set S_k of candidate points, selected based on the current mesh, is evaluated by computing the objective function at each of the points. If none of these points improves upon the current best solution, the algorithm proceeds to the poll step.

The poll set P_k is a subset of points selected from $\mathbf{F}_k$ and $\mathbf{M}_k$. Trivially, $P_k \subseteq \mathbf{F}_k \subseteq \mathbf{M}_k$. Importantly, the mesh size parameter δ_k is allowed to decrease more rapidly than the enabling the poll directions to asymptotically become arbitrarily dense [68]. This aspect is crucial, as it ensures that MADS can explore directions in a finer and more systematic manner than GPS, leading to better convergence properties.

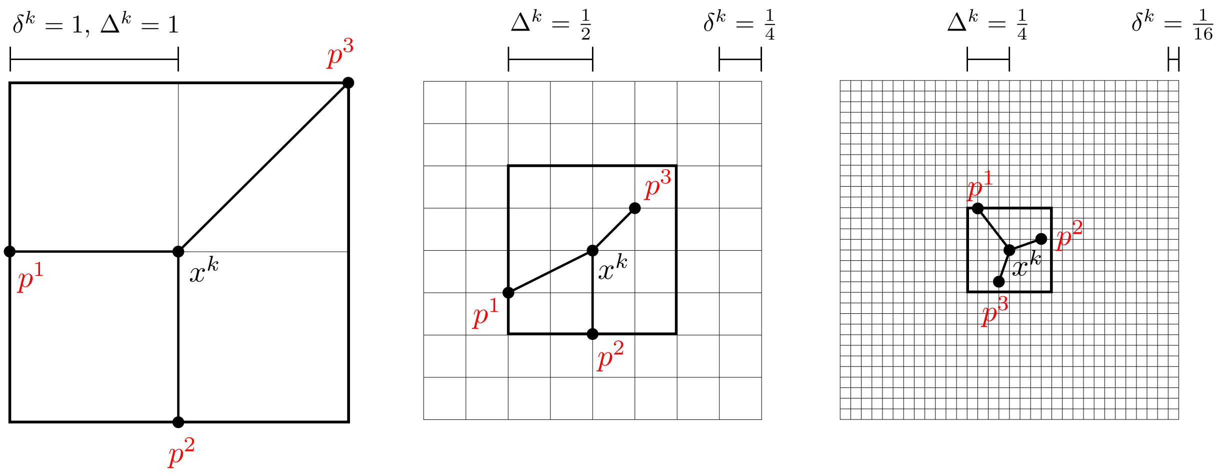


Figure 4.4: Examples of meshes and frames in $\mathbb{R}^2$ for different values of δ_k and Δ_k.

If an improvement is found in the poll step, the new point is set as $\boldsymbol{x}_{k+1}$, and the frame size Δ_k and mesh size δ_k may be increased to encourage further exploration. Conversely, if no improvement is found, $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$, and the mesh size is reduced, i.e., $\delta_{k+1} < \delta_k$, to allow for a more local search. The process of shrinking and refining the frame and mesh sizes continues until δ_k falls below a user-specified threshold, at which point the algorithm terminates.

MADS employs a modified objective function that incorporates an extreme barrier function to handle constraint, similar approach can also be used in GPS. For constrained optimization problems, the objective function is adjusted using an extreme barrier function [59], which assigns an infinite objective value to any infeasible points, effectively penalizing them. The extreme barrier function, denoted as B_∞, represents a special case of barrier functions commonly employed in constrained optimization methods [60]. It is defined by

$$\begin{aligned} B_\infty(\boldsymbol{x}) &= 0, \quad \forall \boldsymbol{x} \in \mathbf{X}, \\ B_\infty(\boldsymbol{x}) &= +\infty, \quad \text{otherwise}. \end{aligned} \tag{4.7}$$

The modified objective function is then given by

$$\begin{aligned} f_\infty(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathbf{X}, \\ f_\infty(\boldsymbol{x}) &= +\infty, \quad \text{otherwise}. \end{aligned} \tag{4.8}$$

Note that, in general, both B_∞ and f_∞ map $\mathbb{R}^n$ to $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. In other words, $B_\infty, f_\infty \to \overline{\mathbb{R}}$.

This simple yet effective strategy ensures that the optimization remains focused on feasible regions of the search space, and the algorithm converges to a stationary point even for constrained problems. A full description of the MADS algorithm and its implementation details can be found in [59].

Algorithm 3 Mesh Adaptive Direct Search (MADS) for constrained optimization

```
Require: Function $f_\infty$, initial point, initial frame size parameter $\Delta_0$, positive spanning matrix $\mathbf{D}$, mesh
    size adjustment parameter $\tau \in (0; 1)$, stopping tolerance $\epsilon_{\text{stop}}$, iteration counter $k \leftarrow 0$
Ensure: Approximate optimal solution $\boldsymbol{x}^\star$ for function $f$
 1: procedure MADS($x_0$)
 2:     while $\Delta_k > \epsilon_{\text{stop}}$ do
1. Parameter Update
 3:         Set the mesh size parameter $\delta_k = \min\{\Delta_k, (\Delta_k)^2\}$
2. Search
 4:         Define a finite set $S_k \subset \mathbf{M}_k$ such that:
 5:         if $f_\infty(t) < f_\infty(x_k)$ for some $t \in S_k$ then
 6:             Set $x_{k+1} \leftarrow t$ and $\Delta_{k+1} \leftarrow \tau^{-1}\Delta_k$
 7:             continue
 8:         else
 9:             Go to Poll step
10:         end if
3. Poll
11:         Select a positive spanning set $\mathbf{D}_k$ and define:
12:         $P_k = \{x_k + \delta_k d : d \in \mathbf{D}_k\}$, a subset of the frame $\mathbf{F}_k$ with extent $\Delta_k$
13:         if $f_\infty(t) < f_\infty(x_k)$ for some $t \in P_k$ then
14:             Set $x_{k+1} \leftarrow t$ and $\Delta_{k+1} \leftarrow \tau^{-1}\Delta_k$
15:         else
16:             Set $x_{k+1} \leftarrow x_k$ and $\Delta_{k+1} \leftarrow \tau\Delta_k$
17:         end if
4. Termination
18:         if $\Delta_{k+1} \le \epsilon_{\text{stop}}$ then
19:             terminate
20:         else
21:             Increment $k \leftarrow k + 1$ and continue
22:         end if
23:     end while
24: end procedure
```

4.2.3 Optimization using a surrogate model

In cases where evaluating the objective function at a specific point is time-consuming or computationally expensive, it can be useful to employ a surrogate model for the objective function during optimization. We define a surrogate model of the given problem as the problem

$$\min_{\boldsymbol{x} \in \tilde{\mathbf{X}}} \tilde{f}(\boldsymbol{x}), \tag{4.9}$$

where

$$\tilde{\mathbf{X}} = \left\{ \boldsymbol{x} \in \mathbf{D} \subseteq \mathbb{R}^n \,\middle|\, \tilde{\boldsymbol{g}}(\boldsymbol{x}) \leq \mathbf{0} \wedge \tilde{\boldsymbol{h}}(\boldsymbol{x}) = \mathbf{0} \right\}, \tag{4.10}$$

and the functions $\tilde{f}, \tilde{\boldsymbol{g}}$, and $\tilde{\boldsymbol{h}}$ have (optimization related) properties similar to those of the functions $f, \boldsymbol{g}$, and $\boldsymbol{h}$ in the original problem. The properties of $\tilde{f}, \tilde{\boldsymbol{g}}$, and $\tilde{\boldsymbol{h}}$ are intentionally left undefined, reflecting the fact that the surrogate model does not necessarily need to be an accurate approximation of the original problem [59, 63, 64]. A good approximative model may not always be a suitable surrogate for optimization purposes, a situation illustrated in Figure 4.5.

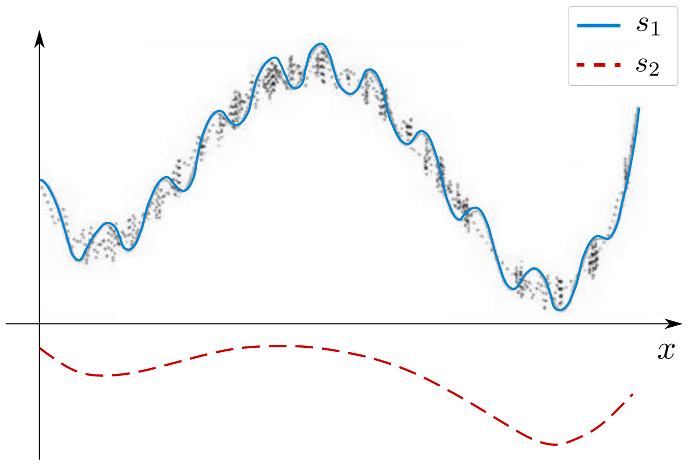


Figure 4.5: An illustration of two surrogate models, s_1 and s_2. The black points represent noisy values of the objective function. While using surrogate model s_1 represents a better choice for approximating the function, it is not suitable for optimization since s_1 contains many undesirable stationary points that the original objective function does not have. On the other hand, while surrogate model s_2 is not as accurate in approximating the function's values, it is a better choice for optimization because the stationary points of s_2 are almost identical to those of the optimized objective function.

Using a surrogate model in optimization is often just a part of a more complex optimization method. Surrogate models can, for example, be used within GPS and MADS methods described in Section 4.2.2, where, during the poll step, we first evaluate the surrogate function $\tilde{f}$ at the points from the poll set, sort these values, and use them to sort the poll set used to evaluate the original function f. This potentially allows us to significantly reduce the time required to complete the poll step, as sorting the points increases the probability of finding a better estimate of the solution at one of the first examined points [59]. Surrogate models can also be used within other methods to accelerate the optimization process, and their application is discussed in detail in [59, 63].

4.3 Description of the optimization framework

A fully automated and modular optimization framework was developed to solve the optimization problems in this work. The proposed framework is composed of several components, which are described in detail below:

1. **Optimization**: The first component encompasses the optimization method, which governs the entire process. In this part, the optimization problem is defined along with any required constraints.

   In this study, we use the Nelder-Mead method, implemented in Python specifically for this work and detailed in Appendix C. Additionally, the framework includes the MADS method (see Section 4.2.2), implemented in the open-source `NOMAD` library [69] in C++. For handling constraints in both the Nelder-Mead and MADS methods, we use the extreme barrier function.

   In previous work [19, 20], we explored the use of gradient-based optimization methods in the context of black-box optimization coupled with LBM simulations. These methods utilized interpolation boundary conditions, such as the Bouzidi condition [70], on the boundaries of obstacles to ensure continuity of the objective function as the geometry changed. It was demonstrated that gradient-free methods are preferable in this context due to their effectiveness and simplicity.

2. **Geometry Generation**: The optimization parameters, updated at each iteration, are passed to the geometry generator. For generating the geometries used in numerical simulations, we use the `meshgen` package detailed in Chapter 3.

3. **Numerical Simulation**: The final component of the optimization framework is the numerical solver, which evaluates the objective function based on the generated geometry. Numerical simulations are performed using the LBM, which is described alongside additional implementation details in Chapter 2.

The interconnection of these components within the optimization framework is schematically illustrated in Figure 4.6.

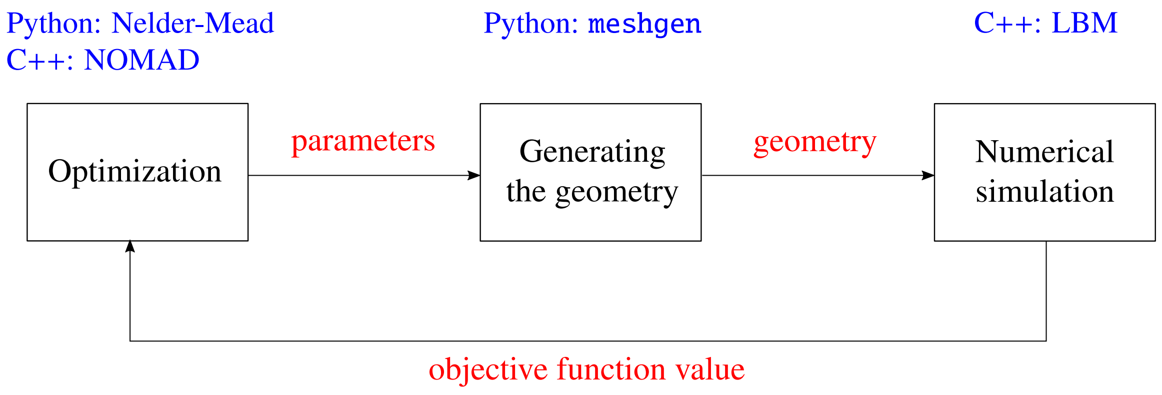


Figure 4.6: Schematic representation of the modular optimization framework. It consists of three main components: (1) Optimization, which defines the problem and governs the iterative process, (2) Geometry generation, where optimization parameters are translated into geometric models using `meshgen`, and (3) Numerical simulation, where the value of the objective function is the output of LBM simulation.

Chapter 5

Studied problems

In this chapter, we analyze two proposed idealized models of the TCPC. These models are designed to incorporate key geometric modifications identified as beneficial in reducing energy dissipation, as described in [6]. These modifications, summarized in Figure 5.1, serve as a foundation for investigating the effects of caval offsetting, curving, flaring, and other geometric factors on flow dynamics.

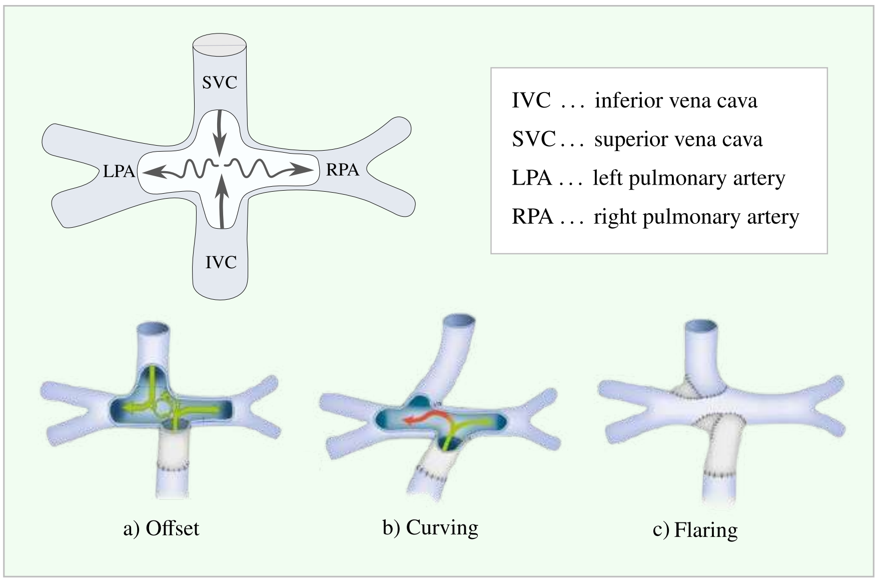

SVC
LPA
RPA
IVC
IVC ... inferior vena cava
SVC ... superior vena cava
LPA ... left pulmonary artery
RPA ... right pulmonary artery
a) Offset
b) Curving
c) Flaring

Figure 5.1: The top portion of the figure illustrates a schematic representation of the TCPC, with abbreviations denoting the corresponding segments. The bottom portion highlights key modifications to improve TCPC geometry and reduce energy dissipation: a) caval offsetting, where the inferior and superior vena cava are misaligned to minimize flow collision; b) curving, where the vessels are curved; and c) flaring, where the connections are widened to reduce sharp corners and flow resistance. Adapted from [6].

5.1 Objective functions and other monitored metrics

Selecting appropriate objective functions is essential for guiding geometric optimization. In this work, we focus primarily on two such functions: the turbulent kinetic energy (TKE) and the shear rate near the walls. In addition, we also present other relevant metrics that are monitored during the simulations.

5.1.1 Turbulent kinetic energy

TKE, defined by Eq. (1.12), measures the intensity of velocity fluctuations within the flow. In the context of the TCPC, higher TKE generally corresponds to increased energy losses [6]. Thus, reducing TKE in the TCPC region is desirable in order to improve hemodynamic efficiency.

To quantify the overall turbulent activity within a specific region of the computational domain, we integrate TKE over a predefined control volume. This integral provides a global metric of turbulence, denoted as T^{A}_{turb} [m^5s^{-2}], where A represents the control volume. The metric is mathematically defined by:

$$T^{A}_{\text{turb}} = \int_{A} T_{\text{turb}} \mathrm{d}V, \tag{5.1}$$

where T_{turb} is the local TKE, and $\mathrm{d}V$ represents an infinitesimal volume element within A.

5.1.2 Shear rate

Shear rate, defined by Eq. (1.8), provides insight into the forces acting on the vessel walls. Excessive shear rate can negatively affect the vascular health [71], making controlling near-wall shear rates crucial for ensuring the long-term patency and physiological compatibility of the TCPC.

However, measuring shear rate directly at the wall in a computational framework – especially using LBM – poses a challenge. The discrete nature of the grid and the no-slip boundary condition mean that the resolution near the wall can affect the computed shear rate and make it sensitive to discretization details. To address this, we introduce a thin layer of thickness $\varepsilon = 1$ mm adjacent to the vessel walls. Within this layer, we sum the shear rate values rather than relying solely on the cells immediately neighboring the walls. This approach was chosen to reduce the sensitivity on grid characteristics while maintaining the physiological relevance of the quantity, ensuring it provides representative measure of local flow behavior. We denote this near-wall shear rate as $\dot{\gamma}^{A}_{\text{nw}}$ [$\text{m}^3\ \text{s}^{-1}$]. The metric is mathematically defined by:

$$\dot{\gamma}^{A}_{\text{nw}} = \int_{A_\varepsilon} \dot{\gamma} \mathrm{d}V, \tag{5.2}$$

where $\dot{\gamma}$ is the local shear rate, and A_ε denotes the layer of thickness ε adjacent to the vessel walls.

5.1.3 IVC and SVC flow split

In addition to the main objective functions, we monitor how fluid from the IVC and SVC divides between the LPA and RPA (see Figure 5.1 for abbreviations). A balanced or at least minimally maintained split of IVC flow into both pulmonary artery branches is desirable [72]. This metric is evaluated in post-processing using the computed mean velocity field. We denote the percentage of fluid directed to the RPA and LPA by F^{RPA} and F^{LPA}, respectively, with subscripts IVC and SVC specifying the flow source.

5.1.4 Mean velocity vector angle at the pulmonary artery outlets

It is beneficial for blood in the pulmonary arteries to flow aligned with their axes [6]. Thus, we examine the angle between the mean velocity vector and the outlet normal vector. This angle provides the information for studying the laminarity of the flow.

5.2 Models

To systematically evaluate the effects of geometric changes, the models were developed with varying levels of complexity, incorporating different numbers of degrees of freedom. Both models represent idealized TCPC geometries and share the same labeling convention for inlets and outlets.

The inlets are denoted as $\Gamma_{\text{in}}^{\text{N}}$ (representing the SVC at the top) and $\Gamma_{\text{in}}^{\text{S}}$ (representing the IVC with the conduit at the bottom). Similarly, the outlets are labeled as $\Gamma_{\text{out}}^{\text{W}}$ (representing the LPA) and $\Gamma_{\text{out}}^{\text{E}}$ (representing the RPA). The shared geometric framework and boundary labels, is illustrated in Figure 5.2. The dimensions of cylindrical segments forming the simplified junction along with their corresponding abbreviations are presented in Table 5.1.

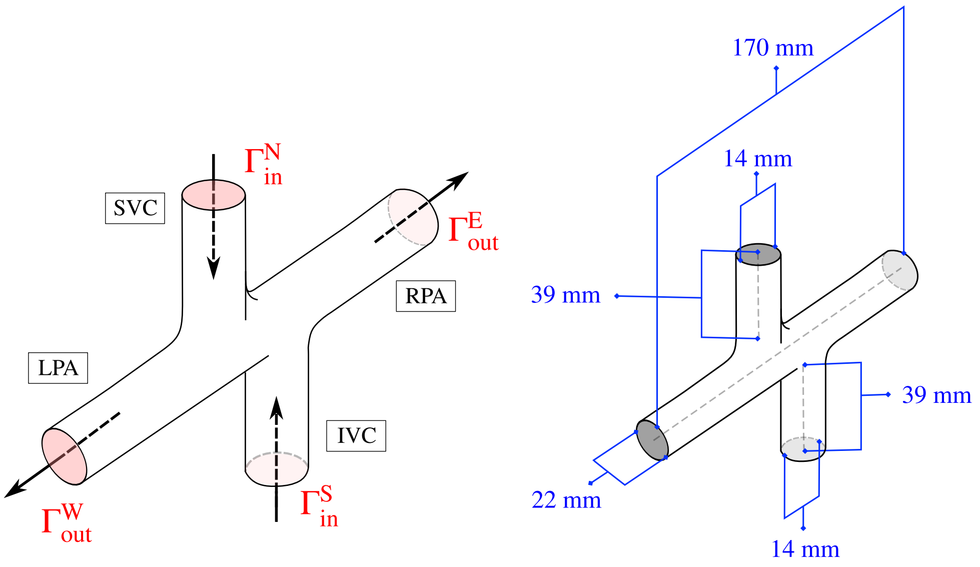

$\Gamma_{\text{in}}^{\text{N}}$
SVC
$\Gamma_{\text{out}}^{\text{E}}$
RPA
LPA
IVC
$\Gamma_{\text{out}}^{\text{W}}$
$\Gamma_{\text{in}}^{\text{S}}$
170 mm
14 mm
39 mm
39 mm
22 mm
14 mm

Figure 5.2: Schematic representation of the idealized TCPC geometry. The labeled inlets ($\Gamma_{\text{in}}^{\text{N}}$, $\Gamma_{\text{in}}^{\text{S}}$) and outlets ($\Gamma_{\text{out}}^{\text{W}}$, $\Gamma_{\text{out}}^{\text{E}}$), directions of inflows and outflows, and the dimensions of the cylindrical segments forming the junction are illustrated.

Segment	Abbreviation	Length	Diameter
Inferior vena cava	IVC	39 mm	14 mm
Superior vena cava	SVC	39 mm	14 mm
Pulmonary artery	PA	170 mm	22 mm

Table 5.1: Dimensions and abbreviations of the cylindrical segments forming the idealized TCPC junction.

In both models, the objective functions are studied within a defined control volume, denoted as A, which is a subset of the entire computational domain. By restricting the analysis to this region, we aim reduce potential errors introduced by the numerical boundary conditions. The control volume involves inlet and outlet boundaries named according to their respective vessel segments: $\Gamma_{\text{in}}^{\text{SVC}}$ for the superior vena cava inlet, $\Gamma_{\text{in}}^{\text{IVC}}$ for the inferior vena cava inlet, $\Gamma_{\text{out}}^{\text{LPA}}$ for the left pulmonary artery outlet, and $\Gamma_{\text{out}}^{\text{RPA}}$ for the right pulmonary artery outlet. The turbulent kinetic energy is evaluated in the entire control volume. The near-wall shear rate is examined along the boundaries of the control volume that are shared with the vessel walls.

Model 1: Simplified cylindrical junction

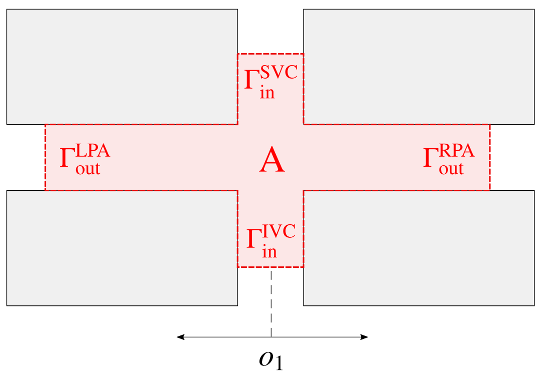
The first model represents a basic cylindrical junction where the IVC and SVC are connected perpendicularly to the PA, forming a cross-like structure, as illustrated in Figure 5.3a. Model 1 was chosen because it introduces only one degree of freedom, the horizontal offset of the IVC axis relative to the SVC, denoted by o_1. This simplicity makes it possible to sample the optimization space and evaluate the objective functions at the points of the sampling. This provides an opportunity to examine the behavior of the objective functions and verify the result generated by the optimization framework.

Model 2: More complex geometric model

Illustration of Model 2 is presented in Figure 5.3a. The second model introduces additional complexity by incorporating five degrees of freedom:

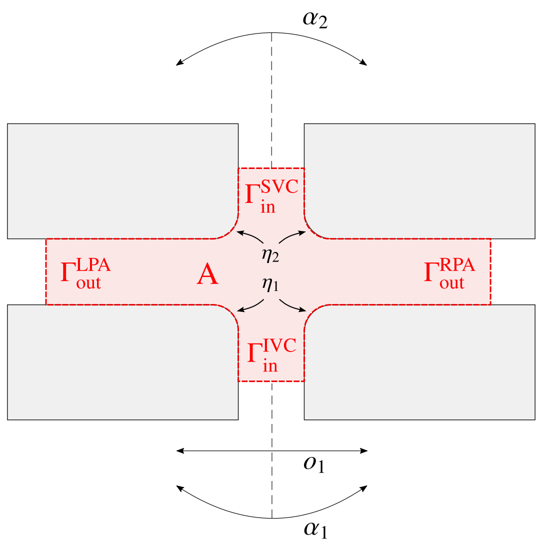
- horizontal offset of the IVC axis, denoted by o_1,
- angle of the IVC axis, denoted by α_1,
- angle of the SVC axis, denoted by α_2,
- width and curvature of the IVC-PA connection, denoted by η_1,
- width and curvature of the SVC-PA connection, denoted by η_2.

Model 2 was selected because its higher complexity allows for more geometric variations.



$\Gamma_{\text{in}}^{\text{SVC}}$
$\Gamma_{\text{out}}^{\text{LPA}}$
A
$\Gamma_{\text{out}}^{\text{RPA}}$
$\Gamma_{\text{in}}^{\text{IVC}}$
o_1

(a) Schematic of Model 1: A simplified cylindrical junction with one degree of freedom, the horizontal offset o_1 of the IVC.



α_2
$\Gamma_{\text{in}}^{\text{SVC}}$
η_2
$\Gamma_{\text{out}}^{\text{LPA}}$
A
η_1
$\Gamma_{\text{out}}^{\text{RPA}}$
$\Gamma_{\text{in}}^{\text{IVC}}$
o_1
α_1

(b) Schematic of Model 2: A more complex model introducing five degrees of freedom: IVC offset o_1, connection angles (α_1, α_2) and flaring (η_1, η_2).

Figure 5.3: Schematic illustrations of Model 1 and Model 2.

5.3 Problem with one optimization parameter

In this section, we focus on a simplified model of the TCPC in a form of a basic cylindrical junction to investigate the behavior of key hemodynamic quantities under varying caval offset. The problem setup is defined in Problem Setup 1. The dimensions of the domain, the value of the kinematic viscosity, and the value of the inflows are chosen to mimic real values occurring in physiological conditions [6].

Problem setup 1: Simplified cylindrical junction

Physical setup:

- Domain: $\Omega = (0; 170\text{ mm}) \times (0; 22\text{ mm}) \times (0; 100\text{ mm})$.
- Kinematic viscosity: $\nu = 3 \cdot 10^{-6}\text{ m}^2\text{ s}^{-1}$.
- Inflow at $\Gamma_{\text{in}}^{\text{IVC}}$: a constant velocity vector $\boldsymbol{u} = (u_1, u_2, u_3)^T$, where $u_1 = u_2 = 0\text{ ms}^{-1}$, and $u_3 = 0.45\text{ ms}^{-1}$. The inflow velocity vector is aligned with the IVC axis in the positive x_3-direction.
- Inflow at $\Gamma_{\text{in}}^{\text{SVC}}$: a constant velocity vector $\boldsymbol{u} = (u_1, u_2, u_3)^T$, where $u_1 = u_2 = 0\text{ ms}^{-1}$, and $u_3 = -0.35\text{ ms}^{-1}$. The inflow velocity vector is aligned with the SVC axis in the negative x_3-direction.

LBM setup:

- Initial condition on $\overline{\overline{\Omega}}$ set as described in Section 2.3.1.
- Boundary conditions at $\Gamma_{\text{out}}^{\text{W}}$ and $\Gamma_{\text{out}}^{\text{E}}$ set according to Section 2.3.2.
- Discretization: $N_1 = 768,\ N_2 = 105, N_3 = 453$ (total of 36529920 lattice sites).
- Kinematic viscosity in lattice units: $\nu^L = 10^{-3}$.

5.3.1 Analysis of the system

The main objective of this study is to minimize two studied quantities, $\dot{\gamma}_{\text{nw}}^A$ and T_{turb}^A. To study the system's behavior and to validate the result generated by the optimization framework, we systematically sample the optimization space prior to running the optimization algorithm.

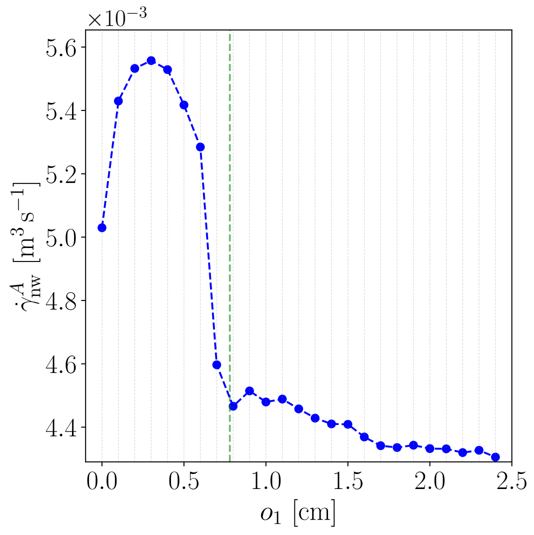
Specifically, we discretize the o_1-parameter space with regular intervals of χ, with $\chi = 0.1$ cm, over the range [0 cm; 2.4 cm], creating a set of sampling points

$$P = \left\{ \mathrm{P}_i \,\middle|\, i = 0, 1, 2, \ldots, 24 \right\}, \tag{5.3}$$

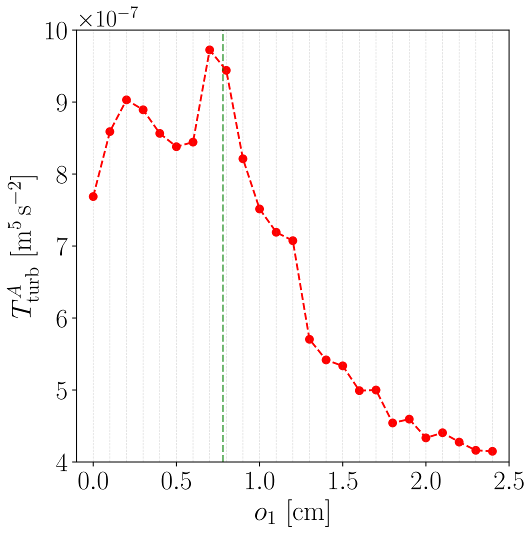
where $P_i = i \cdot \chi$. At each point of P, the system's response is evaluated by computing different flow characteristics. This study serves two purposes:

- It provides insight into the system's behavior. Studying different geometrical setups helps with understanding of how flow characteristics vary with the changing value of parameter o_1.
- By analyzing the sampled data, we can validate the performance of the optimization framework.

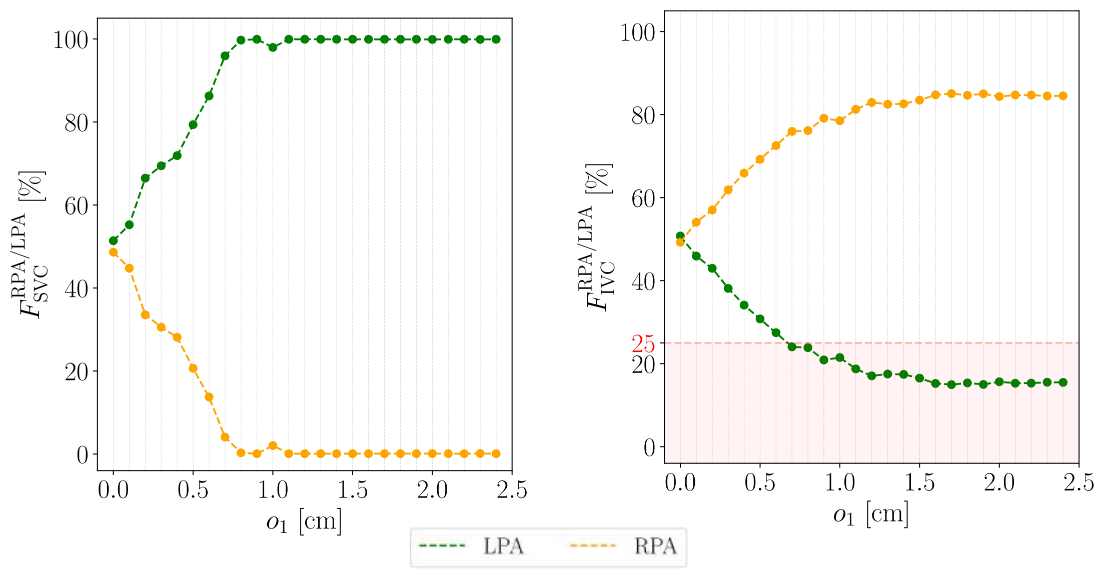
The results of the functions evaluations at the points of sampling are presented in Figure 5.4.



(a) Near-wall shear rate ($\dot{\gamma}^A_{\mathrm{nw}}$), with the green region indicating flow from the IVC exceeding 25%.



(b) Turbulent kinetic energy (T^A_{turb}), with the green region indicating flow from the IVC exceeding 25%.



(c) Percentage of flow directed from the SVC to the LPA (green dashed line) and RPA (orange dashed line).

(d) Percentage of flow directed from the IVC to the LPA and RPA. The red region highlights offsets where the flow fraction to the LPA is below 25%.

Figure 5.4: Summary of key analyzed metrics. (a) Near-wall shear rate ($\dot{\gamma}^A_{\mathrm{nw}}$) in lattice units, with green region indicating offsets for which IVC-LPA flow exceeds 25%; (b) Turbulent kinetic energy (T^A_{turb}) in lattice units, with similar IVC-LPA flow threshold; (c) Flow distribution from the SVC to the LPA and the RPA; (d) Flow distribution from the IVC to the LPA and RPA, with the red region indicating offsets where IVC-LPA flow is below 25%.

Figure 5.4a shows the near-wall shear rate $\dot{\gamma}^A_{\mathrm{nw}}$ as a function of o_1. At $o_1 = 0.0\,\mathrm{cm}$, where the geometry is perfectly symmetric, the shear rate is comparatively lower than it is for slightly larger offsets. Notably, we can observe sharp decrease to a local minimum at $o_1 = 0.7\,\mathrm{cm}$, i.e., an offset almost equal to 0.5-diameter of the IVC. Subsequently, the shear rate exhibits a slight increase and then steadily declines to minimal values at higher offsets–around 1.5-diameter of the IVC.

The turbulent kinetic energy T^A_{turb} is shown in Figure 5.4b. At $o_1 = 0.0\,\mathrm{cm}$, the value is again lower compared to slightly larger offsets. The peak of the TKE occurs at $o_1 = 0.7\,\mathrm{cm}$, i.e., at the offset equal to 0.5-diameter of the IVC. The values then steadily decrease with increasing offset.

Figure 5.4c illustrates the percentage of flow from the SVC directed into the LPA and the RPA. At $o_1 = 0.0\,\mathrm{cm}$, the flow splits evenly between the LPA and RPA, reflecting the symmetry of the geometry. As the offset increases, the flow to the LPA steadily rises up to 100 %, meaning the SVC blood flows exclusively toward the LPA. Conversely, the flow to the RPA steadily falls to 0 %.

The flow split from the IVC is presented in Figure 5.4d. At $o_1 = 0.0\,\mathrm{cm}$, the flow splits evenly between the LPA and RPA. As the offset increases, the flow to the RPA steadily increases up to around 85 %, where it eventually levels. As it was already described in Section 5.1, a balanced or at least minimally maintained split of IVC blood to each pulmonary artery branch is desirable. A threshold of minimal acceptable percentage directed to either one of the parts of the pulmonary artery was set at 25 %. The red shaded region in Figure 5.4d highlights offsets where the IVC flow fraction falls below 25 %. This also determines a offsets considered as acceptable, i.e., those satisfying the aforementioned minimal IVC flow split. Such offsets are highlighted by the green shaded regions in Figure 5.4a and Figure 5.4b.

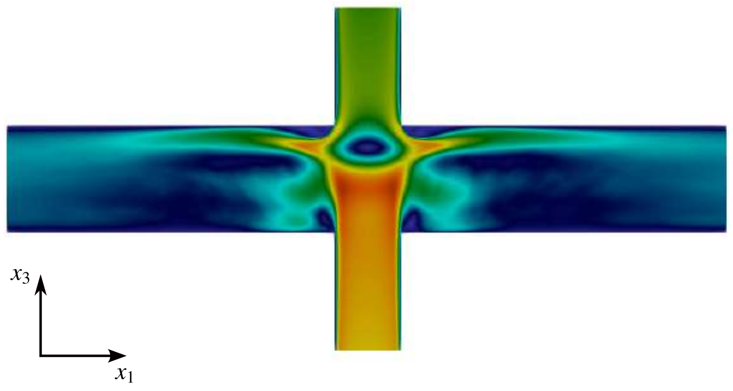
To deepen our understanding of the system's behavior, Table 5.2 details four specific points of interest for which additional metrics and flow characteristics were examined. These points–P_0, P_7, P_8, P_{24}–represent notable points at the functions' behavior. The reason for why each one of the points were chosen for further examination are also detailed in Table 5.2.

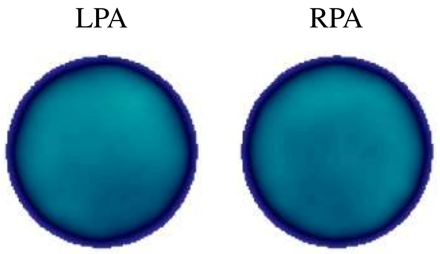
Point	o_1 **[cm]**	**Pages**	**Reason for detailed investigation**
P_0	0.0	49–50	Symmetrical geometry; local minima in both $\dot{\gamma}^A_{\mathrm{nw}}$ and T^A_{turb}.
P_7	0.7	51–52	Offset associated with the maximum T^A_{turb}.
P_8	0.8	53–54	Offset associated with a local minimum in $\dot{\gamma}^A_{\mathrm{nw}}$.
P_{24}	2.4	55–56	Large offset associated with minimal values of both $\dot{\gamma}^A_{\mathrm{nw}}$ and T^A_{turb}.

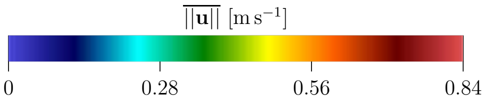
Table 5.2: Key offset points selected for detailed analysis. Each point corresponds to notable extrema or special configurations of $\dot{\gamma}^A_{\mathrm{nw}}$ or T^A_{turb}.

Point P_0: $o_1 = 0.0\,\mathrm{cm}$

Mean velocities

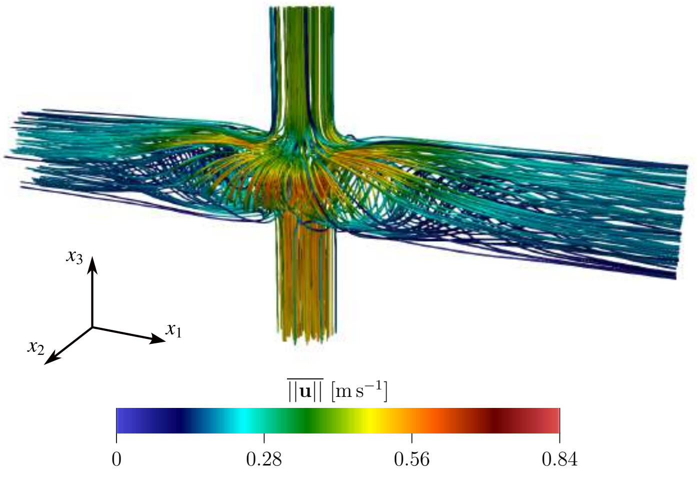




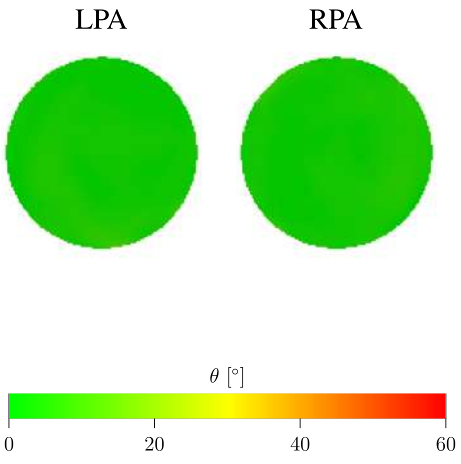


(a) Mean velocity magnitude field in the x_1-x_3 plane.

(b) Magnitudes of mean velocity at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.



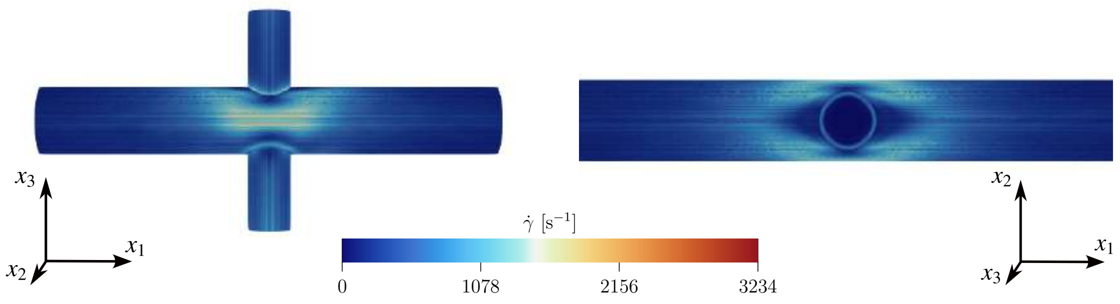
(c) Streamlines of the mean velocity field illustrating the flow trajectory throughout the geometry.



(d) Angle between the mean velocity vector and the normal direction at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.

Figure 5.5: Overview of mean velocity fields and outlet characteristics for the case of point P_0, i.e., when $o_1 = 0.0\,\mathrm{cm}$. Subfigure (a) shows the velocity magnitude field, (b) displays outlet-specific velocity magnitudes, (c) presents streamlines of the mean velocity field, and (d) details the angular alignment of velocity vectors with outlet normals.

Shear rate

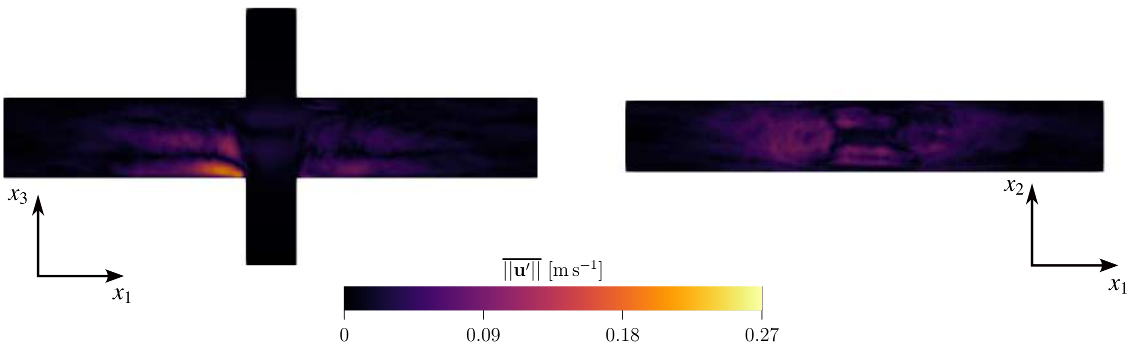


(a) Mean shear rate in the x_1-x_3 plane.

(b) Mean shear rate in the x_1-x_2 plane.

Figure 5.6: Mean shear rate visualizations in the x_1-x_3 and in the x_1-x_2 plane for the case of point P_0, i.e., when $o_1 = 0.0$ cm.

Velocity fluctuations

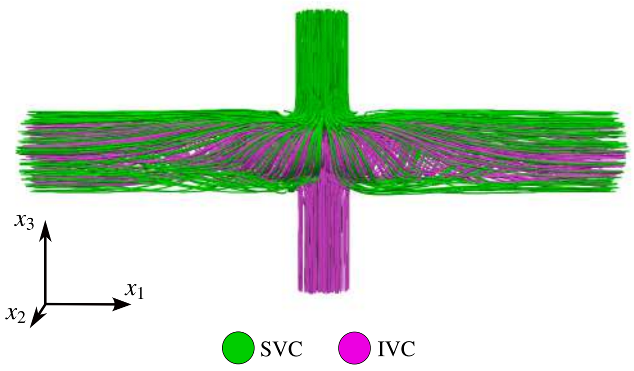


(a) Mean velocity fluctuations magnitude field in the x_1-x_3 plane.

(b) Mean velocity fluctuations magnitude field in the x_1-x_2 plane.

Figure 5.7: Mean velocity fluctuations in the x_1-x_3 and in the x_1-x_2 plane for the case when $o_1 = 0.0$ cm.

SVC/IVC split

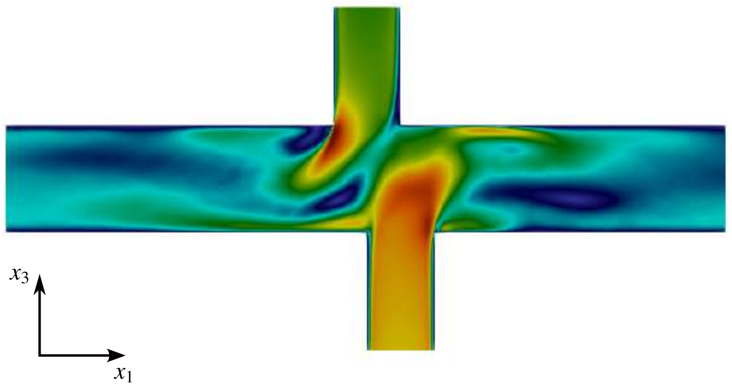


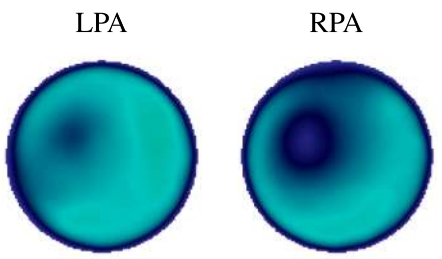
	RPA	LPA
SVC	51.38%	48.62%
IVC	50.76%	49.24%

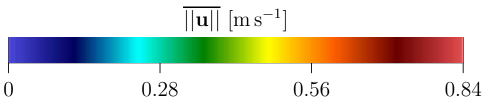
Figure 5.8: Analysis of the split between the SVC and IVC contributions to the LPA and the RPA for the case of point P_0, i.e., when $o_1 = 0.0$ cm. The figure illustrates the flow split, and the table provides the exact percentages.

Point P_7: $o_1 = 0.7\,\text{cm}$

Mean velocities

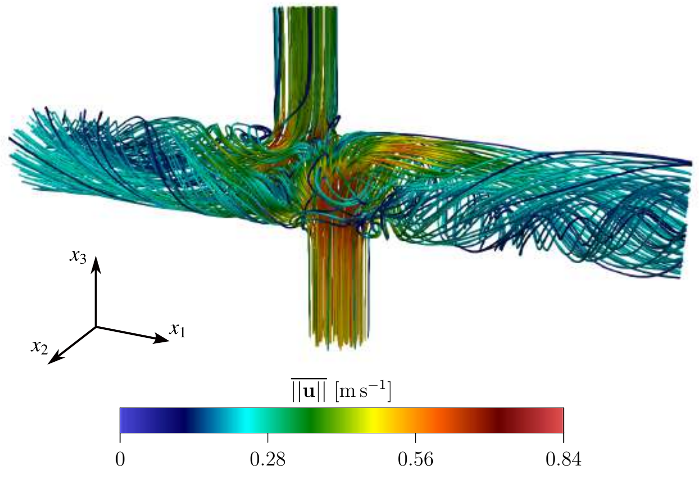




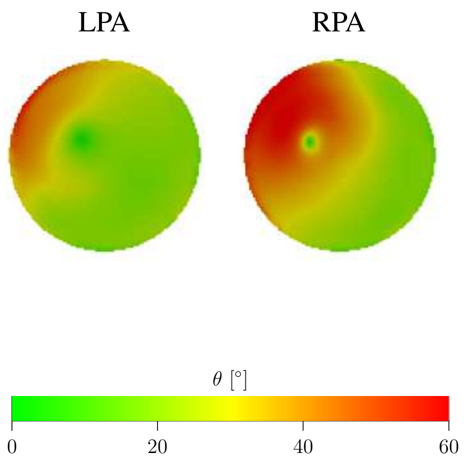


(a) Mean velocity magnitude field in the x_1-x_3 plane.

(b) Magnitudes of mean velocity at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.



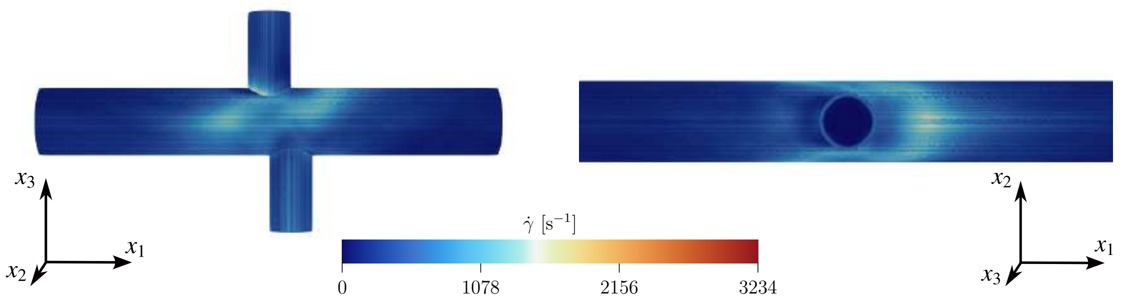
(c) Streamlines of the mean velocity field illustrating the flow trajectory throughout the geometry.



(d) Angle between the mean velocity vector and the normal direction at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.

Figure 5.9: Overview of mean velocity fields and outlet characteristics for the case of point P_7, i.e., when $o_1 = 0.7\,\text{cm}$. Subfigure (a) shows the velocity magnitude field, (b) displays outlet-specific velocity magnitudes, (c) presents streamlines of the mean velocity field, and (d) details the angular alignment of velocity vectors with outlet normals.

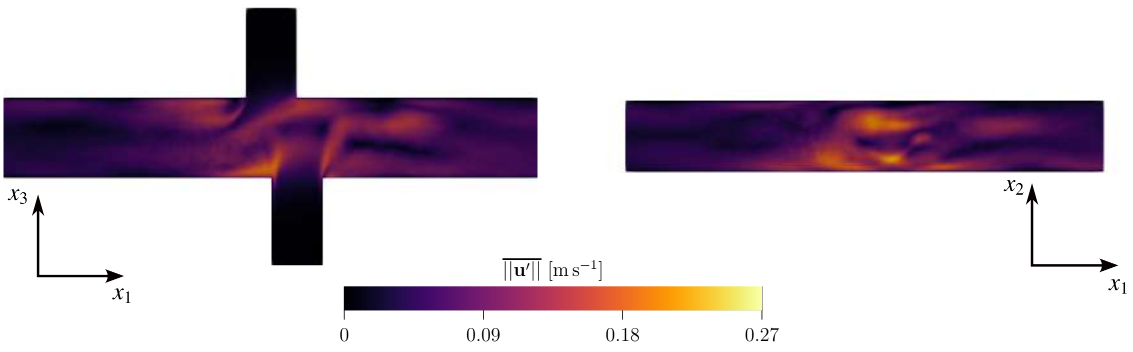
Shear rate



(a) Mean shear rate in the x_1-x_3 plane. (b) Mean shear rate in the x_1-x_2 plane.

Figure 5.10: Mean shear rate visualizations in the x_1-x_3 and in the x_1-x_2 plane for the case of point P_7, i.e., when $o_1 = 0.7$ cm.

Velocity fluctuations

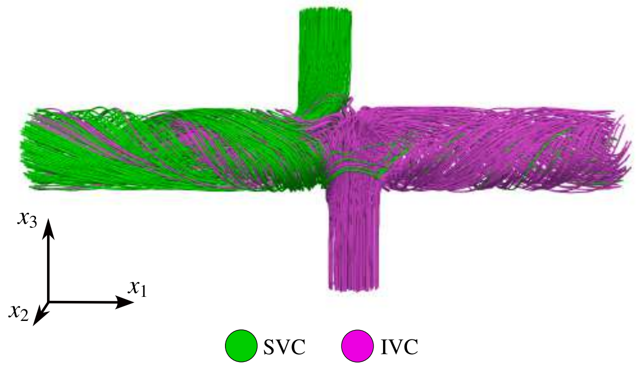


(a) Mean velocity fluctuations magnitude field in the x_1-x_3 plane.

(b) Mean velocity fluctuations magnitude field in the x_1-x_2 plane.

Figure 5.11: Mean velocity fluctuations in the x_1-x_3 and in the x_1-x_2 plane for the case when $o_1 = 0.7$ cm.

SVC/IVC split

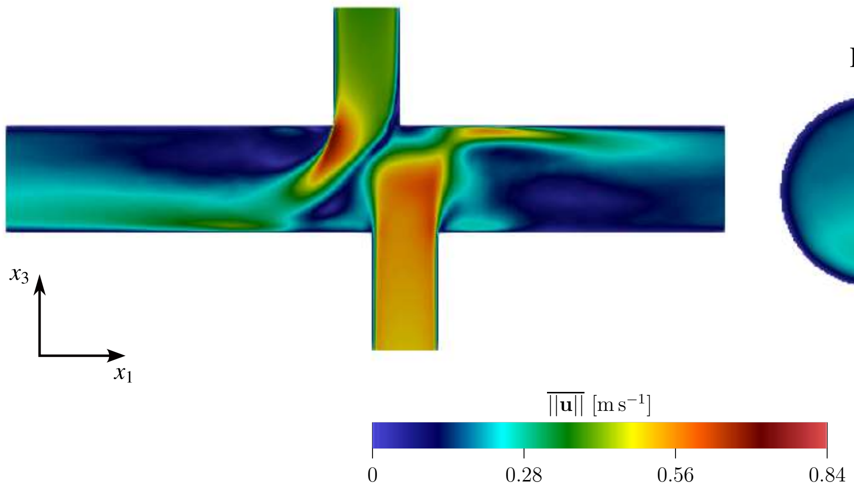


	RPA	LPA
SVC	95.91%	4.09%
IVC	24.05%	75.95%

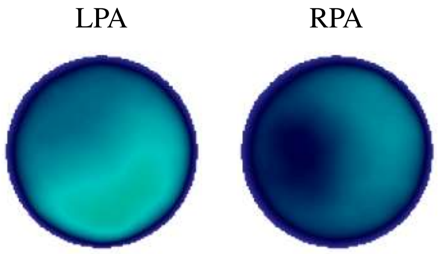
Figure 5.12: Analysis of the split between the SVC and IVC contributions to the LPA and the RPA for the case of point P_7, i.e., when $o_1 = 0.7$ cm. The figure illustrates the flow split, and the table provides the exact percentages.

Point P_8: $o_1 = 0.8\,\text{cm}$

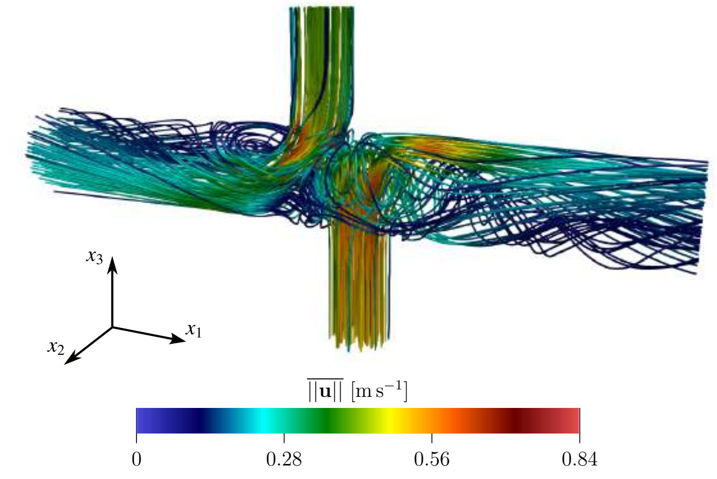
Mean velocities



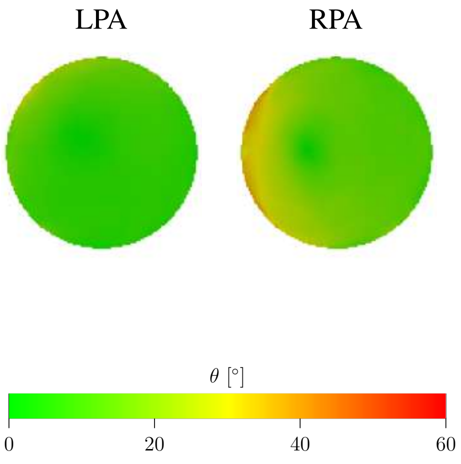
(a) Mean velocity magnitude field in the x_1-x_3 plane.



(b) Magnitudes of mean velocity at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.



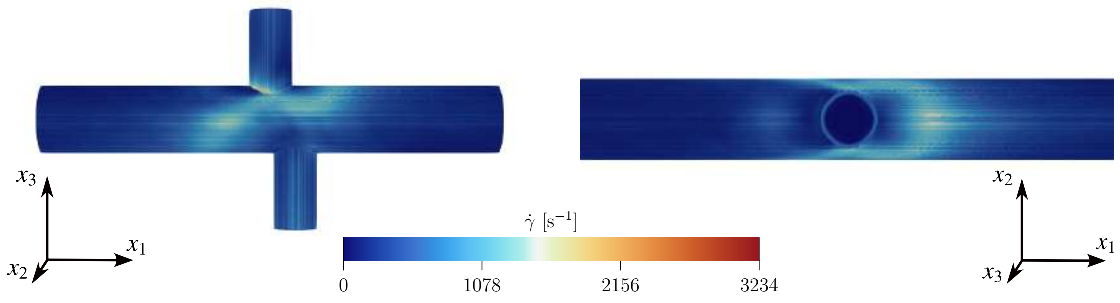
(c) Streamlines of the mean velocity field illustrating the flow trajectory throughout the geometry.



(d) Angle between the mean velocity vector and the normal direction at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.

Figure 5.13: Overview of mean velocity fields and outlet characteristics for the case of point P_8, i.e., when $o_1 = 0.8\,\text{mm}$. Subfigure (a) shows the velocity magnitude field, (b) displays outlet-specific velocity magnitudes, (c) presents streamlines of the mean velocity field, and (d) details the angular alignment of velocity vectors with outlet normals.

Shear rate

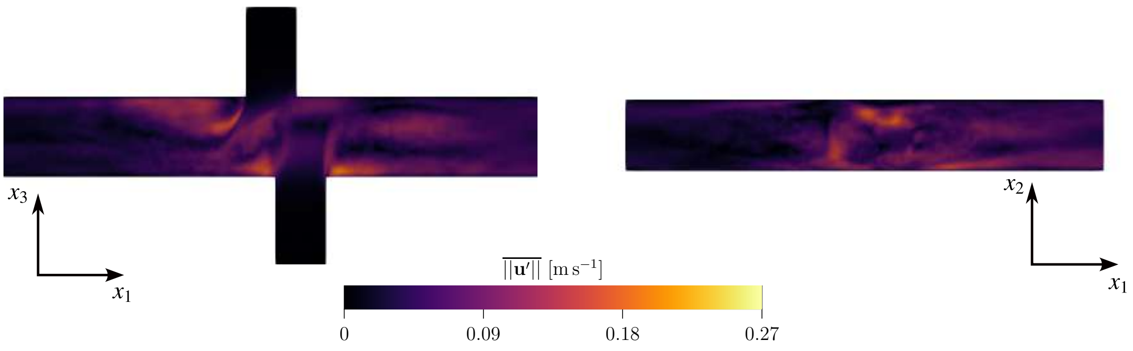


(a) Mean shear rate in the x_1-x_3 plane.

(b) Mean shear rate in the x_1-x_2 plane.

Figure 5.14: Mean shear rate visualizations in the x_1-x_3 and in the x_1-x_2 plane for the case of point P_8, i.e., when $o_1 = 0.8$ mm.

Velocity fluctuations

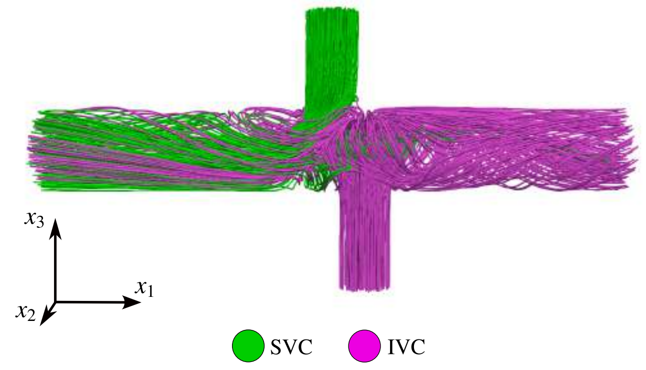


(a) Mean velocity fluctuations magnitude field in the x_1-x_3 plane.

(b) Mean velocity fluctuations magnitude field in the x_1-x_2 plane.

Figure 5.15: Mean velocity fluctuations in the x_1-x_3 and in the x_1-x_2 plane for the case when $o_1 = 0.8$ mm.

SVC/IVC split

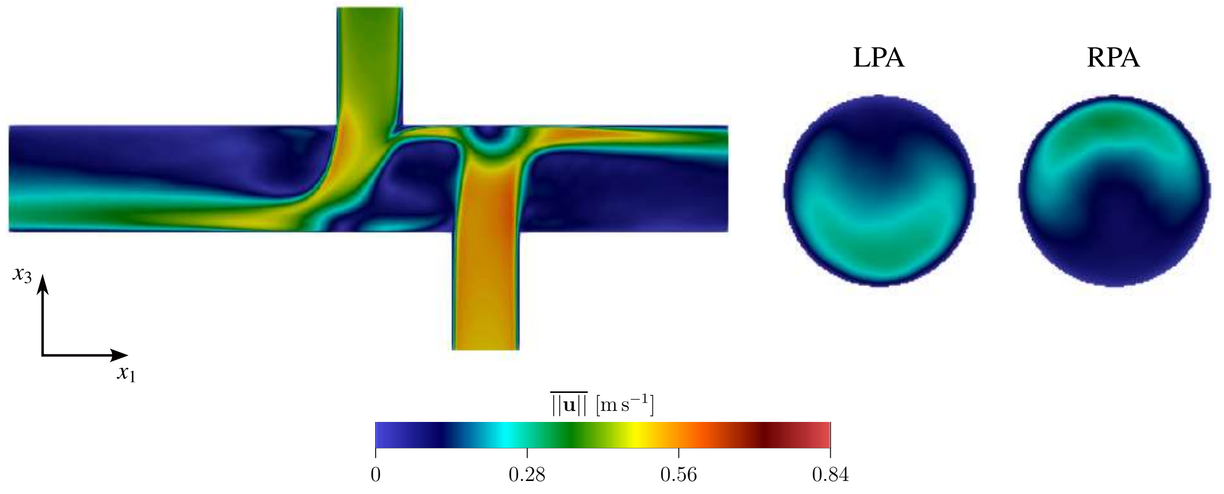


	RPA	LPA
SVC	99.71%	0.29%
IVC	23.85%	76.15%

Figure 5.16: Analysis of the split between the SVC and IVC contributions to the LPA and the RPA for the case of point P_8, i.e., when $o_1 = 0.8$ mm. The figure illustrates the flow split, and the table provides the exact percentages.

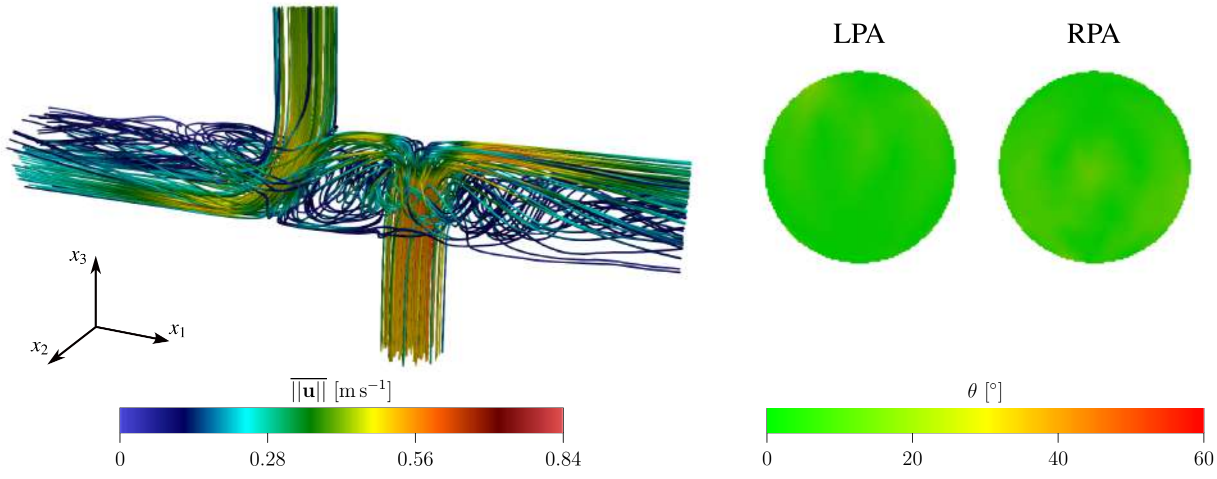
Point P_{24}: $o_1 = 2.4\,\text{cm}$

Mean velocities



(a) Mean velocity magnitude field in the x_1-x_3 plane.

(b) Magnitudes of mean velocity at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.

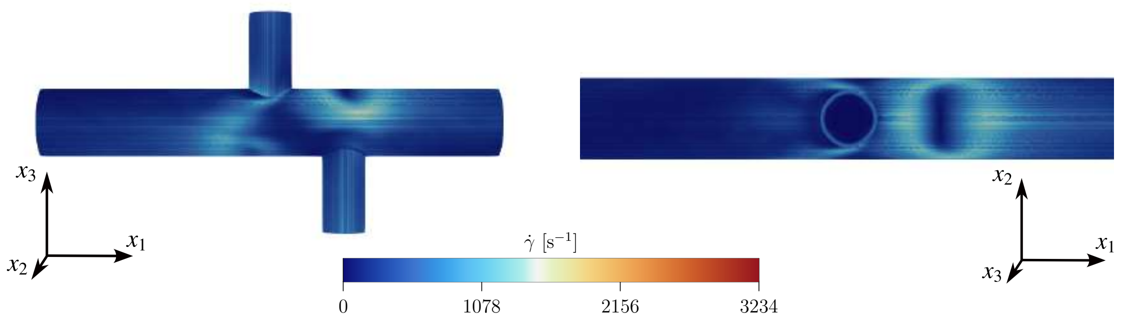


(c) Streamlines of the mean velocity field illustrating the flow trajectory throughout the geometry.

(d) Angle between the mean velocity vector and the normal direction at the outlets $\Gamma_{\text{out}}^{\text{LPA}}$ and $\Gamma_{\text{out}}^{\text{RPA}}$.

Figure 5.17: Overview of mean velocity fields and outlet characteristics for the case of point P_{24}, i.e., when $o_1 = 2.4\,\text{cm}$. Subfigure (a) shows the velocity magnitude field, (b) displays outlet-specific velocity magnitudes, (c) presents streamlines of the mean velocity field, and (d) details the angular alignment of velocity vectors with outlet normals.

Shear rate

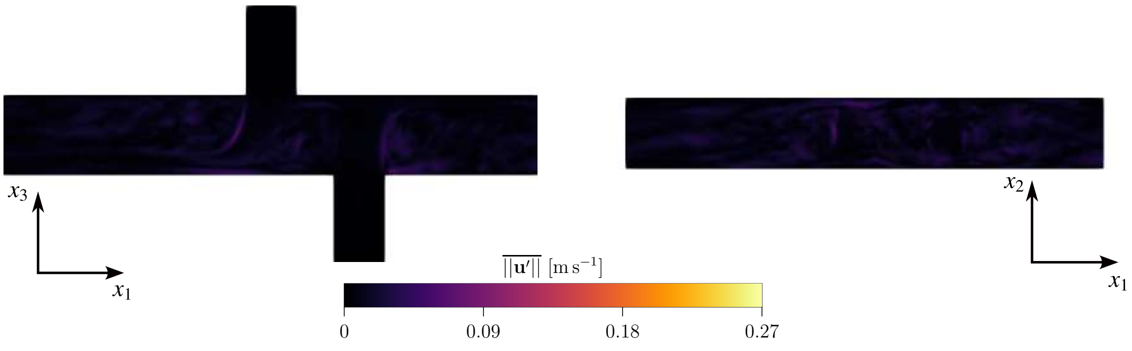


(a) Mean shear rate in the x_1-x_3 plane.

(b) Mean shear rate in the x_1-x_2 plane.

Figure 5.18: Mean shear rate visualizations in the x_1-x_3 and in the x_1-x_2 plane for the case of point P_{24}, i.e., when $o_1 = 2.4$ cm.

Velocity fluctuations

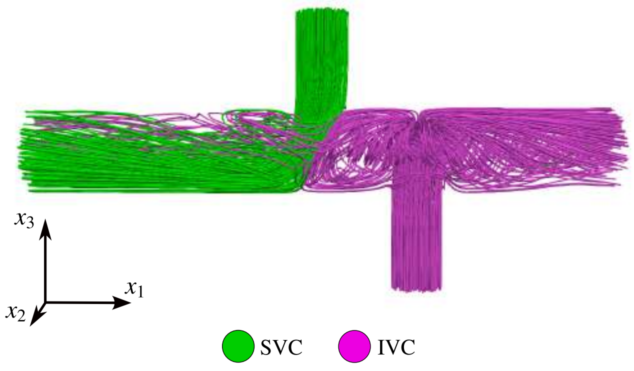


(a) Mean velocity fluctuations magnitude field in the x_1-x_3 plane.

(b) Mean velocity fluctuations magnitude field in the x_1-x_2 plane.

Figure 5.19: Mean velocity fluctuations in the x_1-x_3 and in the x_1-x_2 plane for the case when $o_1 = 2.4$ cm. Note that the color scale is consistent across all investigated points to ensure comparability, which results in lower contrast for this particular case.

SVC/IVC split



	RPA	LPA
SVC	99.92%	0.08%
IVC	15.48%	84.52%

Figure 5.20: Analysis of the split between the SVC and IVC contributions to the LPA and the RPA for the case of point P_{24}, i.e., when $o_1 = 2.4$ cm. The figure illustrates the flow split, and the table provides the exact percentages.

Analysis of key points

Next, we summarize key observations for the studied points:

Point P$_0$: $o_1 = 0.0\,\text{cm}$

- As evident in Figure 5.8, the IVC and SVC flows split nearly 50–50% between the LPA and RPA.
- Streamlines in Figure 5.5c remain closely parallel to the PA axis at the outflows.

Point P$_7$: $o_1 = 0.7\,\text{cm}$

- Figure 5.9c illustrates the emergence of helical flow patterns, particularly into the RPA. Figure 5.12 shows that SVC flow is pushed closer to the vessel wall, enhancing the helical pattern.
- Offsetting by roughly one vessel radius intensifies vortex structures, contributing to higher turbulent kinetic energy.

Point P$_8$: $o_1 = 0.8\,\text{cm}$

- Figure 5.13c illustrates that the helical flow remains but is less intense than at P$_7$.
- While this configuration corresponds to a local minimum of $\dot{\gamma}^A_{\text{nw}}$ at, fluctuations are still prominent as depicted in Figure 5.15.

Point P$_{24}$: $o_1 = 2.4\,\text{cm}$

- At large offsets, helical flow is not as prominent, and flow near the outflows aligns more closely with the PA axis as illustrated if Figure 5.17c and Figure 5.17d.
- As shown in Figure 5.20, IVC flow effectively completely pushes the SVC flow into the LPA, leading to highly unbalanced flow distribution which is physiologically suboptimal.
- In Figure 5.17a, the stagnation zone near the PA vessel wall where the IVC flow impinges on the wall exhibits regions of lower shear rate (Figure 5.18), which was reported to be a marker for blood stasis[1] and potential thrombosis risk [74, 75]. Additionally, this low-shear rate environment may cause endothelial dysfunction[2] due to inadequate wall shear stress [76].

In our idealized system, a symmetric (zero-offset) geometry balances the two inflows, reducing turbulent effects and distributing the near-wall shear rate evenly. While this configuration could be considered locally (in terms of local minima) favorable for this particular studied idealized system, it is not geometrically feasible in realistic TCPC designs. Previous studies have likewise reported beneficial flow patterns in zero-offset idealized TCPC models [41, 77]. However, real physiological data indicate that zero offset often corresponds to increased turbulence and higher energy dissipation [4, 6, 78, 79].

Overall, the obtained results align with other studies also reporting significant flow changes for offsets equal to half the IVC diameter and highlighting the preference for offsets of approximately equal to one IVC diameter. [4, 6, 80].

Finally, note that the anastomosis in the studied idealized system in this section lacks complex geometrical features such as flaring or curving, thus, the model serves mainly for the purposes of validating the optimization framework.

[1]Blood stasis refers to the slowing or pooling of blood within vessels, often caused by low flow rates [73].

[2]The endothelium is the thin layer of cells lining the blood vessels [73].

5.3.2 Optimization results

In this section, we present the results of the optimization studies on the Model 1 configuration using the Nelder-Mead and MADS methods. The objective functions considered are the near-wall shear rate and the turbulent kinetic energy. Our primary goals are to validate the optimization framework, evaluate the convergence of both optimization methods used, and compare their efficiency. Detailed specifications of the hardware used for these computations can be found at [81].

The custom Nelder-Mead implementation was parallelized and executed on four dedicated compute nodes, managed through a Slurm-based job submission system [82]. This setup enabled multiple function evaluations to be performed simultaneously, naturally reducing the total wall time for the optimization process. In contrast, the MADS method was not parallelized and ran sequentially on a single computational node.

To ensure that the optimized configurations remained physiologically meaningful, we imposed constraints on the offset parameter o_1. First, a primary constraint defined the feasible offset range as $0.0\,\text{cm} \leq o_1 \leq 2.4\,\text{cm}$. Next, to achieve the clinically motivated target (discussed in Section 5.1) that at least 25% of the IVC blood flow must be directed into the LPA, we refined the constraint further. Using precomputed splits to identify offsets that satisfied this flow distribution, the offset was ultimately constrained to $0.0\,\text{cm} \leq o_1 \leq 0.78\,\text{cm}$, as shown in Figure 5.4d.

The sampled data points in Figure 5.4 did not cover the entirety of the feasible set and were interpolated linearly, hence determining the exact minima of the objective functions is not possible. However, the data strongly suggests that under the given constraints, the minimum near-wall shear rate is located near the upper bound of the feasible set, i.e., near $o_1 = 0.78\,\text{cm}$. The minimum turbulent kinetic energy appears to be located near the lower bound, i.e., near $o_1 = 0.0\,\text{cm}$. Finally, the initial starting point for the optimization was chosen at the center of the feasible set, $o_1^{\text{init}} = 0.39\,\text{cm}$.

In the following studies, we focus on both the number of function evaluations ($\#f$) and the total wall time required. Here, $\#f$ corresponds to the number of simulation runs needed. Because the custom Nelder-Mead implementation supports parallel execution of these simulations, it can complete a larger number of evaluations in a shorter wall time compared to MADS, which operates sequentially. The longer runtime associated with the MADS approach makes it more computationally demanding. To avoid excessive runtimes, MADS was run with a preset cap of 20 on $\#f$, ensuring that the total computation time remained manageable.

Method	**Objective function**	**Page**
Nelder-Mead	$\dot{\gamma}_{\text{nw}}^{A}$	59
Nelder-Mead	T_{turb}^{A}	60
MADS	$\dot{\gamma}_{\text{nw}}^{A}$	61
MADS	T_{turb}^{A}	62

Table 5.3: List of studied optimization configurations.

Optimization setup 1: Basic cylindrical junction ($\dot{\gamma}^A_{\mathrm{NW}}$)

Objective: Minimizing near-wall shear rate $\dot{\gamma}^A_{\mathrm{nw}}$.

Geometrical model:

- Model 1 as described in Section 5.2.
- Optimization parameters: offset o_1.

Constraints:

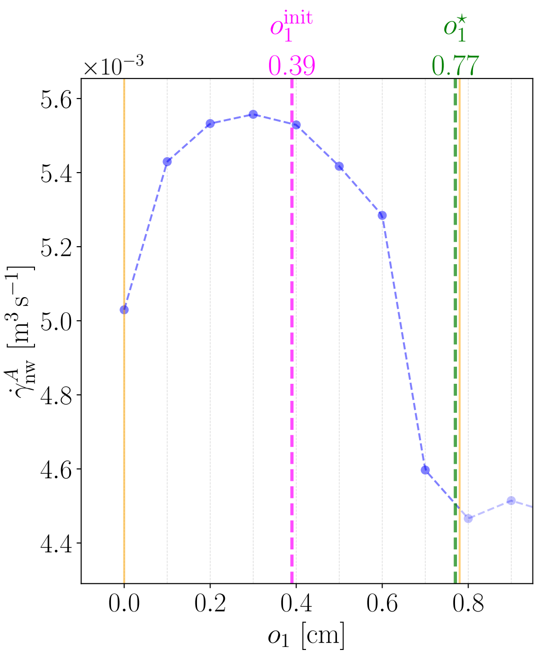
- Offset constraint: 0.0 cm $\leq o_1 \leq$ 2.4 cm.
- Flow split constraint: $F^{\mathrm{LPA}}_{\mathrm{IVC}} \geq 25$ % (see Figure 5.4d).

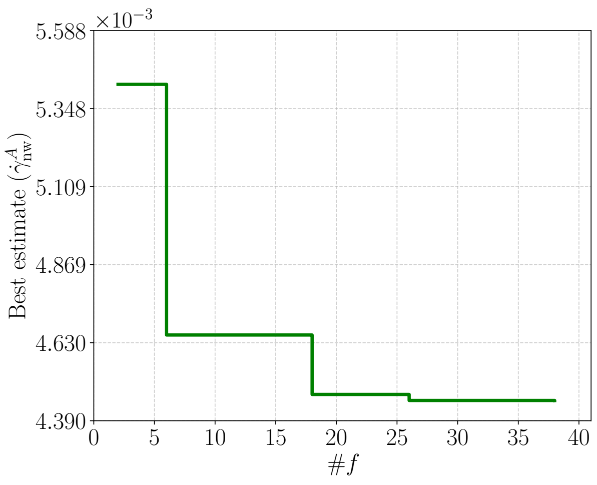
$\Rightarrow$ Combining these constraints leads to the final feasible range: 0.0 cm $\leq o_1 \leq$ 0.78 cm.

Optimization method:

- Nelder-Mead method as described in Section 4.3 and Appendix C.

Initial point: $o_1^{\mathrm{init}} = 0.39$ cm





Time [h]	Iterations
68.9	9

Figure 5.21: Optimization results for Optimization Setup 1. The left plot illustrates the initial point ($o_1^{\mathrm{init}} = 0.39$ cm) and the obtained result ($o_1^\star = 0.77$ cm) with respect to the sampled and interpolated objective function $\dot{\gamma}^A_{\mathrm{nw}}$, constrained within 0.0 cm $\leq o_1 \leq$ 0.78 cm. The right plot demonstrates the convergence of the optimization method, showing the best estimate against the number of objective function evaluations (#f). The table summarizes the total elapsed time of the optimization algorithm and the number of iterations of the used method.

OPTIMIZATION SETUP 2: BASIC CYLINDRICAL JUNCTION (T^A_{turb})

Objective: Minimizing turbulent kinetic energy T^A_{turb}.

Geometrical model:

- Model 1 as described in Section 5.2.
- Optimization parameters: offset o_1.

Constraints:

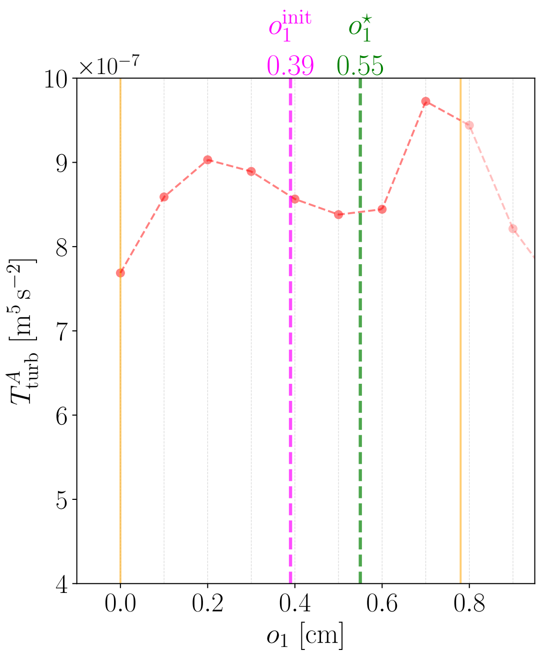
- Offset constraint: 0.0 cm $\le o_1 \le$ 2.4 cm.
- Flow split constraint: $F^{\text{LPA}}_{\text{IVC}} \ge 25$ % (see Figure 5.4d).

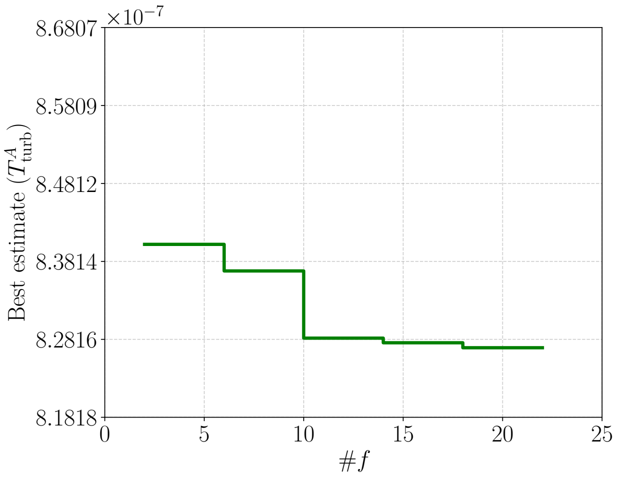
$\Rightarrow$ Combining these constraints leads to the final feasible range: 0.0 cm $\le o_1 \le$ 0.78 cm.

Optimization method:

- Nelder-Mead method as described in Section 4.3 and Appendix C.

Initial point: $o_1^{\text{init}} = 0.39$ cm





Time [h]	Iterations
44.7	5

Figure 5.22: Optimization results for Optimization Setup 2. The left plot illustrates the initial point (o_1^{init} = 0.39 cm) and the obtained result ($o_1^\star$ = 0.55 cm) with respect to the sampled and interpolated objective function T^A_{turb}, constrained within 0.0 cm $\le o_1 \le$ 0.78 cm. The right plot demonstrates the convergence of the optimization method, showing the best estimate against the number of objective function evaluations ($\#f$). The table summarizes the total elapsed time of the optimization algorithm and the number of iterations of the used method.

Optimization setup 3: Basic cylindrical junction ($\dot{\gamma}^A_{\text{nw}}$)

Objective: Minimizing turbulent kinetic energy $\dot{\gamma}^A_{\text{nw}}$.

Geometrical model:

- Model 1 as described in Section 5.2.
- Optimization parameters: offset o_1.

Constraints:

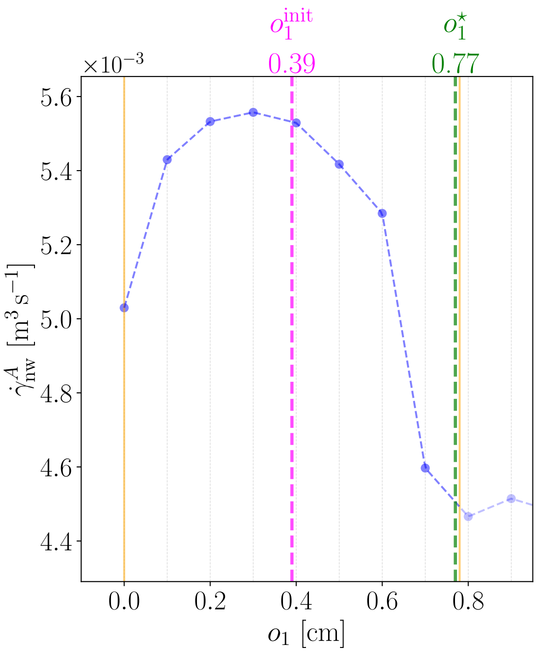
- Offset constraint: 0.0 cm $\leq o_1 \leq$ 2.4 cm.
- Flow split constraint: $F^{\text{LPA}}_{\text{IVC}} \geq 25$ % (see Figure 5.4d).

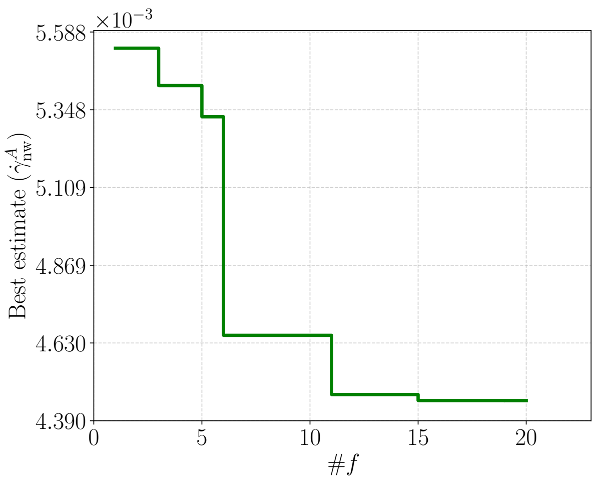
$\Rightarrow$ Combining these constraints leads to the final feasible range: 0.0 cm $\leq o_1 \leq$ 0.78 cm.

Optimization method:

- MADS method described in Section 4.3.

Initial point: $o_1^{\text{init}} = 0.39$ cm





Time $[h]$	Iterations
155.8	6

Figure 5.23: Optimization results for Optimization Setup 3. The left plot illustrates the initial point ($o_1^{\text{init}} = 0.39$ cm) and the obtained result ($o_1^{\star} = 0.77$ cm) with respect to the sampled and interpolated objective function $\dot{\gamma}^A_{\text{nw}}$, constrained within 0.0 cm $\leq o_1 \leq$ 0.78 cm. The right plot demonstrates the convergence of the optimization method, showing the best estimate against the number of objective function evaluations ($\#f$). The table summarizes the total elapsed time of the optimization algorithm and the number of iterations of the used method.

Optimization setup 4: Basic cylindrical junction (T^A_{turb})

Objective: Minimizing turbulent kinetic energy T^A_{turb}.

Geometrical model:

- Model 1 as described in Section 5.2.
- Optimization parameters: offset o_1.

Constraints:

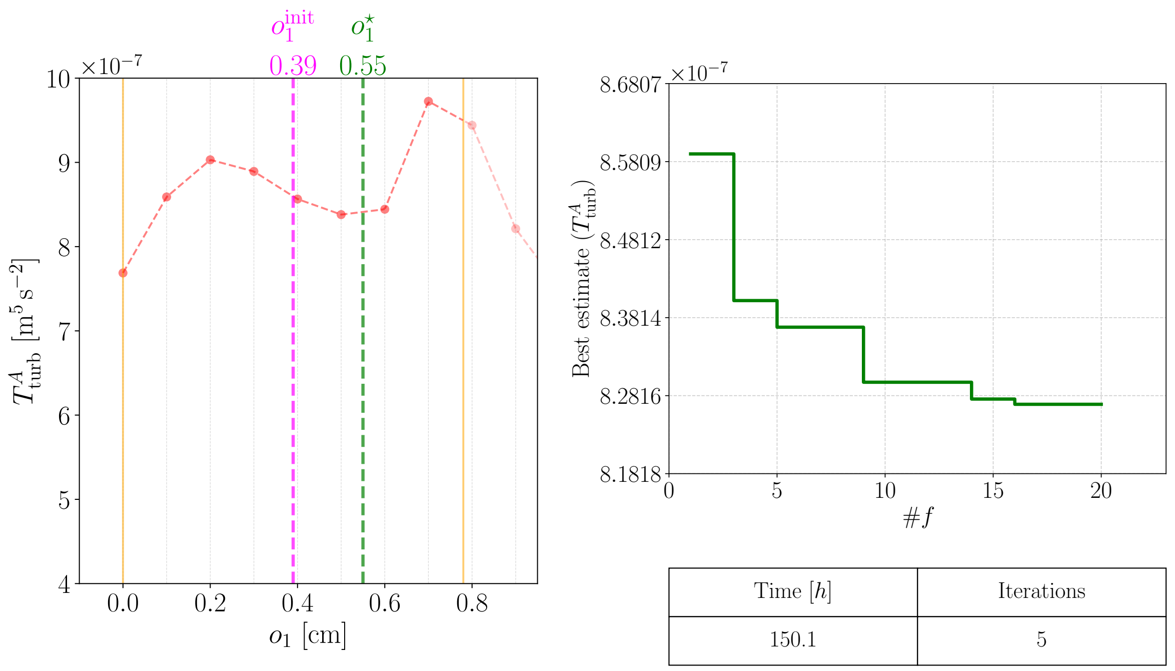
- Offset constraint: 0.0 cm $\leq o_1 \leq$ 2.4 cm.
- Flow split constraint: $F^{\text{LPA}}_{\text{IVC}} \geq 25$ % (see Figure 5.4d).

$\Rightarrow$ Combining these constraints leads to the final feasible range: 0.0 cm $\leq o_1 \leq$ 0.78 cm.

Optimization method:

- MADS method described in Section 4.3.

Initial point: $o_1^{\text{init}} = 0.39$ cm



Time [h]	Iterations
150.1	5

Figure 5.24: Optimization results for Optimization Setup 4. The left plot illustrates the initial point (o_1^{init} = 0.39 cm) and the obtained result ($o_1^{\star}$ = 0.55 cm) with respect to the sampled and interpolated objective function T^A_{turb}, constrained within 0.0 cm $\leq o_1 \leq$ 0.78 cm. The right plot demonstrates the convergence of the optimization method, showing the best estimate against the number of objective function evaluations ($\#f$). The table summarizes the total elapsed time of the optimization algorithm and the number of iterations of the used method.

From the optimization results, it can be observed that both the custom Nelder-Mead algorithm and the MADS method, when starting from the same initial point, resulted in the same outcomes for both the near-wall shear rate and turbulent kinetic energy minimization problems. However, for the turbulent kinetic energy objective, both methods converged to $o_1^\star = 0.55$ cm, even though for the given constraints the true minimum is expected to be at $o_1^\star = 0.0$ cm. This discrepancy is a direct consequence of the fact that neither Nelder-Mead nor MADS is a guaranteed global optimizer [62]. In the case of the near-wall shear rate minimization, both methods identified the minimum at $o_1^\star = 0.77$ cm, which aligns with expectations based on the interpolated sampled data.

It is worth noting that the MADS method consistently reached its predefined maximum number of objective function evaluations, leading to notably longer run times than the Nelder-Mead method. This difference stems from the fact that the current MADS method implementation runs sequentially. If MADS method were parallelized to allow simultaneous function evaluations, its overall execution time could be significantly reduced, potentially making it superior to the parallelized Nelder-Mead approach.

5.4 Problem with multiple optimization parameters

In this section, we focus on a more complex TCPC junction, referred to as Model 2, described in Section 5.2. Unlike the single-parameter system in the previous section, Model 2 incorporates multiple geometric degrees of freedom, offering greater flexibility in capturing realistic variations of the TCPC anatomy. The problem setup, detailed in Problem Setup 2, again aims to employ physiologically relevant domain dimensions, kinematic viscosity, and inflow velocities to reflect in vivo conditions.

Problem setup 2: More complex geometrical model

Physical setup:

- Domain: $\Omega = (0; 170\text{ mm}) \times (0; 22\text{ mm}) \times (0; 100\text{ mm})$.
- Kinematic viscosity: $\nu = 3 \cdot 10^{-6}\text{ m}^2\,\text{s}^{-1}$.
- Inflow at $\Gamma_{\text{in}}^{\text{IVC}}$: a constant velocity vector $\boldsymbol{u} = (u_1, u_2, u_3)^T$ aligned with the IVC axis. The components depend on the angle of the IVC axis α_1 (see Section 5.2), and are given by $u_1 = \sin\alpha_1 \cdot 0.45\text{ ms}^{-1}$, $u_2 = 0\text{ ms}^{-1}$, and $u_3 = \cos\alpha_1 \cdot 0.45\text{ ms}^{-1}$.
- Inflow at $\Gamma_{\text{in}}^{\text{SVC}}$: a constant velocity vector $\boldsymbol{u} = (u_1, u_2, u_3)^T$ aligned with the SVC axis. The components depend on the angle of the SVC axis α_2 (see Section 5.2), and are given by $u_1 = -\sin\alpha_2 \cdot 0.45\text{ ms}^{-1}$, $u_2 = 0\text{ ms}^{-1}$, and $u_3 = -\cos\alpha_2 \cdot 0.45\text{ ms}^{-1}$.

LBM setup:

- Initial condition on $\overline{\overline{\Omega}}$ set as described in Section 2.3.1.
- Boundary conditions at $\Gamma_{\text{out}}^{\text{W}}$ and $\Gamma_{\text{out}}^{\text{E}}$ set according to Section 2.3.2.
- Discretization: $N_1 = 768$, $N_2 = 105$, $N_3 = 453$ (total of 36529920 lattice sites).
- Kinematic viscosity in lattice units: $\nu^L = 10^{-3}$.

Given the increased number of parameters, attempting to sample the full range of possible offsets, angles, and connection curvatures becomes computationally demanding. In contrast to Model 1, where sampling a one-dimensional parameter space provided useful insight into the behavior of the objective functions, a similar approach for Model 2 is infeasible. Nonetheless, the enhanced complexity and flexibility of Model 2 allow it to capture a much wider array of potential TCPC modifications, potentially offering more realistic modeling capabilities compared to the simpler geometry of Model 1.

While Model 1 investigations considered both turbulent kinetic energy and near-wall shear rate, in this section we focus on minimizing the near-wall shear rate alone. As described in Section 5.1, the near-wall shear rate is integrated over a thin layer near the vessel walls. However, the total volume $|A|$ of the control volume (along with the layer $|A_\varepsilon|$) can vary across different geometric configurations in Model 2.

To ensure a fair comparison, we employ a volume-normalized version of the near-wall shear rate, denoted by $\widetilde{\dot{\gamma}}^A_{\mathrm{nw}}$ [s^{-1}]:

$$\widetilde{\dot{\gamma}}^A_{\mathrm{nw}} = \frac{\dot{\gamma}^A_{\mathrm{nw}}}{|A_\varepsilon|}, \tag{5.4}$$

where $\dot{\gamma}^A_{\mathrm{nw}}$ is given by Eq. (5.2), and $|A_\varepsilon|$ is the corresponding volume of the ε-thick layer. By scaling with respect to $|A_\varepsilon|$, we obtain an intensive measure that provides a better comparison of performance across geometrical variations.

In addition to minimizing the near-wall shear rate, we also monitor the flow distribution from the IVC to the PA branches, similarly to the approach used in Section 5.3. However, unlike the method of precomputing the threshold in the previous section, we now dynamically monitor the IVC flow split using a custom postprocessing code.

The code, available on Github at [83], utilizes the computed mean velocity field and calculates the fraction of IVC flow directed to each PA branch. This more flexible approach allows us to enforce physiologically motivated constraints on the IVC flow split during the optimization process.

5.4.1 Optimization results

Unlike in the previous section, where we started the search from the center of the feasible space, we now choose an arbitrary initial point $\boldsymbol{x}^{\mathrm{init}} = (o_1^{\mathrm{init}}, \alpha_1^{\mathrm{init}}, \alpha_2^{\mathrm{init}}, \eta_1^{\mathrm{init}}, \eta_2^{\mathrm{init}})$ with predefined values for each geometric parameter given by:

$$o_1^{\mathrm{init}} = 0.1\,\mathrm{cm}, \quad \alpha_1^{\mathrm{init}} = 4^\circ, \quad \alpha_2^{\mathrm{init}} = -3^\circ, \quad \eta_1^{\mathrm{init}} = 0.1\,\mathrm{cm}, \quad \eta_2^{\mathrm{init}} = 0.1\,\mathrm{cm}. \tag{5.5}$$

This choice serves several purposes. First, it provides a slight perturbation relative to a perfectly symmetric geometry, helping to avoid converging to possible local minimum previously observed in Section 5.3. Second, because it is not simply the center of the optimization space, it could more closely mimic a scenario in which an existing, idealized TCPC model is being refined. As in the previous section, we compare both our custom Nelder–Mead implementation and the MADS method under these new conditions.

Details regarding the optimization parameters and constraints are provided in Optimization Setup 5 and Optimization Setup 6. The comparison of the results obtained using the two methods is shown in Figure 5.25.

	o_1 [cm]	α_1 [°]	α_2 [°]	η_1 [cm]	η_2 [cm]
Lower bound	−1	−20	−20	0	0
Upper bound	1	20	20	0.25	0.25

Table 5.4: Constraints imposed on the geometric parameters in Optimization Setup 5 and Optimization Setup 6.

Optimization setup 5: More complex geometrical model ($\widehat{\dot{\gamma}}^{A}_{\mathrm{nw}}$)

Objective: Minimizing volume-normalized near-wall shear rate $\widehat{\dot{\gamma}}^{A}_{\mathrm{nw}}$.

Geometrical model:

- Model 2 as described in Section 5.2.
- Optimization parameters: o_1, α_1, α_2, η_1, η_2 (meanings of the parameters described in Section 5.2).

Constraints:

- Geometrical constraints as described in Table 5.4.
- Flow split constraint: $F^{\mathrm{LPA}}_{\mathrm{IVC}} \geq 25$ % (calculated during the optimization run using postprocessing).

Optimization method:

- Nelder-Mead method described in Section 4.3 and Appendix C.

Initial point: $\boldsymbol{x}^{\mathrm{init}} = (o_1^{\mathrm{init}}, \alpha_1^{\mathrm{init}}, \alpha_2^{\mathrm{init}}, \eta_1^{\mathrm{init}}, \eta_2^{\mathrm{init}})$ given by Eq. (5.5).

Optimization setup 6: More complex geometrical model ($\widehat{\dot{\gamma}}^{A}_{\mathrm{nw}}$)

Objective: Minimizing volume-normalized near-wall shear rate $\widehat{\dot{\gamma}}^{A}_{\mathrm{nw}}$.

Geometrical model:

- Model 2 as described in Section 5.2.
- Optimization parameters: o_1, α_1, α_2, η_1, η_2 (meanings of the parameters described in Section 5.2).

Constraints:

- Geometrical constraints as described in Table 5.4.
- Flow split constraint: $F^{\mathrm{LPA}}_{\mathrm{IVC}} \geq 25$ % (calculated during the optimization run using postprocessing).

Optimization method:

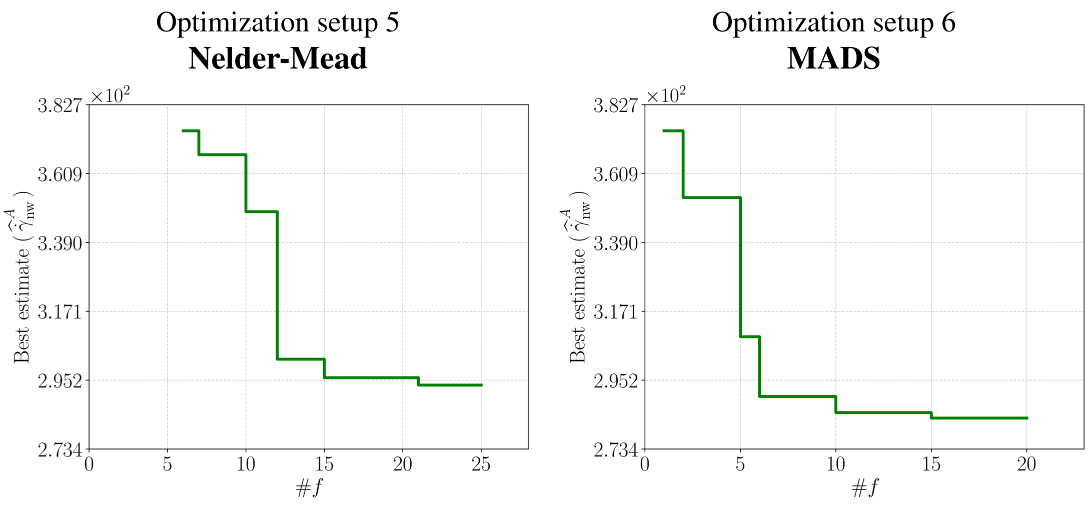
- MADS method described in Section 4.3.

Initial point: $\boldsymbol{x}^{\mathrm{init}} = (o_1^{\mathrm{init}}, \alpha_1^{\mathrm{init}}, \alpha_2^{\mathrm{init}}, \eta_1^{\mathrm{init}}, \eta_2^{\mathrm{init}})$ given by Eq. (5.5).

Both methods successfully improved upon the initial geometry. However, notable differences emerged in their performance and final solutions. Starting from the same initial point, MADS achieved a lower value of the objective function in fewer function evaluations, indicating a superior search strategy through the optimization space. Nonetheless, as in Section 5.3, due to its sequential nature (i.e., no parallelization), MADS required significantly more time to reach its evaluation limit. Note that the MADS method was again run with a preset cap of 20 on $\#f$. In contrast, the parallelized Nelder-Mead method again finished much sooner despite needing more evaluations.

Furthermore, the final geometries found by the two algorithms differ substantially, which underscores how multi-dimensional parameter spaces can magnify distinctions in search strategies. These results also highlight the strong potential of the MADS method: with a parallelized implementation, it could surpass Nelder-Mead in both convergence speed and final accuracy for higher-dimensional optimization problems.

To further examine the different solutions, we investigate additional metrics and flow characteristics within the final geometries–geometry corresponding to $\boldsymbol{x}^\star_{\mathbf{NM}}$ is examined on pages 67–68, and geometry corresponding to $\boldsymbol{x}^\star_{\mathbf{MADS}}$ on pages 69–70.



Time $[h]$	Iterations
49.2	6

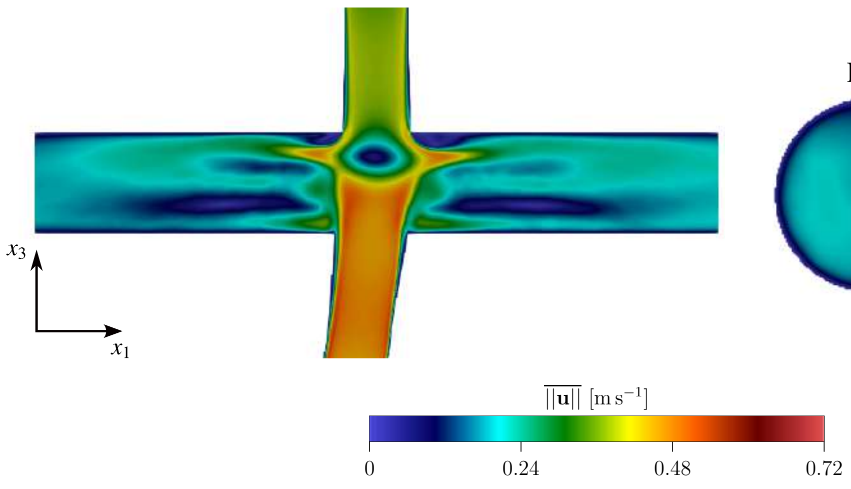
Time $[h]$	Iterations
144.2	6

	o_1 [cm]	α_1 [°]	α_2 [°]	η_1 [cm]	η_2 [cm]
$\boldsymbol{x}^\star_{\mathbf{NM}}$	−0.139	6.8	−0.2	0.058	0.128
$\boldsymbol{x}^\star_{\mathbf{MADS}}$	0.1	16	13	0	0.25

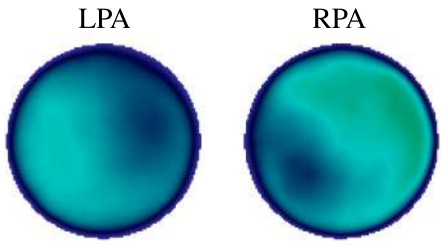
Figure 5.25: Comparison of optimization results for Optimization Setup 5 and Optimization Setup 6. The plots illustrate the convergence behavior by showing the best estimate of the objective function value against the number of evaluations ($\#f$). The tables below the plots provide a summary of the total elapsed time and the number of iterations for each optimization method. Additionally, the bottom table provides the comparison of the points identified as the optimal solution by the Nelder-Mead method ($\boldsymbol{x}^\star_{\mathbf{NM}}$) and the MADS method ($\boldsymbol{x}^\star_{\mathbf{MADS}}$).

Point $\boldsymbol{x}^{\star}_{\mathbf{NM}}$

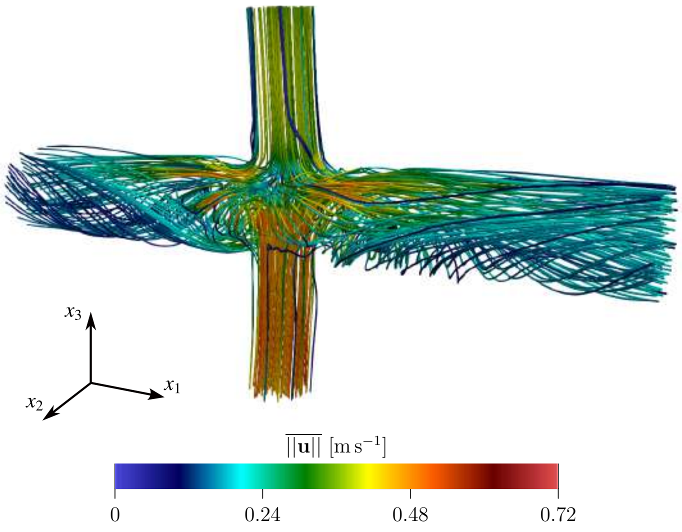
Mean velocities



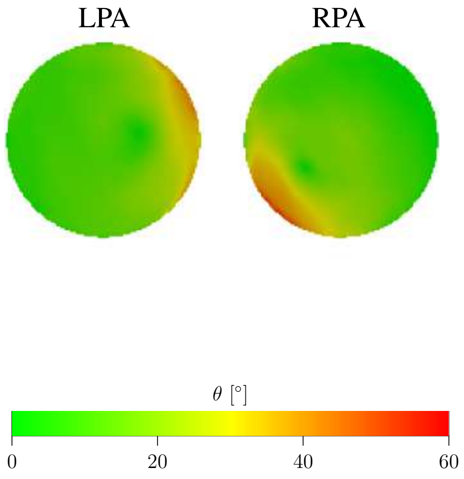
(a) Mean velocity magnitude field in the x_1-x_3 plane.



(b) Magnitudes of mean velocity at the outlets $\Gamma^{\text{LPA}}_{\text{out}}$ and $\Gamma^{\text{RPA}}_{\text{out}}$.



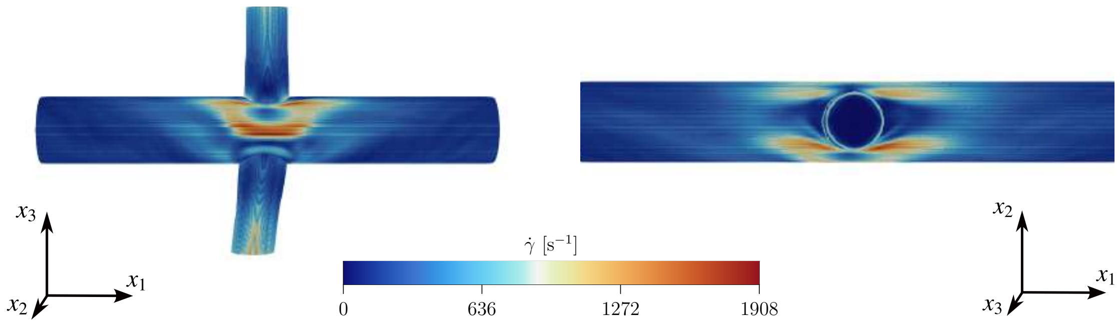
(c) Streamlines of the mean velocity field illustrating the flow trajectory throughout the geometry.



(d) Angle between the mean velocity vector and the normal direction at the outlets $\Gamma^{\text{LPA}}_{\text{out}}$ and $\Gamma^{\text{RPA}}_{\text{out}}$.

Figure 5.26: Overview of mean velocity fields and outlet characteristics for the case of point $\boldsymbol{x}^{\star}_{\mathbf{NM}}$. Subfigure (a) shows the velocity magnitude field, (b) displays outlet-specific velocity magnitudes, (c) presents streamlines of the mean velocity field, and (d) details the angular alignment of velocity vectors with outlet normals.

Shear rate

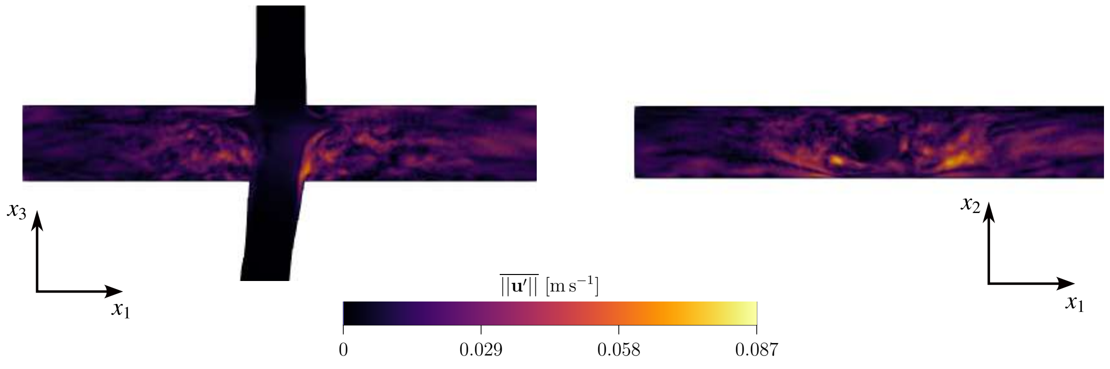


(a) Mean shear rate in the x_1-x_3 plane.

(b) Mean shear rate in the x_1-x_2 plane.

Figure 5.27: Mean shear rate visualizations in the x_1-x_3 and in the x_1-x_2 plane for the case of point $\boldsymbol{x}^\star_{\mathbf{NM}}$.

Velocity fluctuations

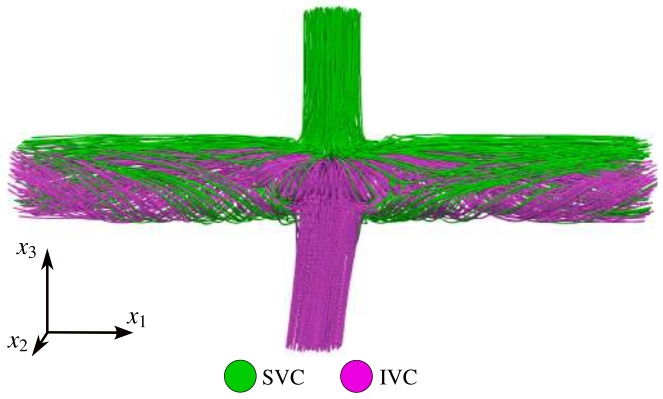


(a) Mean velocity fluctuations magnitude field in the x_1-x_3 plane.

(b) Mean velocity fluctuations magnitude field in the x_1-x_2 plane.

Figure 5.28: Mean velocity fluctuations in the x_1-x_3 and in the x_1-x_2 plane for the point $\boldsymbol{x}^\star_{\mathbf{NM}}$.

SVC/IVC split

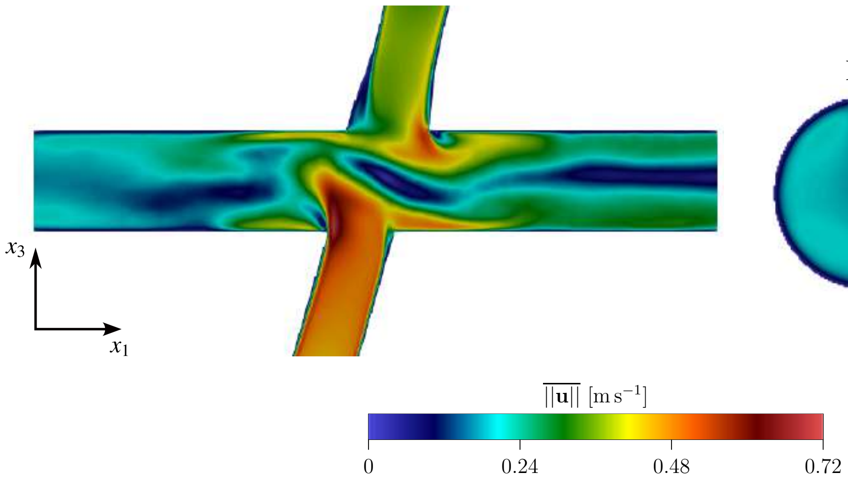


	RPA	LPA
SVC	45.19%	48.81%
IVC	45.67%	54.33%

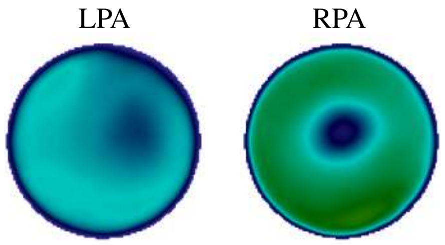
Figure 5.29: Analysis of the split between the SVC and IVC contributions to the LPA and the RPA for the case of point $\boldsymbol{x}^\star_{\mathbf{NM}}$. The figure illustrates the flow split, and the table provides the exact percentages.

Point $\boldsymbol{x}^{\star}_{\mathbf{MADS}}$

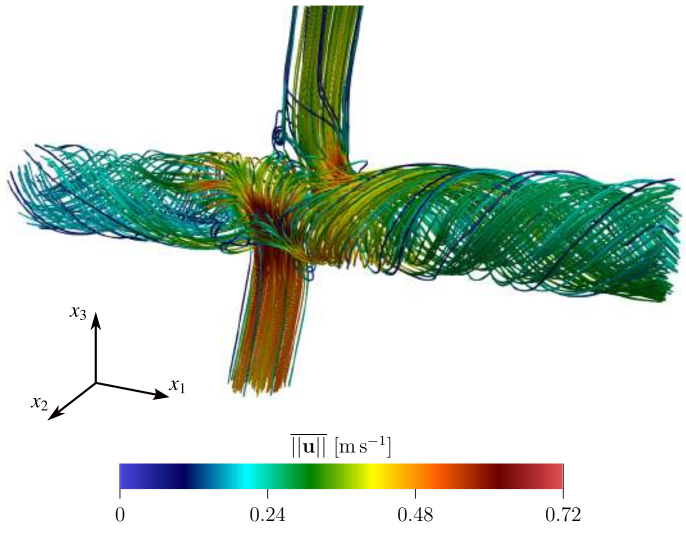
Mean velocities



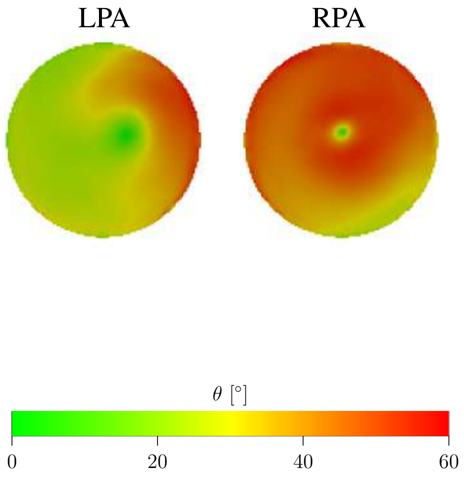
(a) Mean velocity magnitude field in the x_1-x_3 plane.



(b) Magnitudes of mean velocity at the outlets $\Gamma^{\text{LPA}}_{\text{out}}$ and $\Gamma^{\text{RPA}}_{\text{out}}$.



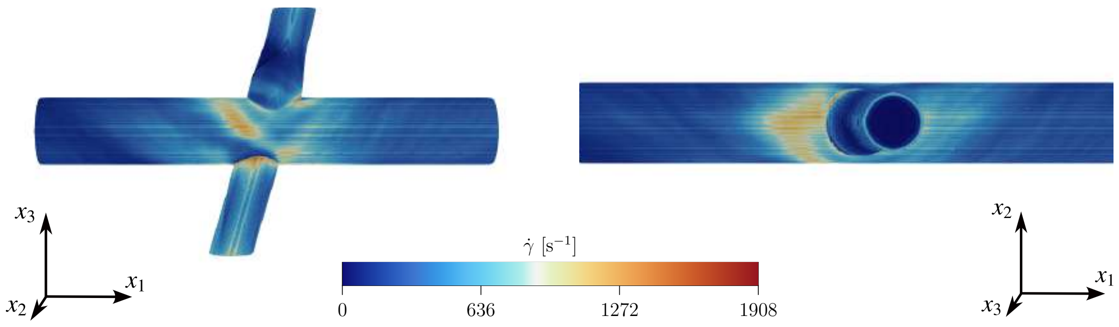
(c) Streamlines of the mean velocity field illustrating the flow trajectory throughout the geometry.



(d) Angle between the mean velocity vector and the normal direction at the outlets $\Gamma^{\text{LPA}}_{\text{out}}$ and $\Gamma^{\text{RPA}}_{\text{out}}$.

Figure 5.30: Overview of mean velocity fields and outlet characteristics for the case of point $\boldsymbol{x}^{\star}_{\mathbf{MADS}}$. Subfigure (a) shows the velocity magnitude field, (b) displays outlet-specific velocity magnitudes, (c) presents streamlines of the mean velocity field, and (d) details the angular aligmadsent of velocity vectors with outlet normals.

Shear rate

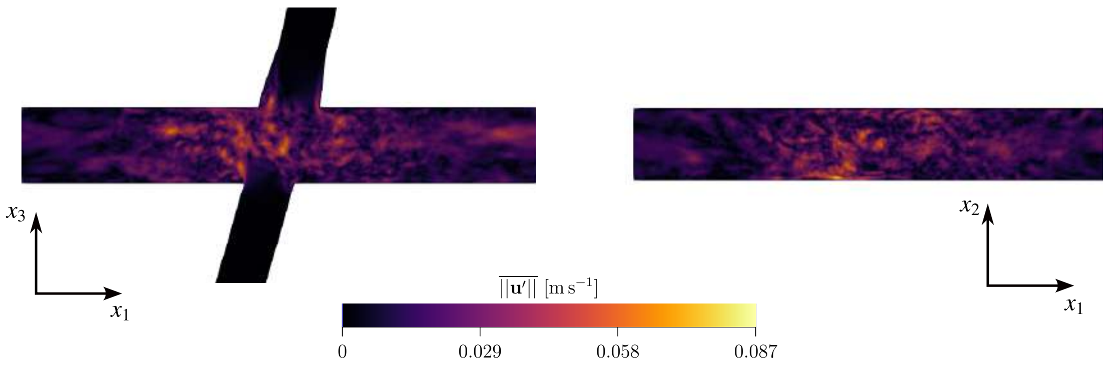


(a) Mean shear rate in the x_1-x_3 plane.

(b) Mean shear rate in the x_1-x_2 plane.

Figure 5.31: Mean shear rate visualizations in the x_1-x_3 and in the x_1-x_2 plane for the case of point $\boldsymbol{x}^\star_{\mathbf{MADS}}$.

Velocity fluctuations

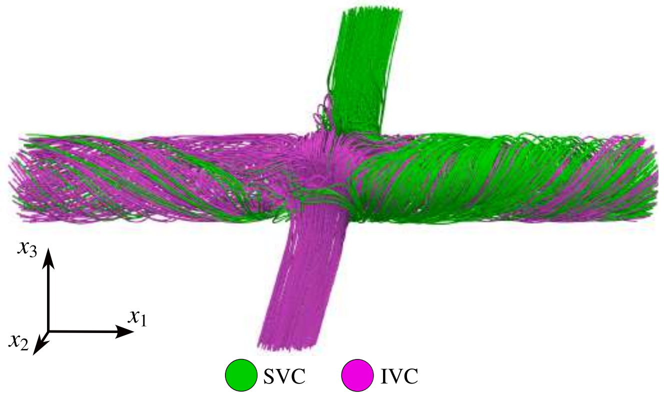


(a) Mean velocity fluctuations magnitude field in the x_1-x_3 plane.

(b) Mean velocity fluctuations magnitude field in the x_1-x_2 plane.

Figure 5.32: Mean velocity fluctuations in the x_1-x_3 and in the x_1-x_2 plane for the point $\boldsymbol{x}^\star_{\mathbf{MADS}}$.

SVC/IVC split



	RPA	LPA
SVC	20.68%	79.32%
IVC	66.02%	33.98%

Figure 5.33: Analysis of the split between the SVC and IVC contributions to the LPA and the RPA for the case of point $\boldsymbol{x}^\star_{\mathbf{MADS}}$. The figure illustrates the flow split, and the table provides the exact percentages.

The resulting geometry of the Nelder-Mead method optimization remains largely symmetric, with the main asymmetry arising from a slight tilt of the IVC. The flow field exhibits a fairly balanced distribution and relatively mild helical structures as can be observed in Figure 5.26. In contrast, the geometry obtained with the MADS method displays more pronounced asymmetry, as both the IVC and SVC are tilted. This induces a stronger helical flow pattern, as shown in Figure 5.30.

5.5 Summary

This chapter presented an in-depth analysis of two idealized TCPC models to evaluate the feasibility and effectiveness of the proposed optimization framework. The study focused on two objective functions–near-wall shear rate and turbulent kinetic energy–while also monitoring flow distribution and velocity alignment at the outlets.

The findings demonstrated the framework's ability to address complex hemodynamic questions with a focus on clinically relevant metrics. In the simplified Model 1, systematic sampling and optimization revealed insights into flow behavior under varying caval offsets. Both the Nelder-Mead and MADS methods validated the framework's functionality, identifying optimal configurations within physiological constraints. However, differences in their convergence behavior and computational requirements underscored opportunities for refinement. Notably, parallelizing the Nelder-Mead method significantly reduced runtimes in both Section 5.3 and Section 5.4 compared to the serial implementation of MADS method, highlighting the importance of optimizing the optimization process itself.

For the more complex Model 2, the framework successfully tackled a multidimensional optimization problem with multiple geometric degrees of freedom. Here, MADS method demonstrated a potentially superior strategy for navigating high-dimensional parameter spaces, suggesting its suitability for more advanced optimization tasks. Nonetheless, the efficiency gains from parallelizing the Nelder-Mead method underscore the crucial role of choosing and parallelizing the right optimization algorithms to improve our framework's performance.

Beyond validating the optimization framework, the results highlighted the utility of LBM for qualitative analyses of cardiovascular systems. Its computational efficiency facilitates high-fidelity flow analyses, making it a powerful tool for capturing detailed hemodynamic information in complex geometries.

In conclusion, the proposed framework has proven feasible and effective, offering significant potential to support extensive studies and optimization tasks in cardiovascular research. Future advancements, such as parallelizing the MADS method and refining optimization algorithms to more effectively target global optima, promise to further enhance the framework's efficiency (in terms of both runtime and computational resources) and broaden its applicability to other optimization problems.

Conclusion

The primary objective of this thesis was to develop and evaluate an optimization framework specifically designed to improve the geometry of an idealized total cavopulmonary connection (TCPC). The framework integrates numerical simulations using the lattice Boltzmann method (LBM), automated geometry generation, and gradient-free optimization techniques.

The thesis begins by introducing the mathematical model used to describe blood flow in vessels, detailing the assumptions and simplifications necessary for computational feasibility. The second chapter provides an in-depth overview of LBM as the selected numerical method. The third chapter describes the custom geometry generation pipeline, emphasizing its ability to produce simulation-ready voxelized geometries from parameterized templates. The fourth chapter discusses gradient-free optimization methods and outlines the design of the proposed optimization framework. Finally, the results of applying this framework to two simplified TCPC models are presented, validating its functionality and demonstrating its flexibility.

Building on previous work [19, 20], this thesis extends the application of optimization to three-dimensional geometries and explores the behavior of simplified TCPC models. The study assumes various simplifications enabling efficient testing and validation of the framework. The results demonstrate the framework's ability to tackle complex problems, validating its potential to optimize surgical designs in the context of cardiovascular systems. Furthermore, the study highlights how computational fluid dynamics (CFD) can provide detailed quantitative and qualitative insights into blood flow, aiding clinicians in making informed decisions. Additionally, the computational efficiency of LBM was instrumental in enabling high-fidelity flow-field analyses across various geometries, offering valuable insights into the studied system.

The insights gained from this work establish a solid foundation for future research. By extending the framework to account for more complex physiological settings and to model patient-specific geometries, it holds potential for addressing clinically relevant challenges. Collaborations with institutions such as the Institute for Clinical and Experimental Medicine (IKEM) in Prague and the UT Southwestern (UTSW) Medical Center in Dallas will provide opportunities to validate the model against experimental data, such as flow MRI, and apply it to real-world cases. These advancements could bridge the gap between theoretical research and practical implementation, paving the way for improved clinical outcomes and personalized treatment planning. This work thus represents a meaningful step forward in applying computational optimization in cardiovascular surgery, while also underlining the potential of CFD for analyzing complex cardiovascular systems.

Appendices

Appendix A

Objective functions and geometry loading in LBM

Loading the geometry

The following function reads an automatically generated `.txt` file with the prescribed dimensions of the lattice meant to be used in a simulation.

```
std::ifstream infile("path_to_dimensions" + fname);
int X_given, Y_given, Z_given, X, Y, Z;

while (infile >> X_given >> Y_given >> Z_given)
{
    X = X_given;
    Y = Y_given;
    Z = Z_given;
}
```

Code Listing A.1: Implementation of loading the dimensions of the domain.

The following function reads a `.txt` file generated by geometry generation process and assigns corresponding types to the lattice nodes.

```
void generateObject()
{
    for (int x = 0; x < globX; x++) {
        for (int y = 0; y < globY; y++) {
            for (int z = 0; z < globZ; z++) {
                near_wall[POS(x, y, z, globX, globY, globZ)] = 0;
            }
        }
    }

    std::ifstream infile("path_to_geometry" + fname);
    int x, y, z, type;

    while (infile >> x >> y >> z >> type) {
        // Assign based on type
        if (type == 0) {
            nse.setMap(x, y, z, BC::GEO_NOTHING);
            wall_type[POS(x, y, z, globX, globY, globZ)] = 0;
        }
        // Additional conditions include assignment of fluid cells,
        // outflow cells, inflow cells, etc. Omitted for brevity...
    }
}
```

Code Listing A.2: Implementation of the `generateObject` function for geometry initialization.

Velocity gradient calculation

This function calculates the velocity gradient at a given point within the simulation domain. It uses finite difference scheme, where either one-sided or central difference is used based on the position of the point relative to a wall.

```
velocityGradient calculateVelocityGradient(idx x, idx y, idx z)
{
    velocityGradient dV{};

    if (wall_type[POS(x, y, z, globX, globY, globZ)] != 1) {
        dV.V11 = dV.V12 = dV.V13 = no0;
        dV.V21 = dV.V22 = dV.V23 = no0;
        dV.V31 = dV.V32 = dV.V33 = no0;
        return dV;
    }
    for (auto& block : nse.blocks) {
    // Example for dv_x/dx
        if (wall_type[POS(x + 1, y, z, globX, globY, globZ)] == 2) {
            dV.V11 = block.hmacro(MACRO::e_vx, x, y, z)
                   - block.hmacro(MACRO::e_vx, x - 1, y, z);
        }
        else if (wall_type[POS(x - 1, y, z, globX, globY, globZ)] == 2) {
            dV.V11 = block.hmacro(MACRO::e_vx, x + 1, y, z)
                   - block.hmacro(MACRO::e_vx, x, y, z);
        }
        else {
            dV.V11 = n1o2 * (block.hmacro(MACRO::e_vx, x + 1, y, z)
                   - block.hmacro(MACRO::e_vx, x - 1, y, z));
        }

    // Other components follow the same scheme, omitted for brevity...
    }
    return dV;
}
```

Code Listing A.3: Implementation of the `calculateVelocityGradient` function for velocity gradient calculation.

Shear rate calculation

This function computes the shear rate $\dot{\gamma}$, defined by 1.8, at a given point in the simulation domain.

```
real dotGamma(idx x, idx y, idx z)
{
    velocityGradient dV = calculateVelocityGradient(x, y, z);
    real shear_rate = sqrt(2) * sqrt(
        SQ(dV.V11) + SQ(dV.V12) + SQ(dV.V13) + SQ(dV.V21) + SQ(dV.V22) +
        SQ(dV.V23) + SQ(dV.V31) + SQ(dV.V32) + SQ(dV.V33));

    return shear_rate;
}
```

Code Listing A.4: Implementation of the `dotGamma` function for shear rate calculation.

Q-Criterion calculation

This function calculates the Q-criterion, which is used to identify vortex structures in the simulation domain. The Q-criterion is defined as

$$Q = \frac{1}{2}\left(\|\mathbf{\Omega}\|^2 - \|\mathbf{D}\|^2\right),$$

where is the symmetric strain rate tensor and $\mathbf{\Omega} = \frac{1}{2}\left[\nabla \boldsymbol{u} - (\nabla \boldsymbol{u})^T\right]$, both obtained from the velocity gradient tensor $\nabla \boldsymbol{u}$ [84]. Regions where $Q > 0$ indicate vorticity regions.

```
    real qCriterion(idx x, idx y, idx z)
    {
        velocityGradient dV = calculateVelocityGradient(x, y, z);
        // Symmetric part (strain rate tensor S)
        real S11 = dV.V11;
        real S12 = n1o2 * (dV.V12 + dV.V21);
        real S13 = n1o2 * (dV.V13 + dV.V31);
        real S22 = dV.V22;
        real S23 = n1o2 * (dV.V23 + dV.V32);
        real S33 = dV.V33;
        // Antisymmetric part (vorticity tensor Omega)
        real Omega12 = n1o2 * (dV.V12 - dV.V21);
        real Omega13 = n1o2 * (dV.V13 - dV.V31);
        real Omega23 = n1o2 * (dV.V23 - dV.V32);

        real normSqrS = SQ(S11) + no2 * NORM(S12, S13, S23) + SQ(S22)
                      + SQ(S33);
        real normSqrOmega = no2 * NORM(Omega12, Omega13, Omega23);

        return n1o2 * (normSqrOmega - normSqrS);
    }
```

Code Listing A.5: Implementation of the `qCriterion` function for vortex structure identification.

Outflow calculation

This function calculates the outflow at an outlet.

```
real calculateOutflow(idx outlet, bool isPositiveDirection)
{
    real Q_out = 0.0;
    int direction = isPositiveDirection ? no1 : -no1;

    for (idx y = 0; y < globY; y++) {
    for (idx z = 0; z < globZ; z++) {
        if (wall_type[POS(outlet, y, z, globX, globY, globZ)] != no2) {
            Q_out += direction *
                     mean_vx[POS(outlet, y, z, globX, globY, globZ)];
        }
    }
    }

    return Q_out;
}
```

Code Listing A.6: Implementation of the `calculateOutflow` function for flow rate calculation.

Appendix B

Gmsh template files

```
SetFactory("OpenCASCADE");
// 1 optimisation parameter - offset of the bottom cylinder
// Characteristic mesh length
h = 0.0005;
Mesh.CharacteristicLengthMin = h;
Mesh.CharacteristicLengthMax = h;

// Cylinder dimensions
LOWER_LENGTH = 0.05;  // Length of the lower cylinder in meters
LOWER_RADIUS = 0.007; // Radius of the lower cylinder in meters
UPPER_LENGTH = 0.05;  // Length of the upper cylinder in meters
UPPER_RADIUS = 0.007; // Radius of the upper cylinder in meters
MIDDLE_LENGTH = 0.17; // Length of the middle cylinder in meters
MIDDLE_RADIUS = 0.011;  // Radius of the middle cylinder in meters
// Offset for positioning along the X-axis
OFFSET = DEFINE_OFFSET;

////// First Cylinder - Lower //////
// Define points along the axis of the lower cylinder
Point(1) = {OFFSET, 0.0, LOWER_LENGTH, h};
Point(2) = {OFFSET, 0.0, 0.0, h};
// Create line and wire for lower cylinder extrusion
Line(1) = {2, 1};
Wire(2) = {1};
// Disk representing the base of the lower cylinder
Disk(1) = {OFFSET, 0.0, LOWER_LENGTH, LOWER_RADIUS};
// Extrude the surface to form the first cylinder volume
Extrude { Surface{1}; } Using Wire {2}

////// Second Cylinder - Upper //////
// Define points along the axis of the upper cylinder
Point(101) = {0.0, 0.0, LOWER_LENGTH + UPPER_LENGTH, h};
Point(102) = {0.0, 0.0, LOWER_LENGTH, h};
// Create line and wire for upper cylinder extrusion
Line(101) = {102, 101};
Wire(102) = {101};
// Disk representing the base of the upper cylinder
```

```
    Disk(101) = {0.0, 0.0, LOWER_LENGTH + UPPER_LENGTH, UPPER_RADIUS};
    // Extrude the surface to form the second cylinder volume
    Extrude { Surface{101}; } Using Wire {102}

    ////// Third Cylinder - Middle //////
    // Define points along the axis of the middle cylinder
    Point(10001) = {-MIDDLE_LENGTH / 2.0, 0.0, LOWER_LENGTH, h};
    Point(10002) = {MIDDLE_LENGTH / 2.0, 0.0, LOWER_LENGTH, h};
    // Create line and wire for middle cylinder extrusion
    Line(10001) = {10002, 10001};
    Wire(10002) = {10001};
    // Disk representing the base of the middle cylinder
    Disk(10001) = {-MIDDLE_LENGTH / 2.0, 0.0, LOWER_LENGTH, MIDDLE_RADIUS
   };
    // Orient disk along the XY plane
    Rotate{{0, 1, 0}, {-MIDDLE_LENGTH / 2.0, 0.0, LOWER_LENGTH}, Pi/2}{
   Surface{10001};}
    // Extrude the surface to form the third cylinder volume
    Extrude { Surface{10001}; } Using Wire {10002}

    ////// Union and Cleanup //////
    // Unite the three volumes and delete original parts
    BooleanUnion{ Volume{1}; Delete; }{ Volume{2, 3}; Delete; }
    // Delete any remaining surfaces
    Recursive Delete { Surface{1, 101, 10001 }; }
    // Delete any remaining lines
    Recursive Delete { Line{1, 101, 10001}; }
```

Code Listing B.1: An example of `.geo` template file implemented in Gmsh.

Appendix C

Custom Nelder-Mead algorithm implementation

C.1 Key features

Initialization

The simplex is initialized around the starting point `x_start` using a defined step size. The function values for each vertex of the simplex are evaluated to initialize scores.

Stopping criteria

The algorithm stops when:

- The maximum number of iterations (`max_iter`) is reached.
- The objective function improvement over consecutive iterations is below a defined threshold (`no_improve_thr`) for a specified number of iterations (`no_improv_break`).

Function evaluations

While the implementation is suboptimal in terms of the total number function evaluations, this design choice was made because, in the case specific for this work, a single function evaluation usually takes up to several hours. Thus, the algorithm prioritizes evaluating all candidate points (with potential parallelization) before making comparisons or updates to the simplex. This approach ensures that computational resources are utilized efficiently, even at the cost of additional function evaluations in some scenarios.

C.2 The implementation

The main part of the custom implementation is presented in Listing C.1. Full implementation is available upon request on Github at `https://github.com/buresjan/nelder_mead`. The online code also includes several test cases showing the functionality of the custom implementation.

```python
def nelder_mead(
    f,
    x_start,
    step=0.01,
    no_improve_thr=1e-6,
    no_improv_break=20,
    max_iter=1000,
    # Standard Nelder-Mead parameters
    delta_e=2.0,  # expansion coefficient
    delta_oc=0.5,  # outside contraction coefficient
    delta_ic=0.5,  # inside contraction coefficient
    gamma=0.5,  # shrink coefficient
    verbose=False,
):
simplex, scores = initialize_simplex(f, x_start, step, verbose)
prev_best = scores[0]
no_improv = 0
iter_count = 0

while True:
    # Order simplex by score
    simplex, scores = order_simplex(simplex, scores)
    best = scores[0]

    # Verbose output
    if verbose:
        print(
            f"Iteration {iter_count}: Best estimate = {simplex[0]}, "
            f"Function value = {best}, "
            f"Simplex: {simplex}"
        )

    # Check stopping criteria
    if max_iter and iter_count >= max_iter:
        return simplex[0], best
    if no_improv >= no_improv_break:
        return simplex[0], best

    iter_count += 1
    if best < prev_best - no_improve_thr:
        prev_best = best
        no_improv = 0
    else:
        no_improv += 1

    # Compute centroid (excluding the worst point)
    centroid = compute_centroid(simplex)

    # Generate candidate points
    candidates = generate_candidate_points(
        centroid, simplex[-1], delta_e, delta_oc, delta_ic
    )

    # Evaluate candidates in parallel
    candidate_keys = [
        "reflection",
        "expansion",
        "outside_contraction",
        "inside_contraction",
    ]
    candidate_values = [candidates[key] for key in candidate_keys]

    with concurrent.futures.ProcessPoolExecutor() as executor:
        future_to_key = {
            executor.submit(f, candidate): key
            for key, candidate in zip(candidate_keys, candidate_values)
        }
        results = {}
        for future in concurrent.futures.as_completed(future_to_key):
            key = future_to_key[future]
            results[key] = future.result()

    f_r = results["reflection"]
    f_e = results["expansion"]
    f_oc = results["outside_contraction"]
    f_ic = results["inside_contraction"]

    # Nelder-Mead logic:
    # 1. If reflection is better than the best point, try expansion
    if f_r < scores[0]:
        if f_e < f_r:
            # Expansion is better than reflection
            simplex[-1] = candidates["expansion"]
            scores[-1] = f_e
            continue
        else:
            # Keep reflection
            simplex[-1] = candidates["reflection"]
            scores[-1] = f_r
            continue

    # 2. If reflection is not better than best, but better than second-worst, accept reflection
    elif f_r < scores[-2]:
        simplex[-1] = candidates["reflection"]
        scores[-1] = f_r
        continue

    # 3. If reflection is worse or equal to second-worst but better than worst, do outside contraction
    elif f_r < scores[-1]:
        if f_oc <= f_r:
            # Outside contraction improved over worst
            simplex[-1] = candidates["outside_contraction"]
            scores[-1] = f_oc
            continue
```

```python
        else:
            # Outside contraction didn’t improve, shrink
            simplex, scores = shrink_simplex(simplex, scores, gamma, f)
            continue

    # 4. Reflection is not better than worst, try inside contraction
        else:
        if f_ic < scores[-1]:
            # Inside contraction improved over worst
            simplex[-1] = candidates["inside_contraction"]
            scores[-1] = f_ic
            continue
        else:
            # Inside contraction didn’t improve, shrink
            simplex, scores = shrink_simplex(simplex, scores, gamma, f)
            continue
```

Code Listing C.1: Main part of the custom Nelder-Mead algorithm implementation. Full implementation is available on Github at https://github.com/buresjan/nelder_mead.

Bibliography

[1] F. Abraham, M. Behr, and M. Heinkenschloss, “Shape optimization in steady blood flow: A numerical study of non-Newtonian effects,” *Computer Methods in Biomechanics and Biomedical Engineering*, pp. 127–137, Apr. 2005.

[2] J. C. Weddell, J. Kwack, P. I. Imoukhuede, and A. Masud, “Hemodynamic Analysis in an Idealized Artery Tree: Differences in Wall Shear Stress between Newtonian and Non-Newtonian Blood Models,” *PLOS ONE*, p. e0124575, Apr. 2015.

[3] A. L. Marsden, J. A. Feinstein, and C. A. Taylor, “A computational framework for derivative-free optimization of cardiovascular geometries,” *Computer Methods in Applied Mechanics and Engineering*, pp. 1890–1905, Apr. 2008.

[4] S. Sharma, S. Goudy, P. Walker, S. Panchal, A. Ensley, K. Kanter, V. Tam, D. Fyfe, and A. Yoganathan, “In vitro flow experiments for determination of optimal geometry of total cavopulmonary connection for surgical repair of children with functional single ventricle,” *Journal of the American College of Cardiology*, pp. 1264–1269, Apr. 1996.

[5] A. E. Ensley, P. Lynch, G. P. Chatzimavroudis, C. Lucas, S. Sharma, and A. P. Yoganathan, “Toward designing the optimal total cavopulmonary connection: an in vitro study,” *The Annals of Thoracic Surgery*, pp. 1384–1390, Oct. 1999.

[6] F. M. Rijnberg, M. G. Hazekamp, J. J. Wentzel, P. J. de Koning, J. J. Westenberg, M. R. Jongbloed, N. A. Blom, and A. A. Roest, “Energetics of blood flow in cardiovascular disease,” *Circulation*, pp. 2393–2407, May 2018.

[7] V. Chaloupecký, “Nemocný s funkčně jedinou srdeční komorou,” FN Motol, Praha, 2004. [Online] Available at: http://pelikan.lf2.cuni.cz/2003-2004/kveten-cerven2004-12roc-cl6.htm, [Accessed: 13-11-2023].

[8] E. Rubtsova, A. Markov, S. Selishchev, J. H. Karimov, and D. Telyshev, “Mathematical modeling of the Fontan blood circulation supported with pediatric ventricular assist device,” *Computer Methods in Biomechanics and Biomedical Engineering*, pp. 653–662, Jan. 2021.

[9] Y. Delorme, K. Anupindi, A. Kerlo, D. Shetty, M. Rodefeld, J. Chen, and S. Frankel, “Large eddy simulation of powered Fontan hemodynamics,” *Journal of Biomechanics*, pp. 408–422, Jan. 2013.

[10] T. M. J. van Bakel, K. D. Lau, J. Hirsch-Romano, S. Trimarchi, A. L. Dorfman, and C. A. Figueroa, “Patient-Specific Modeling of Hemodynamics: Supporting Surgical Planning in a Fontan Circulation Correction,” *Journal of Cardiovascular Translational Research*, pp. 145–155, Apr. 2018.

[11] C. Wang, K. Pekkan, D. de Zélicourt, M. Horner, A. Parihar, A. Kulkarni, and A. P. Yoganathan, “Progress in the CFD Modeling of Flow Instabilities in Anatomical Total Cavopulmonary Connections,” *Annals of Biomedical Engineering*, pp. 1840–1856, Jul. 2007.

[12] A. Porfiryev, A. Markov, A. Galyastov, M. Denisov, O. Burdukova, A. Y. Gerasimenko, and D. Telyshev, “Fontan Hemodynamics Investigation via Modeling and Experimental Characterization of Idealized Pediatric Total Cavopulmonary Connection,” *Applied Sciences*, p. 6910, Oct. 2020.

[13] E. Tang, M. Restrepo, C. M. Haggerty, L. Mirabella, J. Bethel, K. K. Whitehead, M. A. Fogel, and A. P. Yoganathan, “Geometric characterization of patient-specific total cavopulmonary connections and its relationship to hemodynamics,” *JACC: Cardiovascular Imaging*, pp. 215–224, Mar. 2014.

[14] P. Eichler, “Mathematical modeling of fluid flow using lattice Boltzmann method,” Doctoral Thesis, FNSPE, CTU in Prague, 2023.

[15] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, and E. M. Viggen, *The lattice Boltzmann method: Principles and Practice*. Springer International Publishing, 2017.

[16] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations*. Society for Industrial and Applied Mathematics, Jan. 2007.

[17] R. J. LeVeque, *Cambridge texts in applied mathematics: Finite volume methods for hyperbolic problems series number 31*. Cambridge University Press, Aug. 2002.

[18] S. C. Brenner and R. Scott, *The mathematical theory of finite element methods*, Series: Texts in Applied Mathematics. Springer, Nov. 2010.

[19] J. Bureš, “Mathematical modeling of blood flow through vessels,” Bachelor’s Thesis (In Czech), FNSPE, CTU in Prague, 2022.

[20] J. Bureš, “Optimal shape design of walls of blood flow mathematical model focusing on the total cavopulmonary connection,” Research Project (In Czech), FNSPE, CTU in Prague, 2023.

[21] J. D. Anderson, *Computational Fluid Dynamics*, Series: McGraw-Hill Series in mechanical engineering. McGraw-Hill Professional, 1995.

[22] J. Latt, “Hydrodynamic limit of lattice Boltzmann equations,” Doctoral Thesis, Université de Genève, 2007.

[23] H. Schlichting and K. Gersten, *Boundary-layer theory*, 9th ed. Springer, Oct. 2016.

[24] Y. A. Cengel and J. M. Cimbala, *Fluid mechanics: Fundamentals and applications*, 4th ed. McGraw-Hill Education, 2017.

[25] L. D. Landau and E. M. Lifshits, *Fluid Mechanics*, 2nd ed. Elsevier Science, 2013.

[26] J. Sodja, “Turbulence models in CFD,” Master’s Thesis, University of Ljubljana, 2007.

[27] D. Saloner, “Computational fluid dynamics for evaluating hemodynamics,” *Vessel Based Imaging Techniques*, pp. 331–347. Springer International Publishing, Oct. 2019.

[28] C. S. Peskin, “The immersed boundary method,” *Acta Numerica*, pp. 479–517, Jan. 2002.

[29] L. Dempere-Marco, E. Oubel, M. Castro, C. Putman, A. Frangi, and J. Cebral, "CFD analysis incorporating the influence of wall motion: Application to intracranial aneurysms," *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2006*, pp. 438–445. Springer Berlin Heidelberg, 2006.

[30] J. Lantz, J. Renner, and M. Karlsson, "Wall shear stress in a subject specific human aorta — Influence of fluid-structure interaction," *International Journal of Applied Mechanics*, pp. 759–778, Dec. 2011.

[31] M. C. Fishbein and G. A. Fishbein, "Arteriosclerosis: facts and fancy," *Cardiovascular Pathology*, pp. 335–342, Nov. 2015.

[32] M. Syed and M. Lesch, "Coronary artery aneurysm: A review," *Progress in Cardiovascular Diseases*, pp. 77–84, Jul. 1997.

[33] P. Eichler, R. Galabov, R. Fučík, K. Škardová, T. Oberhuber, P. Pauš, J. Tintěra, and R. Chabiniok, "Non-Newtonian turbulent flow through aortic phantom: Experimental and computational study using magnetic resonance imaging and lattice Boltzmann method," *Computers and Mathematics with Applications*, pp. 80–94, Apr. 2023.

[34] J. Boyd, J. M. Buick, and S. Green, "Analysis of the Casson and Carreau-Yasuda non-Newtonian blood models in steady and oscillatory flows using the lattice Boltzmann method," *Physics of Fluids*, p. 093103, Sep. 2007.

[35] A. Sequeira and J. Janela, "An Overview of Some Mathematical Models of Blood Rheology," *A Portrait of State-of-the-Art Research at the Technical University of Lisbon*, pp. 65–87. Springer Netherlands, 2007.

[36] K. M. Saqr, S. Tupin, S. Rashad, T. Endo, K. Niizuma, T. Tominaga, and M. Ohta, "Physiologic blood flow is turbulent," *Scientific Reports*, Sep. 2020.

[37] B. A. Carabello and W. J. Paulus, "Aortic stenosis," *The Lancet*, pp. 956–966, Mar. 2009.

[38] K. Jain, "The effect of varying degrees of stenosis on transition to turbulence in oscillatory flows," *Biomechanics and Modeling in Mechanobiology*, pp. 1029–1041, Apr. 2022.

[39] S. S. Varghese and S. H. Frankel, "Numerical modeling of pulsatile turbulent flow in stenotic vessels," *Journal of Biomechanical Engineering*, pp. 445–460, Aug. 2003.

[40] M. V. Kameneva, G. W. Burgreen, K. Kono, B. Repko, J. F. Antaki, and M. Umezu, "Effects of turbulent stresses upon mechanical hemolysis: Experimental and computational analysis," *ASAIO Journal*, pp. 418–423, Sep. 2004.

[41] J. C. Masters, M. Ketner, M. S. Bleiweis, M. Mill, A. Yoganathan, and C. L. Lucas, "The Effect of Incorporating Vessel Compliance in a Computational Model of Blood Flow in a Total Cavopulmonary Connection (TCPC) with Caval Centerline Offset," *Journal of Biomechanical Engineering*, pp. 709–713, Dec. 2004.

[42] W. Orlando, R. Shandas, and C. DeGroff, "Efficiency differences in computational simulations of the total cavo-pulmonary circulation with and without compliant vessel walls," *Computer Methods and Programs in Biomedicine*, pp. 220–227, Mar. 2006.

[43] M. Geier, M. Schönherr, A. Pasquali, and M. Krafczyk, “The cumulant lattice Boltzmann equation in three dimensions: Theory and validation,” *Computers and Mathematics with Applications*, pp. 507–547, 2015.

[44] D. D’Humières, “Generalized Lattice-Boltzmann Equations,” *Rarefied Gas Dynamics: Theory and Simulations*, pp. 450–458. American Institute of Aeronautics and Astronautics, 1994.

[45] M. Geier, A. Greiner, and J. G. Korvink, “Cascaded digital lattice Boltzmann automata for high Reynolds number flow,” *Physical Review E*, 2006.

[46] I. V. Karlin, A. Ferrante, and H. C. Öttinger, “Perfect entropy functions of the Lattice Boltzmann method,” *Europhysics Letters (EPL)*, pp. 182–188, 1999.

[47] T. Oberhuber, J. Klinkovský, and R. Fučík, “TNL: Numerical library for modern parallel architectures,” *Acta Polytechnica*, pp. 122–134, Feb. 2021.

[48] J. Klinkovský, T. Oberhuber, R. Fučík, and V. Žabka, “Configurable Open-source Data Structure for Distributed Conforming Unstructured Homogeneous Meshes with GPU Support,” *ACM Transactions on Mathematical Software*, pp. 1–30, Sep. 2022.

[49] C. Geuzaine and J. Remacle, “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities,” *International Journal for Numerical Methods in Engineering*, pp. 1309–1331, May 2009.

[50] “STL (STereoLithography) File Format Family,” [Online] Available at: https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml, [Accessed 26-12-2024].

[51] Dawson-Haggerty, et al., “Trimesh 3.2.0,” [Online] Available at: https://trimesh.org/, [Accessed 23-11-2024].

[52] C. R. Harris, et al., “Array programming with NumPy,” *Nature*, pp. 357–362, Sep. 2020, [Accessed 11-11-2024].

[53] Python Software Foundation, “Concurrent Execution,” [Online] Available at: https://docs.python.org/3/library/concurrent.futures.html, 2024, [Accessed 15-11-2024].

[54] TNL Project, “PyTNL: Python bindings for the Template Numerical Library,” [Online] Available at: https://gitlab.com/tnl-project/pytnl, 2022, [Accessed 2024-11-21].

[55] P. Ramachandran and G. Varoquaux, “Mayavi: 3D Visualization of Scientific Data,” *Computing in Science & Engineering*, pp. 40–51, 2011.

[56] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for optimization*. MIT Press, Mar. 2019.

[57] R. Fletcher and M. J. D. Powell, “A Rapidly Convergent Descent Method for Minimization,” *The Computer Journal*, pp. 163–168, Aug. 1963.

[58] C. G. Broyden, “The Convergence of a Class of Double-rank Minimization Algorithms,” *IMA Journal of Applied Mathematics*, pp. 222–231, 1970.

[59] C. Audet and W. Hare, *Derivative-free and blackbox optimization*, 1st ed., Series: Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Dec. 2017.

[60] D. Bertsekas, *Nonlinear programming*. Athena Scientific, Sep. 2016.

[61] D. G. Luenberger and Y. Ye, *Linear and nonlinear programming*, 3rd ed., Series: International series in operations research & management science. Springer, Jul. 2008.

[62] J. Larson, M. Menickelly, and S. M. Wild, "Derivative-free optimization methods," *Acta Numerica*, pp. 287–404, May 2019.

[63] S. Alarie, C. Audet, A. E. Gheribi, M. Kokkolaras, and S. L. Digabel, "Two decades of blackbox optimization applications," *EURO Journal on Computational Optimization*, p. 100011, 2021.

[64] O. Kramer, D. E. Ciaurri, and S. Koziel, "Derivative-free optimization," *Computational Optimization, Methods and Algorithms*, pp. 61–83. Springer Berlin Heidelberg, 2011.

[65] G. B. Dantzig, "Origins of the simplex method," *ACM*, pp. 141–151, Jun. 1990.

[66] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, pp. 308–313, Jan. 1965.

[67] C. Audet and J. E. Dennis, "Analysis of Generalized Pattern Searches," *SIAM Journal on Optimization*, pp. 889–903, Jan. 2002.

[68] C. Audet and J. E. Dennis, "Mesh Adaptive Direct Search Algorithms for Constrained Optimization," *SIAM Journal on Optimization*, pp. 188–217, Jan. 2006.

[69] C. Audet, S. Le Digabel, V. Rochon Montplaisir, and C. Tribes, "Algorithm 1027: NOMAD version 4: Nonlinear optimization with the MADS algorithm," *ACM Transactions on Mathematical Software*, pp. 35:1–35:22, 2022.

[70] M. Bouzidi, M. Firdaouss, and P. Lallemand, "Momentum transfer of a Boltzmann-lattice fluid with boundaries," *Physics of Fluids*, pp. 3452–3459, 2001.

[71] K. S. Sakariassen, L. Orning, and V. T. Turitto, "The Impact of Blood Shear Rate On Arterial Thrombus Formation," *Future Science OA*, Jul. 2015.

[72] S. K. Rajagopal and J. R. Fineman, "Pulmonary Arteriovenous Malformations and the Hepatic 'Black Box'," *JACC: Basic to Translational Science*, pp. 236–238, Mar. 2021.

[73] R. Krams and M. Bäck, *The ESC Textbook of Vascular Biology*, Series: European Society of Cardiology publications. Oxford University Press, Mar. 2017.

[74] K.-I. Tsubota, H. Sonobe, K. Sughimoto, and H. Liu, "Blood Flow Simulation to Determine the Risk of Thrombosis in the Fontan Circulation: Comparison between Atriopulmonary and Total Cavopulmonary Connections," *Fluids*, p. 138, Apr. 2022.

[75] H. Wei, A. L. Cheng, and N. M. Pahlevan, "On the significance of blood flow shear-rate-dependency in modeling of Fontan hemodynamics," *European Journal of Mechanics - B/Fluids*, pp. 1–14, Nov. 2020.

[76] A. L. Cheng, N. M. Pahlevan, D. G. Rinderknecht, J. C. Wood, and M. Gharib, "Experimental investigation of the effect of non-Newtonian behavior of blood flow in the Fontan circulation," *European Journal of Mechanics - B/Fluids*, pp. 184–192, Mar. 2018.

[77] G. Dubini, M. de Leval, R. Pietrabissa, F. Montevecchi, and R. Fumero, "A numerical fluid mechanical study of repaired congenital heart defects. Application to the total cavopulmonary connection," *Journal of Biomechanics*, pp. 111–121, Jan. 1996.

[78] C. G. DeGroff, “Modeling the Fontan Circulation: Where We Are and Where We Need to Go,” *Pediatric Cardiology*, pp. 3–12, Oct. 2007.

[79] A. Amodeo, M. Grigioni, G. D’Avenio, C. Daniele, and R. M. Di Donato, “The patterns of flow in the total extracardiac cavopulmonary connection,” *Cardiology in the Young*, pp. 53–56, Oct. 2004.

[80] K. Pekkan, H. D. Kitajima, D. de Zelicourt, J. M. Forbess, W. J. Parks, M. A. Fogel, S. Sharma, K. R. Kanter, D. Frakes, and A. P. Yoganathan, “Total Cavopulmonary Connection Flow With Functional Left Pulmonary Artery Stenosis: Angioplasty and Fenestration In Vitro,” *Circulation*, pp. 3264–3271, Nov. 2005.

[81] “Hardware Overview – GPX cluster,” [Online] Available at: https://mmg-gitlab.fjfi.cvut.cz/gitlab/mmg/gpx-cluster/-/blob/master/doc/hardware-overview.md?ref_type=heads, [Accessed 19-12-2024].

[82] M. Jette, C. Dunlap, J. Garlick, and M. Grondona, “SLURM: Simple Linux Utility for Resource Management,” *Job Scheduling Strategies for Parallel Processing*, pp. 44–60. Springer Berlin Heidelberg, Jul. 2002.

[83] J. Bureš, “IVC flow split automation using Paraview,” [Online] Available at: https://gist.github.com/buresjan/074dd23c6cc790a79157a56c100f4e7c, 2024, [Accessed 30-12-2024].

[84] M. S. Chong, A. E. Perry, and B. J. Cantwell, “A general classification of three-dimensional flow fields,” *Physics of Fluids A: Fluid Dynamics*, pp. 765–777, May 1990.