



TECHNISCHE  
UNIVERSITÄT  
WIEN



## Master Thesis

# Implementation and validation of a spring-network model for fluid-structure interaction

carried out for the purpose of obtaining the degree of Master of Science (MSc)  
under the supervision of

**Univ.Prof. Dr.-Ing. Manuel García-Villalba**

**Dr. Manuel Moriche**

Institute of Fluid Mechanics and Heat Transfer, E322

submitted at TU Wien

**Faculty of Mechanical and Industrial Engineering**

by

**Benjamin Rigler, BSc**

Mat.No.: 51841531

*Affidavit:*

I declare in lieu of oath, that I wrote this thesis and carried out the associated research myself, using only the literature cited in this volume. If text passages from sources are used literally, they are marked as such.

I confirm that this work is original and has not been submitted for examination elsewhere, nor is it currently under consideration for a thesis elsewhere. I acknowledge that the submitted work will be checked electronically-technically using suitable and state-of-the-art means (plagiarism detection software). On the one hand, this ensures that the submitted work was prepared according to the high-quality standards within the applicable rules to ensure good scientific practice "Code of Conduct" at the TU Wien. On the other hand, a comparison with other student theses avoids violations of my personal copyright.

---

Place/Date

---

Signature

---

# Acknowledgements

I would like to sincerely thank my supervisors, Prof. Manuel Garcia-Villalba and Dr. Manuel Moriche, for their dedicated support, valuable feedback and helpful suggestions during the course of this work.

Also, a big thank you to my family, especially my mother Anita, who always supported me throughout the course of my studies.

---

# Abstract

Fluid flow around thin, flexible membranes occurs in various problems, from understanding the behaviour of Malaria-infected red blood cells or circulating tumor cells to investigating the aerodynamics of flapping, flexible wings of birds, insects and bats. Solvers for fluid-structure interaction require a structural dynamics solver and a fluid dynamics solver. Spring-network models provide a computationally cheap alternative for describing the structural dynamics of thin membranes in comparison to more complex finite element methods. The objective of this work is to implement a spring-network model to simulate thin, flexible, open membranes in a fluid-structure interaction context. The model is coupled with an in-house computational fluid mechanics solver based on a direct-forcing immersed boundary method.

The spring-network model is implemented in C++, for which 2D and 3D mesh data structures have been defined. The spring-network model uses linear and bending springs to model the physics of open membranes. The code is structured such that easy adaptation is possible, so new models can readily be implemented.

The code is validated as a standalone version by simulating the damped and undamped behaviours of a cantilever beam in different configurations. Furthermore, the coupled solver is used to simulate a flexible filament in 2D and a flexible flag in 3D. Comparisons with the literature show issues with the coupling of the spring-network model with the fluid dynamics solver.

Possible solutions, potential improvements and future extensions are presented in the end.

---

# Kurzfassung

Die Strömung um dünne, elastische Membranen tritt in verschiedensten Problemstellungen auf, angefangen beim Verständnis des Verhaltens von mit Malaria infizierten roten Blutkörperchen oder zirkulierenden Tumorzellen bis hin zur Untersuchung der Aerodynamik des Flügelschlags von Vögeln, Insekten und Fledermäusen. Löser für Fluid-Struktur-Interaktion benötigen einen Löser für die Strukturmechanik und einen Löser für die Strömungsmechanik. Feder-Netzwerkmodelle bieten eine rechnerisch weniger aufwendige Alternative zur Beschreibung der Strukturmechanik dünner, elastischer Membranen im Vergleich zu komplexeren Finite-Elemente-Methoden.

Ziel dieser Arbeit ist die Implementierung eines Feder-Netzwerkmodells zur Modellierung dünner, flexibler, offener Membranen im Kontext von Fluid-Struktur-Interaktion. Dazu wird das Feder-Netzwerkmodell mit einem Löser für numerische Strömungsmechanik, der auf einer Immersed Boundary Methode basiert, gekoppelt.

Das Feder-Netzwerkmodell ist in C++ implementiert, wozu Datenstrukturen für 2D und 3D Rechengitter entwickelt wurden. Das Modell verwendet lineare und Biegefedern zur Modellierung von offenen Membranen. Der Code ist dafür ausgelegt, einfach erweitert werden zu können.

Der Code ist in einer ungekoppelten Version anhand des gedämpften und ungedämpften Verhaltens eines Auslegerbalkens in verschiedenen Konfigurationen validiert. Desweiteren wurden mit der gekoppelten Version Simulationen eines flexiblen Filaments in 2D und einer flexiblen Flagge in 3D durchgeführt. Ein Vergleich der Ergebnisse mit Literatur lässt erkennen, dass die Kopplung des Feder-Netzwerkmodells mit dem numerischen Strömungsmechanik-Löser nicht funktioniert.

Abschließend werden mögliche Lösungsvorschläge, potentielle Verbesserungen und künftige Erweiterungen präsentiert.

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Simulation strategies . . . . .	1
1.1.1	Immersed-boundary method . . . . .	1
1.1.2	Spring-network model . . . . .	2
1.1.3	Combining the immersed-boundary method and spring-network models	2
1.2	Applications . . . . .	3
1.3	Thesis scope . . . . .	4
<b>2</b>	<b>Theory</b>	<b>5</b>
2.1	Mesh . . . . .	5
2.2	Solid mechanics . . . . .	6
2.3	Spring-network models and the minimum energy principle . . . . .	9
2.4	Relation between continuum mechanics and spring-network models . . . . .	11
2.5	Fluid mechanics . . . . .	12
2.5.1	Eulerian vs. Lagrangian description . . . . .	12
2.5.2	Navier-Stokes equations . . . . .	13
2.6	Immersed-boundary method . . . . .	13
<b>3</b>	<b>Methodology</b>	<b>15</b>
3.1	Discretization . . . . .	15
3.2	Forces . . . . .	15
3.2.1	Linear springs . . . . .	15
3.2.2	Bending springs in 2D . . . . .	17
3.2.3	Bending springs in 3D . . . . .	20
3.3	Force transfer . . . . .	21
3.4	Integration . . . . .	21
3.5	Implementation . . . . .	22

<b>4</b>	<b>Results and Discussion</b>	<b>27</b>
4.1	Cantilever beam . . . . .	27
4.1.1	Damped case . . . . .	27
4.1.2	Undamped case . . . . .	28
4.2	Flexible filament . . . . .	29
4.3	Flexible flag . . . . .	31
<b>5</b>	<b>Conclusion and Outlook</b>	<b>39</b>
	<b>List of Figures</b>	<b>42</b>
	<b>List of Tables</b>	<b>44</b>
	<b>List of Listings</b>	<b>45</b>
	<b>Nomenclature</b>	<b>46</b>
	<b>Bibliography</b>	<b>49</b>
<b>A</b>	<b>Bending forces</b>	<b>55</b>
A.1	Bending forces in 2D . . . . .	55

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
 The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Chapter 1

## Introduction

The FSI (fluid-structure interaction) of thin membranes occurs in various scientific disciplines, for example, in modelling the flow of blood [1][2] or the aerodynamics of flapping bat, bird and insect wings [3][4]. A standard approach for high fidelity is to describe the structural dynamic behaviour using finite element formulations [5][6][7], although this comes at the cost of high computational effort. An alternative, less computationally demanding approach is to use a SNM (spring-network model). In this chapter an overview of different strategies for modelling FSI of thin membranes is presented and some motivation for the use of SNMs is provided through brief overviews of two applications.

### 1.1 Simulation strategies

Over time, various strategies have emerged to model thin, flexible membranes. These strategies can be divided into two main categories, based on the type of mesh used. The first category uses body-fitted grids. These can be either structured or unstructured [6][8][9]. When the membrane moves or deforms, the computational grid discretizing the fluid domain must be recomputed to fit the new geometry, and the pressure and velocity fields of the previous mesh must be projected onto the new mesh, which is computationally expensive. In contrast, the IBM (immersed boundary method) does not require any recomputations, and therefore provide a computationally less expensive alternative. Furthermore, the procedure to generate a body-fitted grid is problem dependent, which makes IBMs even more attractive to simulate arbitrary-shaped and/or moving bodies. Overviews of these methods can be found in Mittal and Iaccarino [10] or Kim and Choi [11].

#### 1.1.1 Immersed-boundary method

In the IBM, the partial differential equations that govern the fluid are solved over the entire domain, including solid objects, allowing the use of simple Cartesian grids. Besides of not requiring the expensive procedure of generating problem-specific meshes, Cartesian grids also have the advantage of allowing the use of high-performance algorithms, especially when dealing with Poisson problems, which are often the bottleneck when solving incompressible

flows. The original IBM, proposed by Peskin [12], uses a monolithic approach where the NS (Navier-Stokes) equations are solved using a singular force function. The immersed boundary is discretized into Lagrangian markers. The force function along the boundary can be arbitrarily defined to best model the structural behavior of the boundary. Lagrangian markers do not need to coincide with nodes of the Eulerian grid. Therefore, the force function is distributed to neighbouring Eulerian grid points using a regularized discrete version of a Dirac delta function. The solution procedure consists of two parts per timestep. First, the NS equations are solved, and then the Lagrangian markers are advected in time by the local velocity. For example, Fauci and Peskin [13] used this approach to simulate aquatic animal locomotion. Beyer [14] used a similar version to Peskin [12], where springs between the Lagrangian markers and equilibrium positions are included. This approach requires additional parameters for spring stiffness or damping, and possibly imposes strong limitations on the time step size for stiff problems. Goldstein, Handler, and Sirovich [15] used a force feedback approach, which also has strict limitations on the timestep size. Mohd-Yusof [16] developed a direct-forcing approach in combination with spectral methods. Fadlun *et al.* [17] used the same approach in combination with finite difference methods. When using direct-forcing, the no-slip boundary condition between the immersed boundary and fluid is explicitly enforced. Uhlmann [18] further improved the method by computing the forcing at the Lagrangian markers. The approach used by Vanella and Balaras [19] is based on the work of Uhlmann [18], but uses a moving-least-squares reconstruction instead of a discrete Dirac delta function.

### 1.1.2 Spring-network model

SNMs were first proposed in the context of computer graphics applications. Provot [20] developed a model for regular grids with predefined spring stiffness factors. Later Volino and Thalmann [21] developed a model suitable for irregular grids. Gelder [22] compared SNMs to continuum models and found an improved way for calculating spring stiffness based on the geometry of the grid. SNMs are also used in biomedical engineering to simulate the mechanical behaviour of cells [23][1] or for surgical simulations [24][25]. SNMs can also be adapted to so-called rigid-body-spring networks for fracture prediction [26][27][28].

### 1.1.3 Combining the immersed-boundary method and spring-network models

Tullio and Pascazio [29] combined the IBM with an SNM to simulate arbitrarily shaped, thin surfaces that can be open or closed and have shown its capabilities based on various simulations like flow around a flexible filament, flow around a flexible flag or flow through a bio-prosthetic aortic valve. Spandan *et al.* [30] used a similar approach with a focus on liquid-liquid interface dynamics, like drops or bubbles. Coclite *et al.* [31] investigated the tumbling motion of deformable particles using an SNM combined with an LBM (Lattice Boltzmann

method) using a dynamic IBM. Afra *et al.* [32] used an LBM-based IBM to simulate various configurations of flexible filaments. Kumar, Seo, and Mittal [3] used a sharp-interface IBM with an SNM to simulate the wings of a bat. Zhang *et al.* [33] used an LBM-based IBM simulating the behaviour of tumor cells in constricted microvessels.

## 1.2 Applications

Blood is a suspension with non-Newtonian characteristics and contains a high concentration of red blood cells, white blood cells, platelets, blood plasma and other chemical elements. Although plasma has Newtonian characteristics, red blood cells, which make up about 45 % of the blood volume, create non-Newtonian characteristics. Understanding blood flow can help with the development of new drugs and treatments.[34]

Blood in vessels with a characteristic diameter larger than 1 *mm* is often modelled as a Newtonian fluid with constant viscosity, for a diameter smaller than 1 *mm*, non-Newtonian models are used. For diameters smaller than 100 - 200  $\mu\text{m}$  explicit modelling of red blood cells is required. The deformability of red blood cells is an important property because it allows them to pass through small vessels. Therefore, deformability needs to be considered in computational models.[34]

Using SNMs allows for modelling the different components of red blood cells with little computational expense. The elastic deformations of the spectrin network, formed by spectrin proteins linked by short filaments are modelled by in-plane elastic energy and the stiffness of the lipid bilayer by bending energy. Furthermore, additional energy terms can be defined to account for the area preserving properties of the lipid bilayer and the incompressibility of the internal cytosol.[34]

In the Malaria disease, red blood cells get infected by a parasite called Plasmodium falciparum. This parasite causes a significant change in the mechanical and rheological properties of red blood cells. As an example, Fedosov [34] used an SNM to quantify the increase in blood flow resistance in malaria-infected blood.

Another blood flow related problem is the modelling of tumors. Mature tumor cells can detach from their primary site and enter the circulatory system. These cells, known as CTCs (circulating tumor cells), can lead to the formation of secondary tumors. This process, called hematogenous metastasis, is a major cause of treatment failure[33]. Zhang *et al.* [33] used a SNM combined with an LBM-based IBM to determine key factors contributing to the adhesion of CTC in microvessels, which can help to understand the cancer metastasis process.

A second area of interest is the investigation of aero-structural interactions of deformable flapping wings. Understanding the complex behaviour of wings of bats, birds and insects is of importance due to their lower requirement in energy to produce lift [35]. Therefore, understanding the complex behaviour of flexible wings can help with the design of micro-aerial vehicles [36]. Kumar, Seo, and Mittal [3] developed a model of a bat wing, where the

wing is modeled as an elastic membrane and is divided into multiple segments similar to an actual bat wing. A strong difference in the effectiveness of generating lift, drag and thrust forces between different sections of the wing has been shown. Furthermore, it was shown that even though the mean drag does not significantly change upon comparing flexible with rigid wings, the lift of a flexible wing is almost twice as large as compared with rigid wings. Another example is the investigation of bumblebee wings by [4].

## 1.3 Thesis scope

Within the scope of this thesis, a code will be implemented to model the dynamic structural behaviour of thin, open membranes in both 2D and 3D using SNMs. The model should support both in-plane and out-of-plane forces. The solver will not be limited to a specific structural model, so the code will be designed to easily vary the computation of forces. To test the SNM in a FSI context, the code will be coupled with the serial 2D and 3D versions of the IBM solver *TXPS*, which is currently under development at the Institute of Fluid Mechanics and Heat Transfer at TU Wien. *TXPS* is based on the method proposed by Uhlmann [18]. The model is evaluated using three different test cases. First, a simple cantilever beam will be simulated and validated in reference to existing literature. For this purpose, a standalone version is used. The coupled solver will be validated by simulating a flexible filament in 2D and a flexible flag in 3D and comparing the results to existing literature. At the end, we will draw some conclusions and propose lines of work for the future.

### 2.1 Mesh

Meshes or computational grids are used to represent a domain or object in a discrete fashion. They are used in various applications, ranging from scientific computing to computational geometry to computer graphics. A grid contains two building blocks: sets of entities like vertices, edges and faces and the topology explaining how the entities are related to each other. Meshes can be classified as either structured or unstructured.

Structured grids use indices  $i, j, k$  to define both entities and topology in one, two or three dimensions, respectively. They can represent a simple rectangular grid, where simple heuristics can be employed to compute the coordinates of the vertices. Furthermore, the neighbouring vertices are directly known from the indices, so all the necessary topological information is embodied in the index set. This leads to simple implementations and high efficiency of the underlying application, with the pitfall of only allowing the modelling of simple geometries. More complex geometries can be discretized by employing body-fitted grids, with the disadvantage of increased complexity, as the problem must be rewritten in the body-fitted coordinate system.

Unstructured grids allow a more general approach to generate the mesh defining the geometry, at the cost of being more computationally expensive in the simulation phase. The entities have to be explicitly stored, and additionally, the topology connecting the entities with each other has to be known. Corresponding data structures have to be designed considering parameters like robustness, generality, portability, coding efficiency, maintainability and difficulty of usage [37]. Mesh data structures can generally be divided into *edge-based* or *face-based* data structures [38].

*Edge-based* data structures are commonly used in computer graphics applications. Examples are the *Winged-Edge* [39], *Doubly Connected Edge List* [40], *Quad-Edge* [41] or *Half-Edge* [42] data structures. These data structures typically store topological information per edge, like the two adjacent faces or pointers to the next and/or previous edges.

In *face-based* data structures, faces store pointers to their corresponding vertices. For fast data access, additional topological information may be stored, like the adjacent faces of a face. Examples for *face-based* data structures are *CGAL* [43] or *Triangle* [44].

## 2.2 Solid mechanics

In this chapter, some fundamental principles of the mechanics of solids shall be presented. Introductions can be found, for example, in Tanaka, Wada, and Nakamura [45] or Bhavikatti [46].

Stress at a point is defined as

$$\sigma = \lim_{\Delta A \rightarrow 0} \frac{\Delta F}{\Delta A} = \frac{dF}{dA}, \quad (2.1)$$

where  $F$  is a force and  $A$  is the area on which the load is applied. If a force is applied on an object, the object deforms, which can be measured in strain, which is defined as

$$\varepsilon = \frac{\Delta L}{L_0}, \quad (2.2)$$

where  $\Delta L$  is the change in length and  $L_0$  is the length before a load gets applied.

Stress and strain can be related via Hooke's law, which states that stress and strain are proportional up to the elastic limit and is written as

$$\sigma = E\varepsilon, \quad (2.3)$$

where  $E$  is the Young's modulus, which is a property of the material. If the length of an object changes (e.g. a rod gets stretched due to an applied load), the object's size in the other dimensions is also going to change (e.g. the diameter of the rod decreases), even if there is no force applied in the respective direction. This relationship is denoted by the Poisson's ratio

$$\nu = \frac{-\varepsilon_{\text{trans}}}{\varepsilon_{\text{axial}}}, \quad (2.4)$$

where  $\varepsilon_{\text{axial}}$  is the strain in the load's direction and  $\varepsilon_{\text{trans}}$  in the corresponding normal directions. In three dimensions, stress can be written as

$$\sigma_{ij} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix}. \quad (2.5)$$

Symmetries  $\sigma_{xy} = \sigma_{yx}$ ,  $\sigma_{xz} = \sigma_{zx}$  and  $\sigma_{yz} = \sigma_{zy}$  can be used based on a requirement in equilibrium in torque, reducing the tensor to six components. Displacement is defined as the

difference in distance between a point in the undeformed object and the same point in the deformed object

$$u_i = x_i - X_i, \quad (2.6)$$

where  $x_i$  is the position of point  $i$  in the deformed object and  $X_i$  is the position in the undeformed object. The length of a line element in the undeformed object can be defined as  $ds_0^2 = dX_i dX_i$  and in the deformed object as  $ds^2 = dx_i dx_i$ . These distances can be related by the Green strain tensor  $E_{ij}$  or Almansi strain tensor  $e_{ij}$  by

$$ds^2 - ds_0^2 = 2E_{ij}dX_i dX_j = 2e_{ij}dx_i dx_j, \quad (2.7)$$

where

$$E_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial X_j} + \frac{\partial u_j}{\partial X_i} + \frac{\partial u_k}{\partial X_i} \frac{\partial u_k}{\partial X_j} \right) \quad (2.8)$$

and

$$e_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} + \frac{\partial u_k}{\partial x_i} \frac{\partial u_k}{\partial x_j} \right). \quad (2.9)$$

For small deformations the higher order terms can be neglected and both strain tensors can be written as

$$\varepsilon_{ij} = \frac{1}{2} \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right). \quad (2.10)$$

Hooke's law in three dimensions can be written as

$$\sigma_{ij} = C_{ijkl} \varepsilon_{kl}, \quad (2.11)$$

where  $C_{ijkl}$  is a tensor of rank 4 of constant coefficients relating stress with strain. For isotropic materials the tensor can be written as

$$C_{ijkl} = \lambda \delta_{ij} \delta_{kl} + \mu (\delta_{ik} \delta_{jl} + \delta_{il} \delta_{jk}), \quad (2.12)$$

where  $\delta$  denotes the Kronecker- $\delta$  and  $\lambda$  and  $\mu$  are Lamé's constants, which are defined as

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad (2.13a)$$

$$\mu = \frac{E}{2(1+\nu)}. \quad (2.13b)$$

It has to be noted that the Young modulus in 1D has to be scaled under the assumption of no strain in any direction except in axial direction [47]. To show this relationship, the expressions for strain in two dimensions can be written as

$$\varepsilon_x = \frac{\sigma_x}{E} - \nu \frac{\sigma_y}{E}, \quad (2.14a)$$

$$\varepsilon_y = -\nu \frac{\sigma_x}{E} + \frac{\sigma_y}{E}. \quad (2.14b)$$

Assuming there is no strain in the  $y$ -direction, 2.14 can be rearranged into

$$\varepsilon_x = \frac{\sigma_x}{E} (1 - \nu^2). \quad (2.15)$$

Further rearranging yields the relationship

$$\tilde{E} = \frac{E}{(1 - \nu^2)}, \quad (2.16)$$

where  $\tilde{E}$  is the scaled Young modulus. Some further material parameters are the bulk modulus and shear modulus. Bulk modulus quantifies the change in volume due to a constant pressure acting in all directions as

$$K = \frac{p}{\frac{\Delta V}{V_0}} \quad (2.17)$$

where  $p$  is the pressure and  $\Delta V$  the change in the object's volume and  $V_0$  the object's initial volume. The shear modulus is defined as the ratio between shearing stress  $q$  and shearing strain  $\phi$  as

$$G = \frac{q}{\phi}. \quad (2.18)$$

The bulk and shear modulus can be related to the Young's modulus and Poisson's ratio for isotropic materials by

$$K = \frac{E}{3(1 - 2\nu)} \quad (2.19)$$

and

$$G = \frac{E}{2(1 + \nu)}. \quad (2.20)$$

## 2.3 Spring-network models and the minimum energy principle

A brief summary of SNMs and the minimum energy principle is given here. Let's consider a simple linear spring with spring constant  $k_e$ , a displacement  $x$  of one end from its relaxed position and where the other end is fixed. Then the force  $F$  acting on the spring is

$$F^{\text{ext}} = k_e x. \quad (2.21)$$

The elastic energy stored in the spring can be computed as

$$W^e = \int_0^x F dx = \frac{1}{2} k_e x^2. \quad (2.22)$$

The internal force in the spring can then be computed using the elastic energy by

$$F^{\text{int}} = -\frac{dW^e}{dx}. \quad (2.23)$$

In SNMs, the membrane to be modelled is discretized into nodes, such that the mass is distributed equally over all nodes. These nodes are connected with springs, such that a mesh of edges in 2D and a mesh of triangles in 3D is created. In two or three dimensions, the external forces of a linear spring between two nodes can be written as

$$\mathbf{F}_1^{\text{ext}} = k_e (L - L_0) \frac{\mathbf{r}_{12}}{L}, \quad (2.24a)$$

$$\mathbf{F}_2^{\text{ext}} = k_e (L - L_0) \frac{\mathbf{r}_{21}}{L}, \quad (2.24b)$$

where  $\mathbf{r}_i$  is the position vector of node  $i$ ,  $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$  and  $L = |\mathbf{r}_{ij}|$ . The internal forces can

be computed by

$$\mathbf{F}^{\text{int}} = -\frac{\partial W^e}{\partial \mathbf{r}_i}. \quad (2.25)$$

Different types of springs may be considered in addition to or instead of linear springs. A common possibility is a bending spring between two adjacent edges or triangles, which is discussed in more detail in chapters 3.2.2 and 3.2.3. Additionally, forces derived from any differentiable energy function may be used. For example, when modelling closed membranes, often an energy function based on the change of triangle area is used, which can be defined as

$$W^a = \frac{1}{2} k_a \left( \frac{A - A_0}{A_0} \right)^2 A_0 \quad (2.26)$$

with

$$A = \frac{1}{2} |(\mathbf{r}_2 - \mathbf{r}_1) \times (\mathbf{r}_3 - \mathbf{r}_1)|, \quad (2.27)$$

where  $\mathbf{r}_1$ ,  $\mathbf{r}_2$  and  $\mathbf{r}_3$  are the vertices of a triangle,  $A_0$  is the relaxed triangle area and  $k_a$  is a material parameter.

Assuming body forces  $\mathbf{G}_i$  and surface forces  $\mathbf{P}_i$  acting on a node  $i$  are conservative, the total potential energy of a system can be written as

$$\mathbf{G}_i = -\frac{\partial \Phi_G}{\partial \mathbf{r}_i}, \quad (2.28)$$

$$\mathbf{P}_i = -\frac{\partial \Phi_P}{\partial \mathbf{r}_i}, \quad (2.29)$$

$$\Pi = W + \Phi_G + \Phi_P, \quad (2.30)$$

where  $W$  is the total elastic energy (linear springs, bending springs, area elasticity, etc.). It can be shown that when the system is at equilibrium, the total potential energy is at a minimum by writing

$$0 = \mathbf{F}_i + \mathbf{P}_i + \mathbf{G}_i = -\frac{\partial \Pi}{\partial \mathbf{r}_i}. \quad (2.31)$$

For the system to achieve equilibrium, the mass points have to be moved to reach minimum potential energy with respect to the motion equation

$$m_i \frac{d^2 \mathbf{r}_i}{dt^2} + \gamma \frac{d\mathbf{r}_i}{dt} = \mathbf{F}_i. \quad (2.32)$$

where  $m_i$  is the mass of a mass point and  $\gamma$  is a damping factor. The forces can be computed by

$$\mathbf{F}_i = -\frac{\partial \Pi}{\partial \mathbf{r}_i}. \quad (2.33)$$

Equation 2.33 can then be solved using a numerical integration method. A detailed introduction to the minimum energy principle is given by Tanaka, Wada, and Nakamura [45]. A brief summary is given here.

## 2.4 Relation between continuum mechanics and spring-network models

Gelder [22] has shown that, in general, an exact simulation is impossible by comparing equations derived using a SNM with a finite element approach based on the constant strain model for a two-dimensional planar membrane with constant thickness. He also proposed a formula for the computation of spring coefficients of the edges, such that the triangles discretizing the membrane deform in the same manner as an isotropic elastic membrane when second and higher order effects are neglected, based on the mesh geometry. In this context, the general equation for the spring stiffness of an edge is written as

$$k_e = \left( \frac{Eh}{1+\nu} \right) \frac{A_T}{L^2} + \left( \frac{Eh\nu}{1-\nu^2} \right) \frac{(L_a^2 + L_b^2 - L^2)}{8A_T}, \quad (2.34)$$

where  $E$  is the Young's modulus,  $\nu$  the Poisson's ratio,  $h$  the membrane thickness,  $A_T$  the triangle area,  $L$  the edge length and  $L_a$  and  $L_b$  are the lengths of the remaining triangle's edges. For  $\nu \neq 0$  the formula can yield unphysical negative values for obtuse triangles, so commonly a version assuming  $\nu = 0$  is used. The final equation, taking into account both triangles adjacent to an edge, reads

$$k_e = \frac{Eh \sum_i A_i}{L^2}, \quad (2.35)$$

where  $\sum_i A_i$  is the sum of areas of all adjacent triangles. Lloyd, Szekely, and Harders [48] used an approach based on a comparison between a constant strain triangle finite element formulation and a linearized SNM, and found

$$k_e = \sum_i Eh \frac{\sqrt{3}}{4} \quad (2.36)$$

for the spring stiffness, assuming  $\nu = \frac{1}{3}$  and the sum is again over all triangles adjacent to an edge. It has to be noted that for equilateral triangles the formulas 2.35 and 2.36 are equivalent.

The spring coefficient for bending springs can be derived by equating the bending spring energy to a reference energy. Fedosov, Caswell, and Karniadakis [1] derived a bending stiffness based on the total curvature elastic energy [49] defined as

$$W = \frac{B}{2} \int_A (C_1 + C_2 - 2C_0)^2 dA + k_g \int_A C_1 C_2 dA, \quad (2.37)$$

where  $B$  and  $k_g$  are the bending rigidities,  $C_1$  and  $C_2$  are the local principal curvatures and  $C_0$  is the spontaneous curvature. Integration over a sphere and comparison with the energy due to bending springs yields

$$k_b = B \frac{2}{\sqrt{3}}. \quad (2.38)$$

## 2.5 Fluid mechanics

In this chapter, the different descriptions used in fluid mechanics and the NS equations are briefly introduced. Detailed explanations can be found, for example, in Tanaka, Wada, and Nakamura [45] or Moukalled, Mangani, and Darwish [50].

### 2.5.1 Eulerian vs. Lagrangian description

In fluid mechanics, an Eulerian or Lagrangian description can be used. In an Eulerian description, changes in properties like velocity and pressure are looked at at fixed points in space. The Eulerian description is a field description, where flow is expressed as a function of the coordinates and time:

$$\mathbf{u} = \mathbf{u}(x, y, z, t). \quad (2.39)$$

The acceleration can be calculated using the material derivative as

$$\frac{D\mathbf{u}}{Dt} = \frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u}. \quad (2.40)$$

In a Lagrangian description, particles are followed. The position of a particle is given as a function of the initial position and time as

$$\mathbf{r} = \mathbf{r}(\mathbf{r}_0, t). \quad (2.41)$$

The velocity and acceleration of particles can simply be written as

$$\mathbf{u} = \frac{\partial\mathbf{r}}{\partial t} \quad (2.42)$$

and

$$\mathbf{a} = \frac{\partial^2\mathbf{r}}{\partial t^2}. \quad (2.43)$$

### 2.5.2 Navier-Stokes equations

Fluid flow in an Eulerian description is governed by conservation equations for mass, momentum and energy. Since incompressibility is assumed, i.e. constant density, the energy transport equation can be dropped. Then the NS equations for Newtonian fluids with constant density,  $\rho_f$ , and viscosity,  $\nu$ , read:

$$\partial_t\mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \frac{\nabla p}{\rho_f} = \nu\nabla^2\mathbf{u} + \mathbf{f} \quad (2.44a)$$

$$\nabla \cdot \mathbf{u} = 0 \quad (2.44b)$$

where  $\mathbf{u}$  are the fluid velocities,  $p$  the pressure and  $\mathbf{f}$  the present external forces per unit mass.

## 2.6 Immersed-boundary method

The IBM in the solver used in this work is based on the work of Uhlmann [18] and shall be summarized in the following. Fluid flow is solved over the whole domain using an equidistant structured grid, which is going to be called the Eulerian grid. Inside the domain, an arbitrary amount of objects may reside. An object is represented by a number of points called Lagrangian markers. A discrete volume  $\Delta V_i$  is associated with each Lagrangian marker, such that the sum of the discrete volumes forms a membrane.

These Lagrangian markers may be arbitrarily located inside the domain (i.e. they do not coincide with the Eulerian grid points), therefore, we need to define some interpolating/spreading operators to communicate velocities/forces between the Eulerian and Lagrangian frames, for which a regularized delta function is used. The boundary conditions between solid objects and fluid are defined as no-slip conditions. These are incorporated by computing a forcing on the fluid to ensure the no-slip condition is satisfied. For this, first, an estimation of the velocity is computed in the Eulerian frame and interpolated to the Lagrangian markers using the discrete delta function. Then, the forcing term is computed directly at the Lagrangian markers. Finally, the forcing term is spread to the Eulerian grid and included in the right hand side of the momentum equation. Fluid flow is spatially discretized using a uniform Cartesian grid. A staggered grid is used to prevent checkerboard-like oscillations. Second-order central finite differences are used. For temporal discretization, a two-step Adams-Bashforth method is used for the convective terms and a Crank-Nicolson method is used for the diffusive terms. Because the equation for continuity has no temporal information, we need to rely on a fractional step method in which the velocity is first advanced without considering conservation of mass. In the second step, a pseudo-pressure is computed which will be used to correct the velocity and update the pressure at the end of every step ensuring that continuity is fulfilled. Afterwards, the equations corresponding to the motion of the objects may be solved.

# Chapter 3

## Methodology

### 3.1 Discretization

In this chapter, the different discretizations in 2D and 3D are presented. The mesh in 2D is a curve, which is discretized into vertices that are connected by edges. The mesh can be open or closed. The vertices are also the mass nodes. The Lagrangian markers are the midpoints of the edges. The mesh is sketched in fig. 3.1. Every edge has a associated volume

$$\Delta V_l = h_l^2 L_l, \quad (3.1)$$

where  $h_l$  is the local average Eulerian mesh spacing and  $L_l$  is the edge length.

In 3D, the mesh is a surface discretized using triangles. The mesh can be open or closed. The vertices are also the mass nodes. The Lagrangian markers are the centroids of the triangles. The mesh is sketched in fig. 3.2. Every triangle has a associated volume

$$\Delta V_l = h_l A_l, \quad (3.2)$$

where  $h_l$  is the local average Eulerian mesh spacing and  $A_l$  is the triangle area, which can be computed as  $A_l = |\mathbf{r}_{ij} \times \mathbf{r}_{ik}|/2$ , where  $\mathbf{r}$  is the position vector,  $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$  and  $i, j$  and  $k$  are the vertex indices of face  $l$ .

### 3.2 Forces

#### 3.2.1 Linear springs

The forces acting on the mass springs due to linear springs are computed according to formulas 2.24 in a slightly reformulated way for computational efficiency

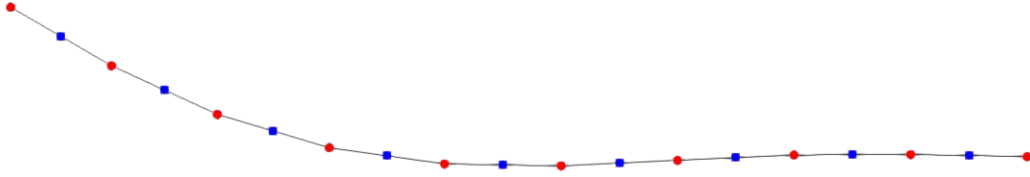


Figure 3.1: A 2D mesh consisting of nodes (red) and edges (black) and the corresponding Lagrangian markers (blue).

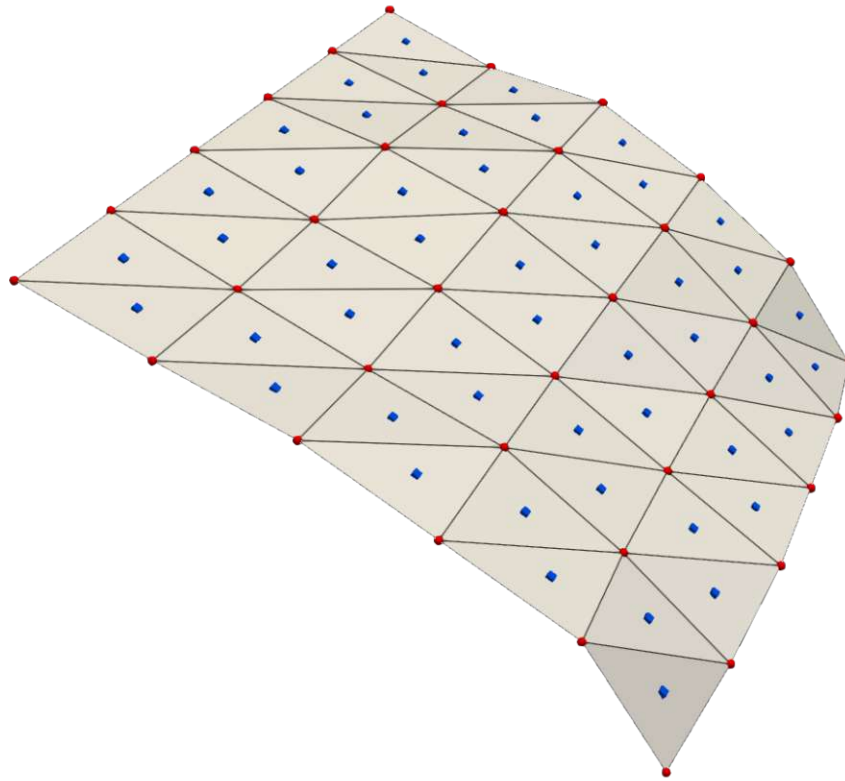


Figure 3.2: A 3D mesh consisting of nodes (red), edges (black) and faces (grey) and the corresponding Lagrangian markers (blue).

$$\mathbf{F}_i^e = -k_e (L - L_0) \frac{\mathbf{r}_{ij}}{L}, \quad (3.3a)$$

$$\mathbf{F}_j^e = -\mathbf{F}_i^e, \quad (3.3b)$$

where  $\mathbf{r}_i$  is the position vector of node  $i$ ,  $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ ,  $L = |\mathbf{r}_{ij}|$  and  $L_0$  is the relaxed spring length. To compute the spring coefficient  $k_e$ , Hooke's law (eq. 2.3) can be used in 2D to write

$$k_e = \frac{Eh}{L_0}, \quad (3.4)$$

where  $h$  is the membrane thickness. In 3D, eq. 2.35 is used.

### 3.2.2 Bending springs in 2D

Bending springs in this work are modelled using two formulations based on the corresponding bending energy terms

$$W^{b,F} = k_b [1 - \cos(\theta - \theta_0)] \quad (3.5)$$

and

$$W^{b,L} = \frac{k_b}{2} (\theta - \theta_0)^2, \quad (3.6)$$

where  $\theta$  and  $\theta_0$  are the bending angle and the bending angle in a stress-free configuration, respectively.

It can be seen that  $W^{b,L}$  is a linearized form of  $W^{b,F}$  by computing the Taylor series using the relative angle  $\tilde{\theta} = \theta - \theta_0$ :

$$\begin{aligned} k_b \cos(\tilde{\theta}) \approx & k_b \cos(\tilde{\theta}_0) - k_b \sin(\tilde{\theta}_0) (\tilde{\theta} - \tilde{\theta}_0) - \frac{k_b \cos(\tilde{\theta}_0)}{2} (\tilde{\theta} - \tilde{\theta}_0)^2 \\ & + \frac{k_b \sin(\tilde{\theta}_0)}{6} (\tilde{\theta} - \tilde{\theta}_0)^3 + \frac{k_b \cos(\tilde{\theta}_0)}{24} (\tilde{\theta} - \tilde{\theta}_0)^4 \dots \end{aligned} \quad (3.7)$$

Upon setting  $\tilde{\theta}_0 = 0$  and cutting off after the third order term, one gets

$$k_b \cos(\theta - \theta_0) \approx k_b - \frac{k_b}{2} (\theta - \theta_0)^2, \quad (3.8)$$

which can be reordered as

$$k_b [1 - \cos(\theta - \theta_0)] \approx \frac{k_b}{2} (\theta - \theta_0)^2. \quad (3.9)$$

The formulas of the nodal forces of the two energy terms follow the same pattern, which can be seen by using eq. 2.25 and the chain rule to write

$$-\frac{\partial W^{b,F}}{\partial \mathbf{r}_i} = -k_b \sin(\theta - \theta_0) \frac{\partial \theta}{\partial \mathbf{r}_i} = -k_b \Gamma_F \frac{\partial \theta}{\partial \mathbf{r}_i} \quad (3.10)$$

and

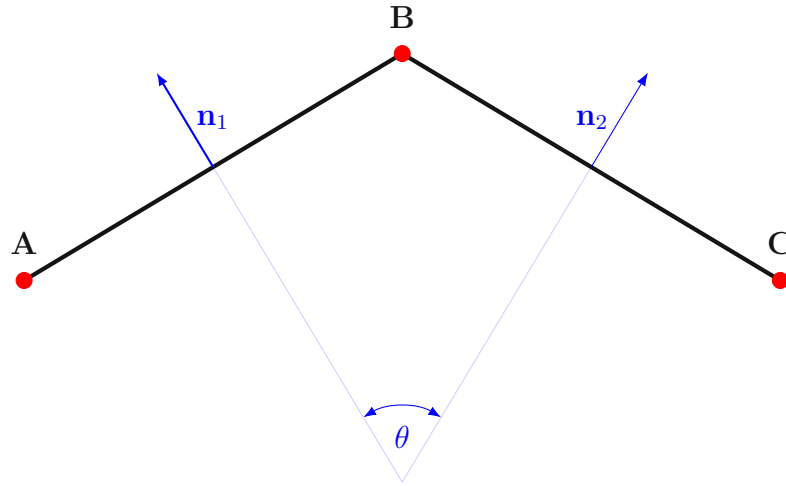


Figure 3.3: The setup for the computation of forces due to bending energy includes two edges connecting vertices  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$ , the two unit normal vectors  $\mathbf{n}_1$  and  $\mathbf{n}_2$  and the bending angle  $\theta$ .

$$-\frac{\partial W^{b,L}}{\partial \mathbf{r}_i} = -k_b (\theta - \theta_0) \frac{\partial \theta}{\mathbf{r}_i} = -k_b \Gamma_L \frac{\partial \theta}{\mathbf{r}_i}. \quad (3.11)$$

It can be seen that the computation of forces only differs in the coefficients  $\Gamma_F$  and  $\Gamma_L$ . The setup for the nodal forces due to bending energy is shown in fig. 3.3.

A detailed derivation is provided in A.1. The results can be summarized as

$$\mathbf{F}_A^b = k_b \Gamma \frac{\mathbf{n}_1}{N_1}, \quad (3.12a)$$

$$\mathbf{F}_C^b = k_b \Gamma \frac{\mathbf{n}_2}{N_2}, \quad (3.12b)$$

$$\mathbf{F}_B^b = -\mathbf{F}_A^b - \mathbf{F}_C^b. \quad (3.12c)$$

where  $\Gamma$  can either be  $\Gamma_F$  or  $\Gamma_L$  and  $\mathbf{n}_1$  and  $\mathbf{n}_2$  are the unit normal vectors defined as

$$N_1 = (r_A^{(2)} - r_B^{(2)})^2 + (r_B^{(1)} - r_A^{(1)})^2, \quad (3.13a)$$

$$N_2 = (r_B^{(2)} - r_C^{(2)})^2 + (r_C^{(1)} - r_B^{(1)})^2, \quad (3.13b)$$

$$\mathbf{n}_1 = \frac{1}{\sqrt{N_1}} \begin{pmatrix} r_A^{(2)} - r_B^{(2)} \\ r_B^{(1)} - r_A^{(1)} \end{pmatrix}, \quad (3.13c)$$

$$\mathbf{n}_2 = \frac{1}{\sqrt{N_2}} \begin{pmatrix} r_B^{(2)} - r_C^{(2)} \\ r_C^{(1)} - r_B^{(1)} \end{pmatrix}, \quad (3.13d)$$

where the superscript denotes the vector component. A different approach to compute the nodal forces presented by Truong *et al.* [4] is to use a torque acting on a node  $i$ :

$$T_i = -k_b(\theta - \theta_0). \quad (3.14)$$

It can be shown that the bending energy with respect to the torque is the same as in eq. 3.6 and a slightly modified version corresponds to the same bending energy as in eq. 3.5:

$$\int_0^{\tilde{\theta}} k_b \tilde{\theta} d\tilde{\theta} = \frac{k_b}{2} (\theta - \theta_0)^2 = W^{b,L} \quad (3.15)$$

and

$$\int_0^{\tilde{\theta}} k_b \sin(\tilde{\theta}) d\tilde{\theta} = k_b [1 - \cos(\theta - \theta_0)] = W^{b,F}. \quad (3.16)$$

In 3D, the nodal forces according to Truong *et al.* [4] can be written as

$$\mathbf{F}_A^b = k_b \Gamma \frac{(\mathbf{r}_{AB} \times \mathbf{r}_{BC}) \times \mathbf{r}_{AB}}{|\mathbf{r}_{AB}|^2 |\mathbf{r}_{BC}|}, \quad (3.17a)$$

$$\mathbf{F}_C^b = k_b \Gamma \frac{(\mathbf{r}_{AB} \times \mathbf{r}_{BC}) \times \mathbf{r}_{BC}}{|\mathbf{r}_{AB}| |\mathbf{r}_{BC}|^2}, \quad (3.17b)$$

$$\mathbf{F}_B^b = -\mathbf{F}_A^b - \mathbf{F}_C^b, \quad (3.17c)$$

where  $\Gamma$  can either be  $\Gamma_F$  or  $\Gamma_L$ , depending on the used model. In the two-dimensional case the first cross product always evaluates to

$$\frac{\mathbf{r}_{AB} \times \mathbf{r}_{BC}}{|\mathbf{r}_{AB}| |\mathbf{r}_{BC}|} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}. \quad (3.18)$$

Then the equations can be rewritten in the same form as 3.12. The bending energy requires the computation of a signed angle, which can be computed as

$$\theta = \arctan2(\sin(\theta), \cos(\theta)) = \arctan2(\|\mathbf{r}_{BA} \times \mathbf{r}_{CB}\|, \mathbf{r}_{BA} \cdot \mathbf{r}_{CB}) \quad (3.19)$$

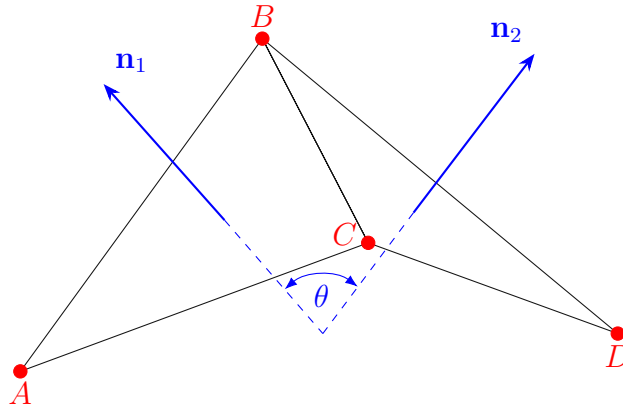


Figure 3.4: The setup for the computation of forces due to bending energy includes two adjacent triangles connecting vertices  $\mathbf{A}$ ,  $\mathbf{B}$ ,  $\mathbf{C}$  and  $\mathbf{D}$ , the two unit normal vectors  $\mathbf{n}_1$  and  $\mathbf{n}_2$  and the bending angle  $\theta$ .

### 3.2.3 Bending springs in 3D

Bending springs in 3D are modelled using the same energy terms as already used in 2D,  $W^{b,F}$  (eq. 3.5) and  $W^{b,L}$  (eq. 3.6). A sketch of the setup for the computation of the nodal forces is shown in 3.4.

The equations for the nodal forces can be summarized as follows:

$$\mathbf{F}_B^b = k_b \Gamma \left( \frac{\mathbf{N}_1}{|\mathbf{N}_1|^2} \frac{\mathbf{r}_{CB} \cdot \mathbf{r}_{AC}}{|\mathbf{r}_{CB}|} + \frac{\mathbf{N}_2}{|\mathbf{N}_2|^2} \frac{\mathbf{r}_{CB} \cdot \mathbf{r}_{DC}}{|\mathbf{r}_{CB}|} \right), \quad (3.20a)$$

$$\mathbf{F}_C^b = k_b \Gamma \left( \frac{\mathbf{N}_1}{|\mathbf{N}_1|^2} \frac{\mathbf{r}_{CB} \cdot \mathbf{r}_{BD}}{|\mathbf{r}_{CB}|} + \frac{\mathbf{N}_2}{|\mathbf{N}_2|^2} \frac{\mathbf{r}_{CB} \cdot \mathbf{r}_{BA}}{|\mathbf{r}_{CB}|} \right), \quad (3.20b)$$

$$\mathbf{F}_A^b = k_b \Gamma |\mathbf{r}_{CB}| \frac{\mathbf{N}_1}{|\mathbf{N}_1|^2}, \quad (3.20c)$$

$$\mathbf{F}_D^b = k_b \Gamma |\mathbf{r}_{CB}| \frac{\mathbf{N}_2}{|\mathbf{N}_2|^2}, \quad (3.20d)$$

$$(3.20e)$$

where  $\Gamma$  can either be  $\Gamma_F$  or  $\Gamma_L$ , depending on the used model and  $\mathbf{N}_1$  and  $\mathbf{N}_2$  are the normal vectors, which can be defined as

$$\mathbf{N}_1 = \mathbf{r}_{CB} \times \mathbf{r}_{AB}, \quad (3.21a)$$

$$\mathbf{N}_2 = \mathbf{r}_{CB} \times \mathbf{r}_{BD}. \quad (3.21b)$$

Detailed derivations of the forces can be found in Cimrak and Jancigova [51]. Note that the approach to derive the forces by Cimrak and Jancigova [51] is based on the assumption that a pair of triangles has to be force and torque free in the absence of external forces, which

was proven in their work to be mathematically equivalent to expressions derived by Fedosov [34] or Tullio and Pascazio [29], but are computationally cheaper. The bending angle  $\theta$  can be computed as

$$\theta = \arctan2((\mathbf{N}_1 \times \mathbf{N}_2) \cdot \mathbf{r}_{CB}, \mathbf{N}_1 \cdot \mathbf{N}_2). \quad (3.22)$$

### 3.3 Force transfer

Within the IBM context, the forces of the fluid acting on the SNM are computed at the Lagrangian markers, so a transfer of the forces from the Lagrangian markers to the mass nodes is required. In 2D, a Lagrangian marker is always at the midpoint of an edge, so half the force can be added to each mass node. In 3D, a Lagrangian marker is always at the centroid of a triangle, so forces are equally distributed to all three nodes of a triangle. The force on node  $i$  can then be computed as [30][3]

$$\mathbf{F}_i^{\text{ext}} = \frac{1}{3} \sum_{j=1}^{n_{fi}} \mathbf{F}_{l,f}, \quad (3.23)$$

where  $n_{fi}$  is the number of faces adjacent to vertex  $i$  and  $\mathbf{F}_{l,f}$  the force at the Lagrangian marker of faces  $f$  adjacent to vertex  $i$ . Note that the forces computed by the IBM are per unit mass, whereby the SNM requires forces in  $N$ . This is done using the volumes associated with the Lagrangian markers defined in eq. 3.1 and eq. 3.2:

$$\mathbf{F}_{l,f} = \rho_f \mathbf{F}_l \Delta V_l, \quad (3.24)$$

where  $\rho_f$  is the fluid density and  $\mathbf{F}_l$  is the force from the IBM.

### 3.4 Integration

For numerical integration of eq. 2.32, the ordinary differential equation of second-order has to be rewritten into a system of first-order ordinary differential equations. Eq. 2.32 can be written using dot notation as

$$m_i \ddot{\mathbf{r}}_i + \gamma \dot{\mathbf{r}}_i = \mathbf{F}_i. \quad (3.25)$$

The corresponding linear system of first-order ordinary differential equations is

$$\begin{pmatrix} \dot{\mathbf{r}}_i \\ \dot{\mathbf{v}}_i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & -\frac{\gamma_i}{m_i} \end{pmatrix} \begin{pmatrix} \mathbf{r}_i \\ \mathbf{v}_i \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{\mathbf{F}_i}{m_i} \end{pmatrix}. \quad (3.26)$$

Since the fluid solver uses a two-stage Adam-Bashforth method and due to the small time steps required in practice, a semi-implicit Euler method is used for numerical integration. Within a single time step, the corresponding external and internal forces need to be computed and the pinned boundary condition has to be applied by setting forces acting on a mass point to zero. The general integration algorithm is summarized in algorithm 1.

---

**Algorithm 1** Force integration
 

---

```

function INTEGRATE( $\alpha, \mathbf{F}_l$ )  $\triangleright \mathbf{F}_l$  are forces at Lagrangian markers
  for all M in Membranes do
     $\mathbf{F}_{\text{int}} \leftarrow \text{M.COMPUTE\_INTERNAL\_FORCES}()$ 
    for all Nodes  $i$  in M do
       $\mathbf{F}_i \leftarrow m\mathbf{g} + \mathbf{F}_{\text{int},i}$ 
    end for
    if 2D then
      for all Linear Edges in M do
        for all Nodes  $i$  in a Linear Edge do
           $\mathbf{F}_i \leftarrow \mathbf{F}_i + \frac{1}{2}\mathbf{F}_l$   $\triangleright l$  is Lagrangian marker of respective Edge
        end for
      end for
    else if 3D then
      for all Faces in M do
        for all Nodes  $i$  in a Face do
           $\mathbf{F}_i \leftarrow \mathbf{F}_i + \frac{1}{3}\mathbf{F}_l$   $\triangleright l$  is Lagrangian marker of respective Face
        end for
      end for
    end if
    for all Nodes  $i$  in Boundary do
       $\mathbf{F}_i = 0$ 
    end for
    for all Nodes  $i$  in M do
       $\mathbf{v}_i \leftarrow \mathbf{v}_i + \alpha \frac{\mathbf{F}_i}{m}$ 
       $\mathbf{r}_i \leftarrow \mathbf{r}_i + \alpha \mathbf{v}_i$ 
    end for
  end for
end function

```

---

## 3.5 Implementation

The SNM is implemented in C++14 [52]. A code structure based on templates and inheritance is used where suitable to reduce the amount of code. An overview of the code is given

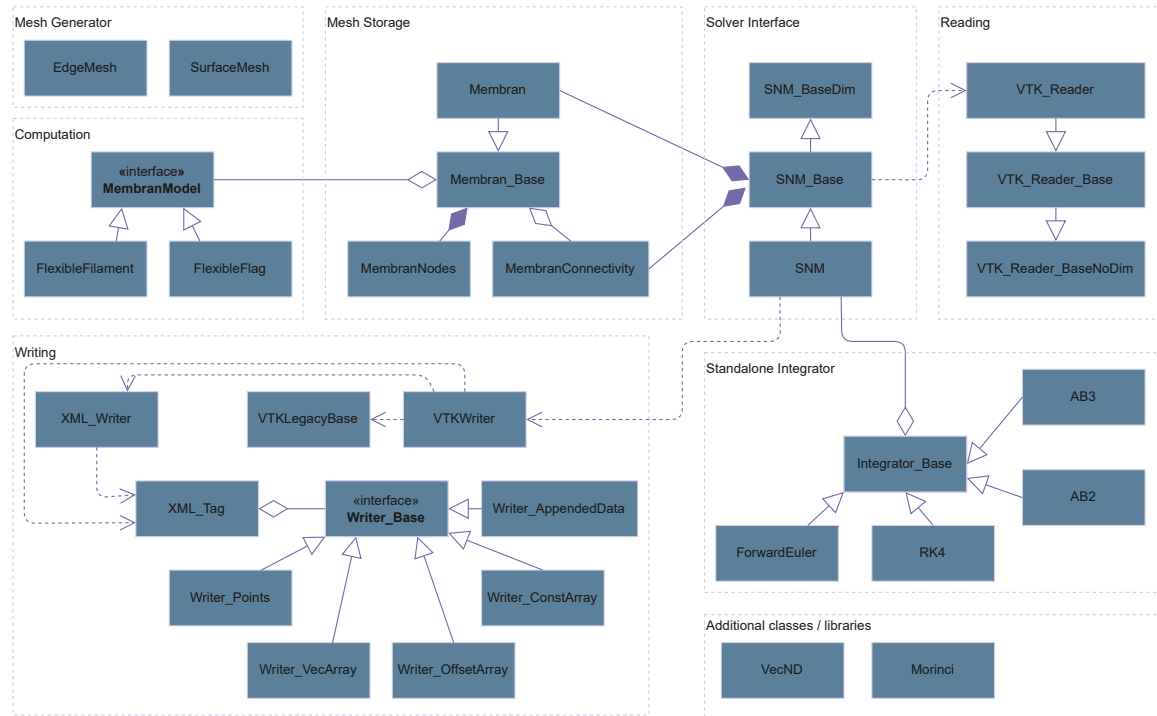
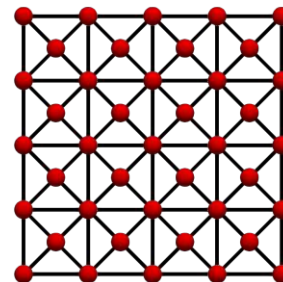
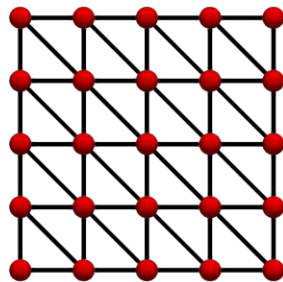
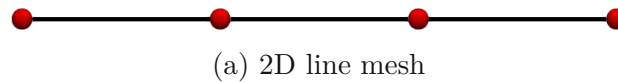


Figure 3.5: An overview of the code structure of the SNM solver.

in fig. 3.5. The code is divided into 7 modules: *Mesh Generator*, *Mesh Storage*, *Reading*, *Writing*, *Solver Interface*, *Computation* and *Additional classes/libraries*.

The two classes *EdgeMesh* and *SurfaceMesh* in the *Mesh Generator* module are used to generate 2D and 3D meshes, respectively. The generated 2D mesh is a straight line (fig. 3.6a). In 3D, two different types of triangle meshes discretizing a rectangular membrane can be generated. In both cases, a rectangle gets divided into smaller  $N_x \times N_y$  rectangles. These rectangles are then split into either two (fig. 3.6b) or four (fig. 3.6c) triangles.

The SNM solver itself is implemented in the *Solver Interface* module, which uses the *Reading* and *Writing* modules to read and write mesh files and offers the interface for the coupling



(b) 3D mesh with two triangles per rectangle (c) 3D mesh with four triangles per rectangle

Figure 3.6: An overview of the mesh types supported by the mesh generator.

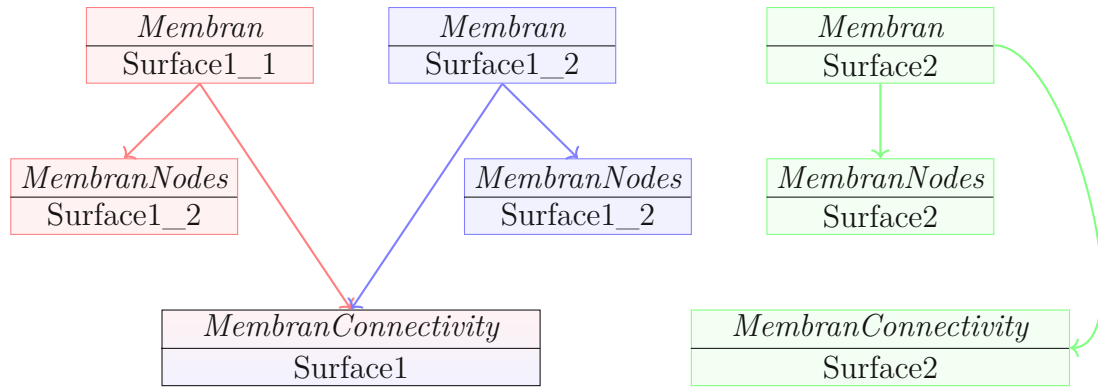


Figure 3.7: The IDs of the *Membran* instances are used to control memory storage.

with fluid solvers.

Mesh files are written using the legacy *vtk* file format or the XML (*vtu*) file format [53]. The data can be written in ASCII or binary. XML offers the advantage of allowing a keyword to specify the byte order, so no swapping of bytes is required. Additionally, a second file is required for the definition of the boundary patches. The boundary file is structured such that the name (ID) of a boundary patch is followed by a single number corresponding to the boundary vertex in 2D and two numbers corresponding to the first and last vertex of a boundary patch in 3D in the line below.

Within the solver, meshes are stored in the *Mesh Storage* module, where node-specific data (e.g. coordinates or velocities of vertices) is stored in the class *MembranNodes*. Data about the connectivity is stored in the class *MembranConnectivity*. All *MembranConnectivity* and *Membran* instances are stored within *SNM*, whereby a *Membran* object contains an instance of *MembranNodes* and a pointer to the respective *MembranConnectivity* instance. Every *Membran* object is associated with an ID, that controls what is stored. If IDs of different *Membran* instances contain the same string, which is followed by an underscore and a number, the connectivity is only stored once for memory efficiency (fig. 3.7).

The 2D mesh is sketched in 3.8 and the corresponding data structure is shown in fig. 3.9. For fast retrieval of the required data for the computation of both linear springs and bending springs, two types of edges are used: linear and angular edges, respectively. Linear edges store the indices of their two vertices and their vertex indices are sorted in a way such that the first vertex of an linear edge is the second vertex of the previous linear edge. Angular edges store the indices of two neighbouring linear edges. Although theoretically storing vertex indices instead of linear edge indices would suffice for the forces considered within the scope of this work, linear edge indices are stored to provide more flexibility. A boundary angular edge will store a  $-1$  instead of an linear edge index and boundary angular edges are always stored after the internal angular edges. The first linear edge in an angular edge is always the linear edge where the first vertex corresponds to the vertex in the center. The second linear edge is always the edge where the second vertex corresponds to the center. Boundary information is stored within a hashmap, which stores the boundary angular ID as

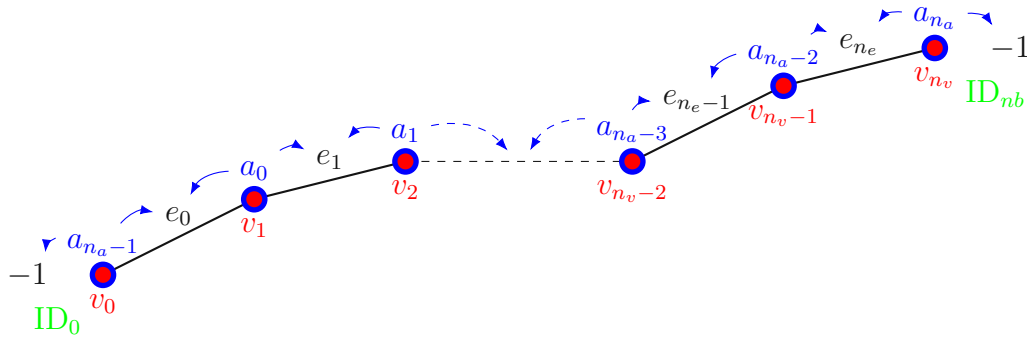


Figure 3.8: The 2D mesh data structure includes vertices (red), linear edges (black), angular edges (blue) and boundary patches (green).

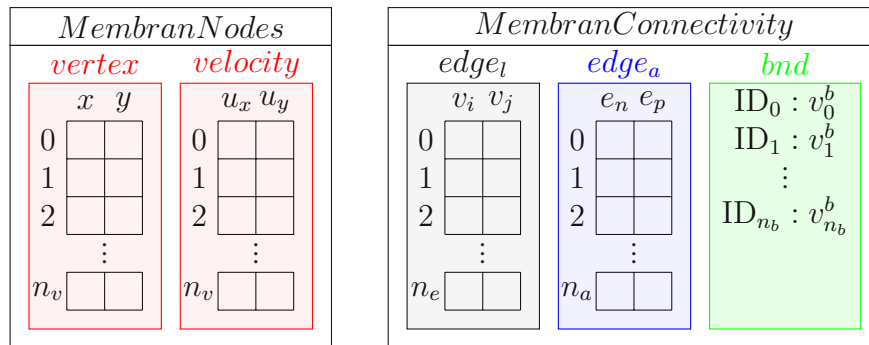


Figure 3.9: The class *MembranNodes* stores the vertices and nodal velocities. The class *MembranConnectivity* stores the linear edges, angular edges and boundary information.

key and the corresponding vertex index as value.

The 3D mesh is sketched in 3.10 and the corresponding data structure is shown in fig. 3.11. For simple use and fast retrieval of the necessary data required for the computation of linear and bending springs, an edge-based data structure is used. Edges store the indices of the two vertices and additionally the indices of the two adjacent faces. Boundary edges store a  $-1$  instead of one of the face indices. Edges are ordered so that internal edges appear first, followed by boundary edges grouped by boundary patch. Boundary information is stored within a hashmap, which stores the boundary patch ID as key and the corresponding index of the first edge and the number of edges as value. Additionally, faces are used, which store their three vertex indices.

In the *Computation* module, the computation of the nodal forces is defined. *MembranModel* is designed as an interface using an abstract class, so new classes defining different equations can easily be added. Furthermore, boundary conditions and damping can be changed using this class.

The *Standalone Integrator* module is used to define numerical integration methods like the Forward Euler method or the fourth-order Runge-Kutta method used for the standalone version.

The solver uses the *Morinci* library, which is part of *TXPS*, for some helper functions for file input and output.

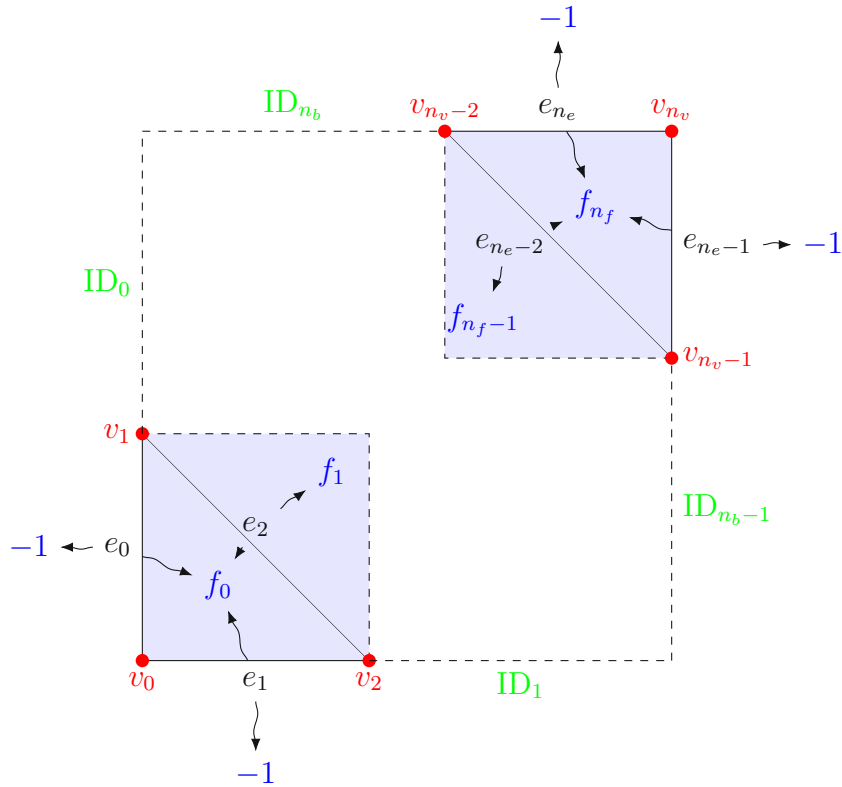


Figure 3.10: The 3D mesh data structure includes vertices (red), edges (black), faces (blue) and boundary patches (green).

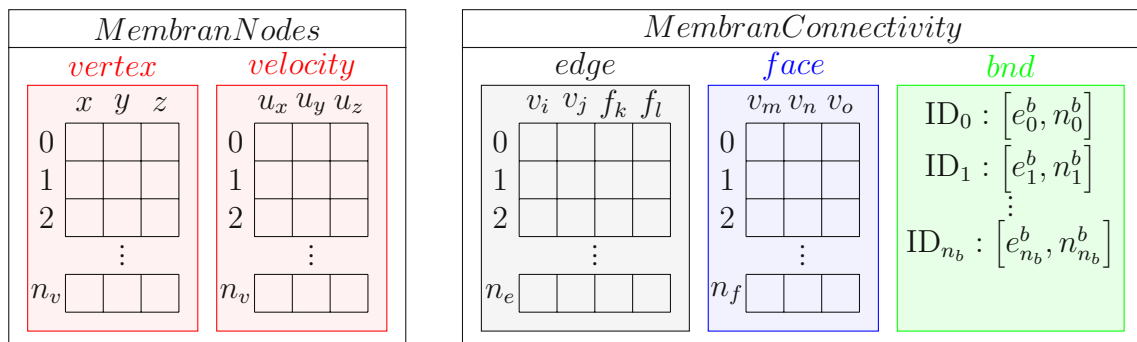


Figure 3.11: The class *MembranNodes* stores the vertices and nodal velocities. The class *MembranConnectivity* stores the edges, faces and boundary information.

## Results and Discussion

### 4.1 Cantilever beam

First, the SNM is tested as a standalone version and is validated against the cantilever beam validation case in vacuum and is compared against values from Truong *et al.* [4] for the damped case and Turek and Hron [54] and Engels [47] for the undamped case. The clamped boundary condition at the left-hand side is implemented by setting the force at the leftmost and second leftmost node to zero, fixing both nodes in space. The bending stiffness was derived by Truong *et al.* [4] using Euler-Bernoulli beam theory as

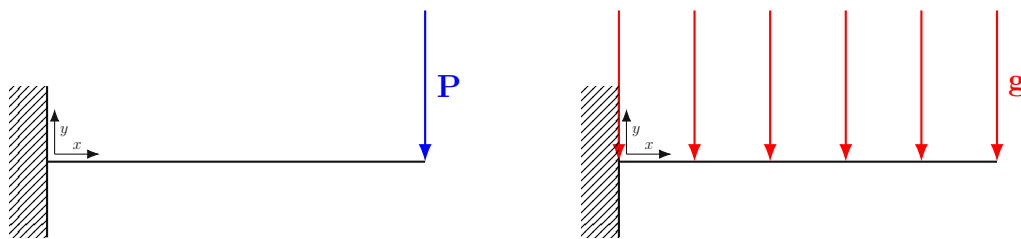
$$k_b = \frac{EI}{L_{cb}} \frac{n(2n+1)}{2(n+1)}, \quad (4.1)$$

where  $E$  is the Young modulus,  $I$  is the second moment of area,  $L_{cb}$  is the beam length and the number of nodes is  $n + 2$ .

#### 4.1.1 Damped case

For the damped case, the damping factor in 3.25 is set to

$$\gamma = 2\sqrt{k_b m_i}, \quad (4.2)$$



(a) Damped cantilever beam case.

(b) Undamped cantilever beam case.

Figure 4.1: Sketches of the different cantilever beam validation cases.

Parameter	Value	Unit
$E$	28800	$\frac{N}{m^2}$
height	0.1	m
width	1	m
length	0.3	m
mass	23.55	kg
number of nodes	{4, 8, 16, 32, 64, 128}	1
$\mathbf{g}$	(0, 0)	$\frac{m}{s^2}$
$\mathbf{P}$	{(0, -0.39), (0, -1.96), (0, -3.92), (0, -7.84), (0, -11.76)}	N
time step	0.0001	s
simulation time	50	s

Table 4.1: The simulation and material parameters of the cantilever beam validation case with damping.

which results in a reasonably fast convergence to a steady state solution. A force  $\mathbf{P}$  of various magnitudes is applied at the tip, as it is sketched in fig. 4.1a. The cross-section is assumed to be a rectangle. For numerical integration, a fourth-order Runge-Kutta method is used with a step size of 0.0001. Integration is done for 500000 time steps. An overview of material and simulation parameters is given in table 4.1.

Fig. 4.2 shows the position of the tip in the  $x$ -direction over the applied load and fig. 4.3 the corresponding relative error with respect to the reference values. Small deviations for  $n = 4$  can be seen, but with increasing resolution, the results converge to the reference values and therefore the relative error gets close to zero. Furthermore, it can be seen that the error increases slightly with increasing load.

Fig. 4.4 shows the position of the tip in the  $y$ -direction over the applied load and fig. 4.5 the corresponding relative error with respect to the reference values. It can be seen that the deviation and therefore also the relative error increases with the applied load, which is to be expected, since eq. 4.1 assumes only small deformations. With increasing mesh resolution, the solution converges to the reference values. Furthermore, it can be seen that for low resolutions the nonlinear bending energy results in less error, although this difference gets negligible for higher mesh resolutions.

### 4.1.2 Undamped case

The case without damping is compared to the results obtained by Turek and Hron [54] and Engels [47]. The solution by Turek and Hron [54] is based on a finite element formulation, whereby Engels [47] numerically integrates nonlinear beam equations. No force is applied at the tip, but gravity is set to a nonzero value, as it is sketched in fig. 4.1b. The beam has a rectangular cross-section. The material and simulation parameters are summarized in table 4.2.

Table 4.3 shows the results for the deflection of the tip in the  $x$ -direction in the format  $mean \pm amplitude [frequency]$ . The relative error in comparison with the finite element

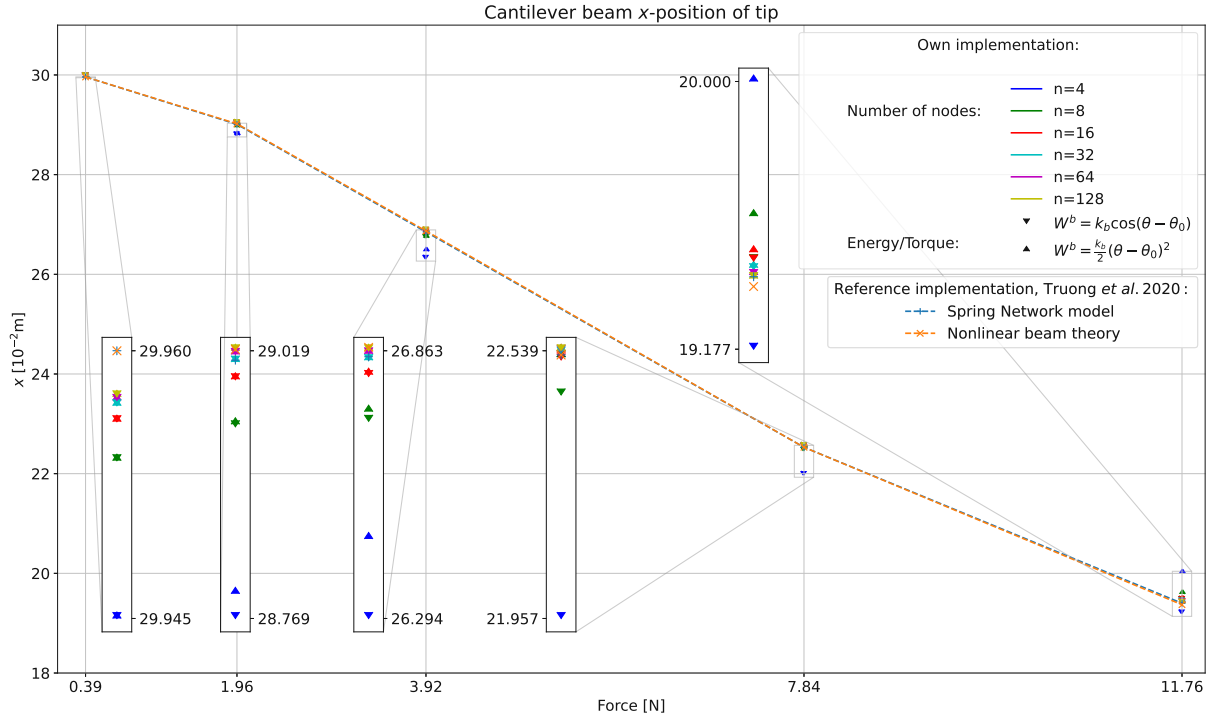


Figure 4.2: The x-position of the tip of the cantilever beam in comparison with the results obtained from Truong *et al.* [4].

formulation is around 1 %, whereas the relative error in comparison with the nonlinear beam model is significantly higher. This can be explained by the assumption of no elongation of the nonlinear beam model.

Table 4.4 shows the results for the deflection of the tip in the  $y$ -direction in the format  $mean \pm amplitude [frequency]$ . In comparison with both references, a relative error of around one to two percent can be seen.

In all cases, no significant change in error for the different timesteps can be seen. It has to be noted that simulations with a larger timestep diverged. This means that the timestep required for convergence is already so small, that no improvement in accuracy can be expected in this case. Furthermore, no significant difference between the nonlinear and linear energy terms can be seen.

## 4.2 Flexible filament

For testing of the 2D coupled solver, a flexible filament in a free stream configuration is simulated. The filament is fixed at the leading edge and is initially oriented, such that there is an angle  $\alpha_0 = 0.1\pi$  between the filament and the flow direction. The leading edge is located at the origin and the filament has a length  $L$  and thickness  $h = 0.01L$ . Gravity is acting alongside the flow direction. The computational domain size with respect to the filament length is  $[-2L, 6L] \times [-4L, 4L]$  in  $x$  and  $y$  direction, respectively. The ratio of solid to fluid density is defined as  $\gamma = \rho_s/\rho_f = 150$ . The Reynolds number is set to  $Re = UL/\nu = 200$ ,

Parameter	Value	Unit
$\nu$	0.4	1
$\tilde{E}$	$1.6 \times 10^6$	$\frac{\text{N}}{\text{m}^2}$
height	0.02	m
width	1	m
length	0.35	m
mass	7	kg
number of nodes	64	1
$\mathbf{g}$	(0, -2)	$\frac{\text{m}}{\text{s}^2}$
$\mathbf{P}$	(0, 0)	N
time step	{0.00005, 0.00001, 0.000005}	s
simulation time	10	s

Table 4.2: The simulation and material parameters of the cantilever beam validation case without damping.

Simulation	dt [s]	x-displacement [ $10^{-3}$ ]	$\epsilon_{\text{rel,ref1}}$ [%]	$\epsilon_{\text{rel,ref2}}$ [%]
$W^{b,L}$	5e-05	-14.46±14.46[1.111]	1.084±1.084[1.046]	8.673±8.266[0.716]
	1e-05	-14.428±14.428[1.112]	0.86±0.86[1.137]	8.432±8.026[0.807]
	5e-06	-14.446±14.446[1.111]	0.986±0.986[1.046]	8.568±8.161[0.716]
$W^{b,F}$	5e-05	-14.459±14.459[1.111]	1.077±1.077[1.046]	8.665±8.258[0.716]
	1e-05	-14.426±14.426[1.112]	0.846±0.846[1.137]	8.417±8.011[0.807]
	5e-06	-14.445±14.445[1.111]	0.979±0.979[1.046]	8.56±8.154[0.716]
Ref 1 [54]	0.005	-14.305 ± 14.305 [1.0995]	-	-
Ref 2 [47]	-	13.306 ± 13.306 [1.1031]	-	-

Table 4.3: Results for the x-displacement of the cantilever beam validation case without damping in comparison with results obtained from Turek and Hron [54] and Engels [47]. Results are given as *mean ± amplitude[frequency]*.

Simulation	dt [s]	y-displacement [ $10^{-3}$ ]	$\epsilon_{\text{rel,ref1}}$ [%]	$\epsilon_{\text{rel,ref2}}$ [%]
$W^{b,L}$	5e-05	-64.133±64.078[1.081]	0.827±1.661[1.683]	0.616±1.816[2.003]
	1e-05	-64.063±64.054[1.082]	0.717±1.697[1.592]	0.508±1.853[1.913]
	5e-06	-64.102±64.071[1.081]	0.778±1.671[1.683]	0.568±1.826[2.003]
$W^{b,F}$	5e-05	-64.132±64.078[1.081]	0.825±1.661[1.683]	0.615±1.816[2.003]
	1e-05	-64.061±64.052[1.082]	0.714±1.7[1.592]	0.505±1.856[1.913]
	5e-06	-64.1±64.07[1.081]	0.775±1.673[1.683]	0.565±1.828[2.003]
Ref 1 [54]	0.005	-63.607 ± 65.16 [1.0995]	-	-
Ref 2 [47]	-	-63.734 ± 65.263 [1.1031]	-	-

Table 4.4: Results for the y-displacement of the cantilever beam validation case without damping in comparison with results obtained from Turek and Hron [54] and Engels [47]. Results are given as *mean ± amplitude[frequency]*.

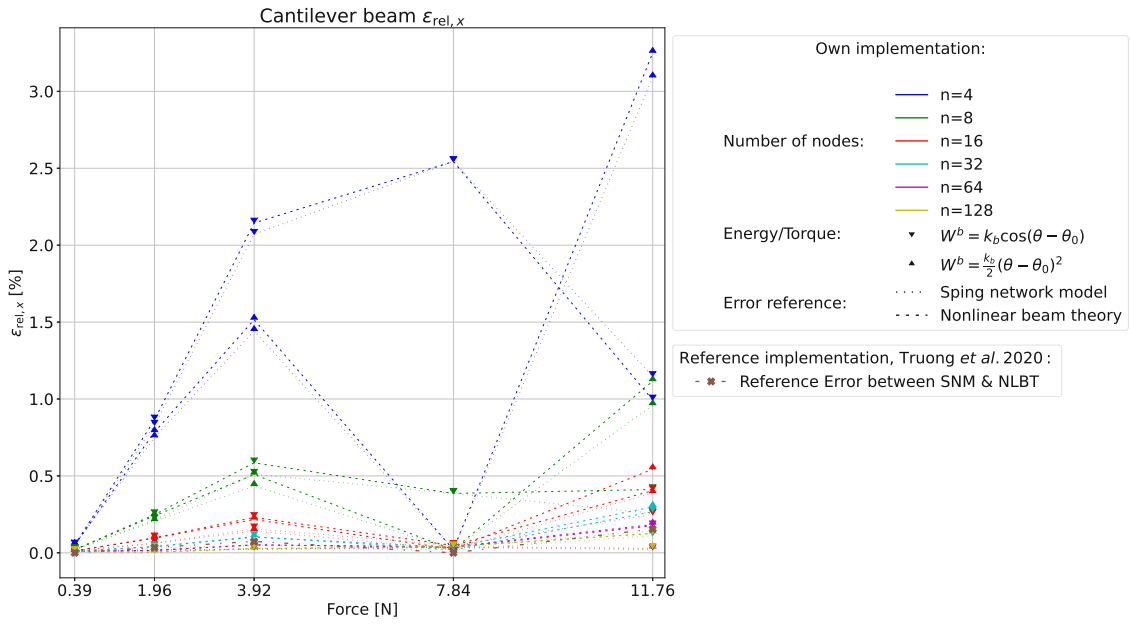


Figure 4.3: The relative error of the x-position of the tip of the cantilever beam with respect to the reference values obtained from Truong *et al.* [4].

whereby the Froude number is set to  $Fr = U/\sqrt{gL} = 1.414$ . It is assumed that the filament does not experience elongation, which is enforced by setting a large nondimensional elastic modulus  $\epsilon = Eh/(\rho_f U^2 L) = 2500$ . The bending stiffness is  $\beta = B/(\rho_f U^2 L^3) = 0.0015$  and a computational model based on  $W^{b,F}$  is used. The resulting input parameters are presented in table 4.5.

A constant velocity boundary condition is applied at the inlet and an outflow boundary condition is applied at the outlet, whereby periodic boundary conditions are used at the remaining boundaries. The setup is shown in fig. 4.6.

Fig. 4.7 shows the trailing edge location over time in comparison with Tullio and Pascazio [29]. A significant difference between both, amplitudes and frequencies can be seen. Since the SNM is validated and working as expected (see chapter 4.1), it has to be concluded that the coupling is not working correctly.

### 4.3 Flexible flag

The flow around a flexible flag is considered for testing of the 3D coupled solver. The flag is a square with an edge size  $L$ . The center of the leading edge is located at the origin and the flag is oriented such that initially there is an angle  $\alpha_0 = 0.1\pi$  between the flag and the flow direction and the flag thickness is  $h = 0.01L$ . The Reynolds number is set to  $Re = UL/\nu = 200$ . The ratio between solid and fluid density is set to  $\gamma = \rho_s/\rho_f = 100$ . It is assumed that the filament does not experience elongation, which is enforced by setting a large nondimensional elastic modulus  $\epsilon = Eh/(\rho_f U^2 L) = 2500$ . The bending stiffness is set to

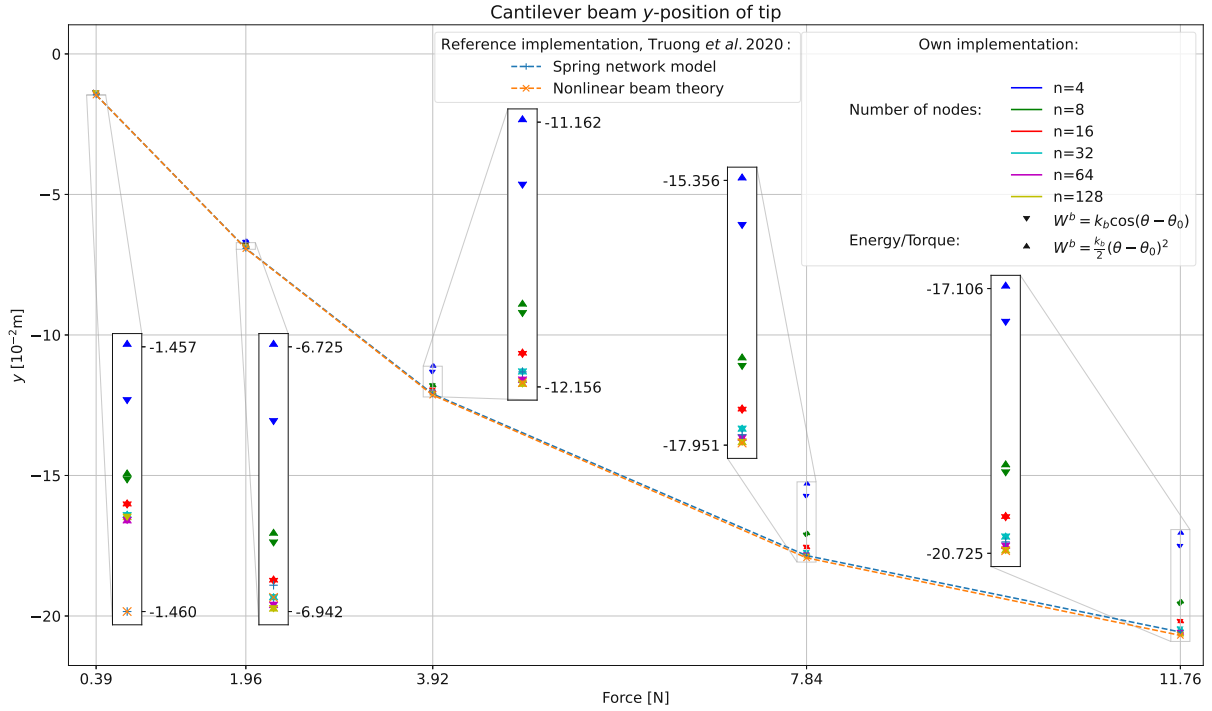


Figure 4.4: The y-position of the tip of the cantilever beam in comparison with the results obtained from Truong *et al.* [4].

three different values, namely  $\beta = B/(\rho_f U^2 L^3) = \{0.0001, 0.001, 0.01\}$  and a computational model based on  $W^{b,F}$  is used. Gravity is neglected. The resulting input parameters are presented in table 4.6.

The computational box is of dimensions  $[-L - L] \times [-4L, 4L] \times [-L, 7L]$ . A constant velocity boundary condition is applied at the inlet and an outflow boundary condition is applied at the outlet, whereby periodic boundary conditions are used at all other remaining boundaries. The setup is shown in fig. 4.6.

In chapter 4.2 it was already established that the coupling between the fluid solver and the SNM is faulty. Due to this, the correctness of the SNM in 3D can not be proven within the scope of this thesis. For qualitative validation, the flexible flag case is run with different bending moduli. Fig. 4.8 shows the cross-section at the center of the flag. The flag shows a higher stiffness for a higher bending modulus, as expected. The results at  $t = 3.0s$  also show that for  $B = 10^{-3}$  and  $B = 10^{-4}$  the flag intersects with itself, meaning that the simulation is physically meaningless after a certain point.

Fig. 4.9 and 4.10 show similar plots for the cross-section at the bottom and top of the flag, respectively. Similar behaviour as discussed for the center-line can be observed. It has to be noted that the fluid solver has stability issues with a larger number of Lagrangian markers. So the simulation was only performed for 4096 Lagrangian marker, which is significantly lower compared to Tullio and Pascazio [29].

Parameter	Value	Unit
$E$	$2.5 \times 10^5$	$\frac{\text{N}}{\text{m}^2}$
$B$	$1.5 \times 10^{-3}$	Nm
$L$	1	m
$h$	0.01	m
$\rho_f$	1	$\frac{\text{kg}}{\text{m}^3}$
$\rho_s$	150	$\frac{\text{kg}}{\text{m}^3}$
$\mathbf{U}$	(0, 1)	$\frac{\text{m}}{\text{s}}$
$\mathbf{g}$	(0, 0.500151)	$\frac{\text{m}}{\text{s}^2}$
domain height	8	m
domain width	8	m
Eulerian grid spacing	1/64	m
number of Lagrangian markers	92	1
time step	1/6400	s
simulation time	25	s

Table 4.5: The simulation and material parameters of the flexible filament case.

Parameter	Value	Unit
$E$	$2.5 \times 10^5$	$\frac{\text{N}}{\text{m}^2}$
$B$	$\{10^{-4}, 10^{-3}, 10^{-2}\}$	Nm
$L$	1	m
$h$	0.01	m
$\rho_f$	1	$\frac{\text{kg}}{\text{m}^3}$
$\rho_s$	100	$\frac{\text{kg}}{\text{m}^3}$
$\mathbf{U}$	(0, 0, 1)	$\frac{\text{m}}{\text{s}}$
$\mathbf{g}$	(0, 0, 0)	$\frac{\text{m}}{\text{s}^2}$
domain length	8	m
domain width	8	m
domain height	2	m
Eulerian grid spacing	1/64	m
number of Lagrangian markers	4096	1
time step	1/5000	s
simulation time	40	s

Table 4.6: The simulation and material parameters of the flexible flag case.

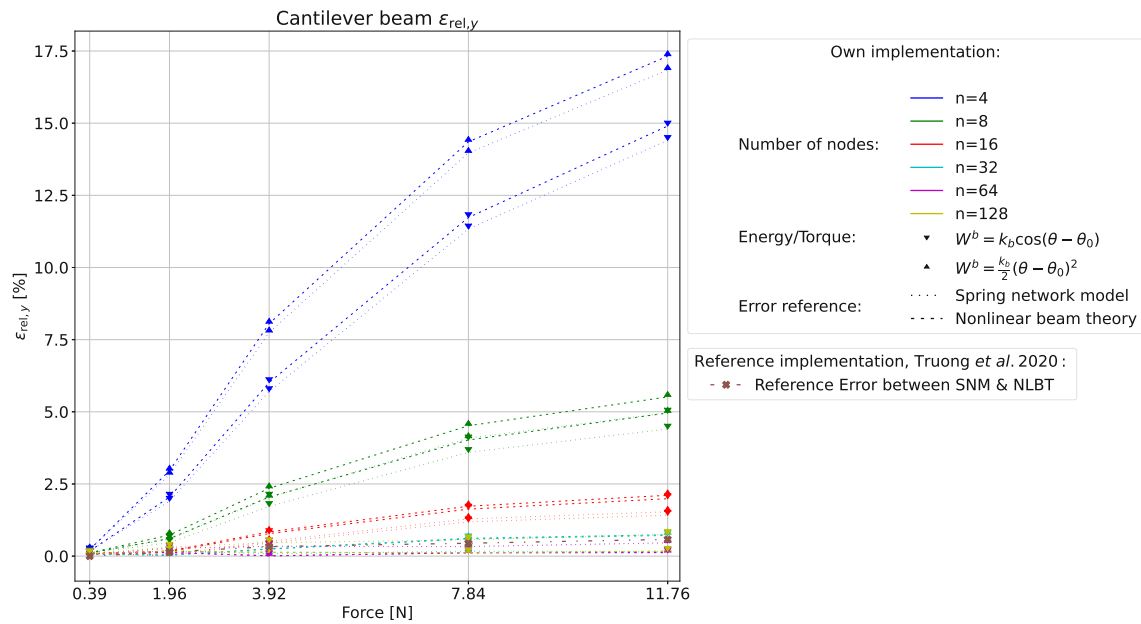


Figure 4.5: The relative error of the y-position of the tip of the cantilever beam with respect to the reference values obtained from Truong *et al.* [4].

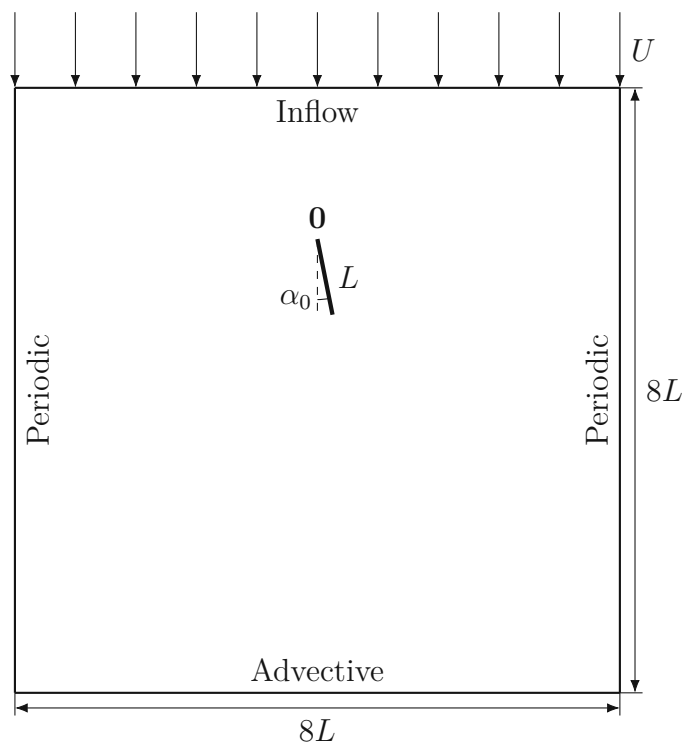


Figure 4.6: Setup of the flexible filament and flexible flag case.

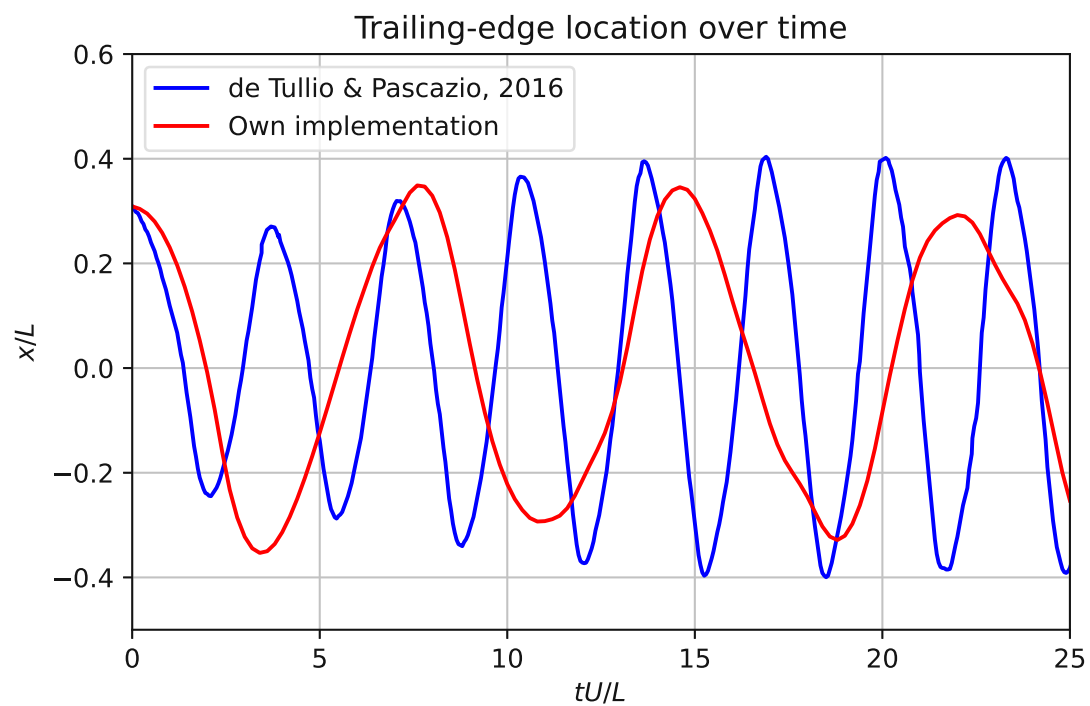


Figure 4.7: Comparison of the trailing-edge location over time with results from Tullio and Pascazio [29].

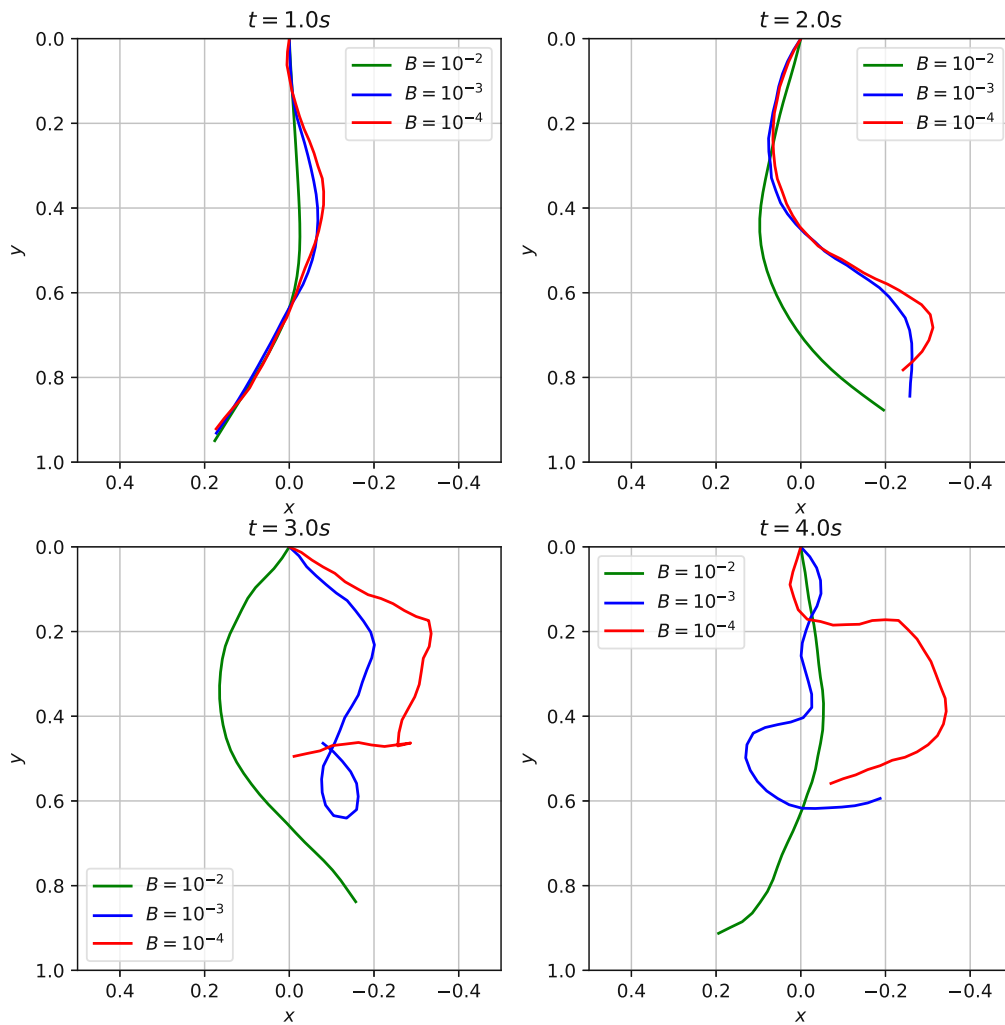


Figure 4.8: Comparison of the flag center cross-section for  $B = \{10^{-4}, 10^{-3}, 10^{-2}\}$ .

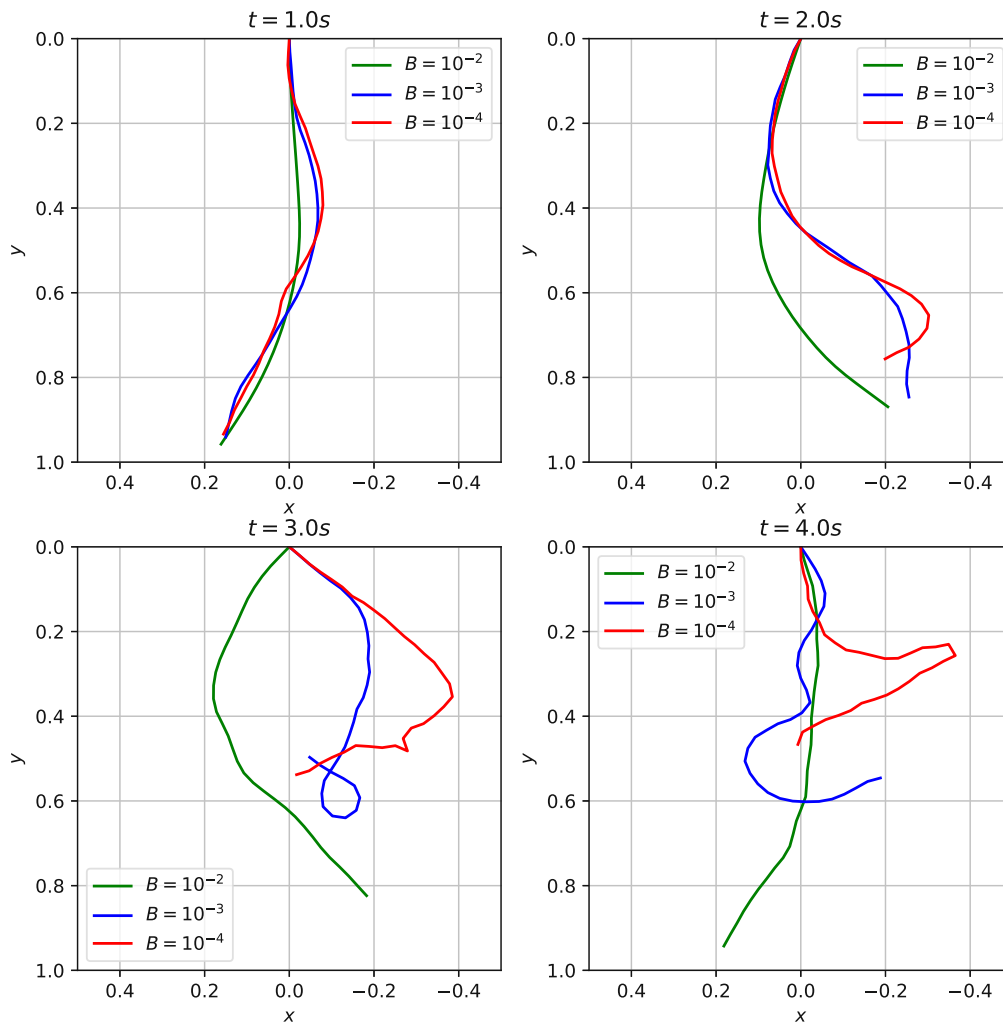


Figure 4.9: Comparison of the flag bottom cross-section for  $B = \{10^{-4}, 10^{-3}, 10^{-2}\}$ .

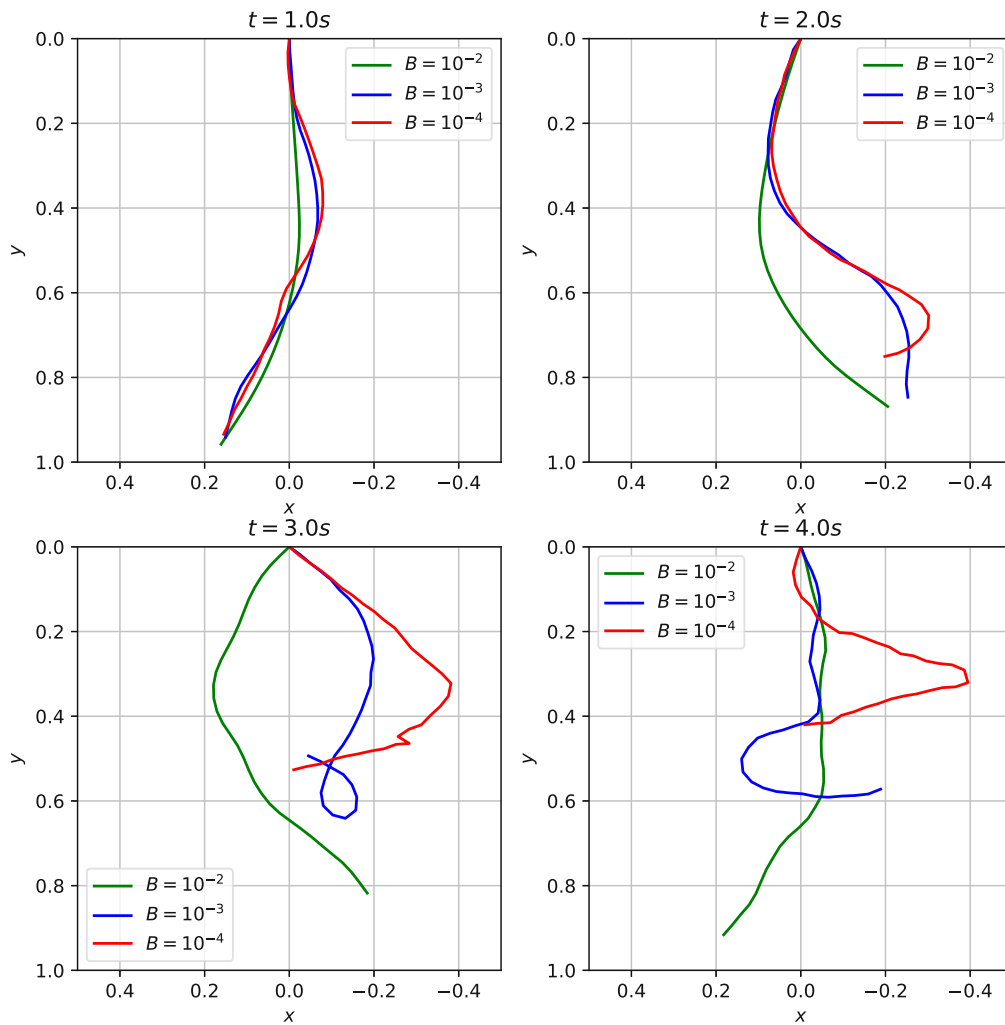


Figure 4.10: Comparison of the flag top cross-section for  $B = \{10^{-4}, 10^{-3}, 10^{-2}\}$ .

---

# Conclusion and Outlook

The SNM in 2D has been validated in chapter 4.1 and shows good agreement with literature. This proves the correctness of the implemented mesh data structure, computation of forces and the integration methods in 2D, as well as the working of the general solver infrastructure, like file input and output. In chapter 4.2, it has been shown that the current coupling procedure fails. Due to this, only a simple qualitative validation is provided for the 3D SNM in chapter 4.3, which shows a behaviour as expected, so there is reason to assume a correct implementation of the mesh data structure in 3D. Since the integration methods in 3D are the same as in 2D, there is also reason to assume the correctness of the aforementioned. Only the computation of the forces in 3D is not validated yet due to missing quantitative validation. But over the course of this work, another formulation of the force computation had been implemented based on literature and own derivations, which is mathematically equivalent to the version used in the end. Both versions resulted in the exact same results, so a wrong implementation is unlikely. Since this work is focused on the SNM and its implementation, the coupling problem is not further investigated.

The lower frequency of the trailing edge of the flexible filament compared to the frequency of the reference solution might indicate that the forces acting on the SNM are underestimated, whereby self-collision of the flag might indicate an overestimation of the forces. A possible error source lies in the way the coupling forces are computed, where possibly either the method of how the external forces acting on the SNM are computed from the IBM forces or its implementation is erroneous.

When using SNMs, commonly the forces are computed from the pressure and/or viscous stresses (see, for example, [29] or [3]) and not from the IBM forces themselves. It has to be noted that Huang, Shin, and Sung [55] used the IBM forces, although with an IBM based on a force-feedback approach and a structural solver based on Lagrangian mechanics. A straightforward way to check for issues in the force computation would be to additionally compute the forces the same way as Tullio and Pascazio [29] and compare them to the forces currently computed.

Another possible error source is the coupling method, although Truong *et al.* [4] and Kumar, Seo, and Mittal [3] use similar approaches.

Possibly, the force transfer in 3D described in 3.3 can be improved. Let us assume a square membrane consisting of two triangles. The two Lagrangian forces are normal with respect to the membrane and of equal magnitude. The expected physical behaviour would be to have all nodes of the triangles travel with the same acceleration in the direction of the forces. However, the transfer from the force located at the Lagrangian markers to the nodes introduces unequal accelerations of the nodes, theoretically resulting in a "flapping" movement. Although the fluid damps these oscillations, it might be interesting to see if a different transfer approach influences accuracy or the required timestep size.

The coupled code only works on serial processes, what works well for problem sizes similar to the validation cases, but especially when using multiple membranes, better performance is needed. Since for fluid solvers, commonly a domain composition approach is used, this approach might also be used for the parallelization for cases with multiple membranes [56][30]. Due to the rise of GPGPUs (General Purpose Computation on Graphics Processing Units) and especially due to the use of a structured mesh for the fluid solver, a GPGPU implementation might further improve performance [57].

Especially for simulations with multiple membranes, collisions might have to be considered, to appropriately model physics. Collision modelling consists of two parts: the collision detection and the collision response. Collision detection in general is expensive since it requires an  $\mathcal{O}(n_O^2)$  search, where  $n_O$  is the number of objects. SNMs present an additional difficulty in this regard. Since the shape of a membrane is not fixed, every mesh entity (possibly of different types, depending on the model) has to be checked for collision instead of just the object. For example, when comparing rigid and flexible particles consisting of 320 faces and collision detection is done based on faces, the computational load increases by a factor of roughly  $10^5$ . To tackle this, data structures like Bounding Volume Hierarchies can be employed. A Bounding Volume Hierarchy is a tree-like data structure consisting of volumes, such as axis-aligned bounding boxes, where the root node contains a volume that encloses the entire object. The branches contain more fine-grained volumes. The deeper the data structure goes, the finer the volumes become, and the leaves correspond to a single or a specified number of neighbouring mesh entities. This allows fast collision detection for objects that are far away since only the volumes in the root nodes have to be compared instead of all mesh entities. Also, for objects close to each other or that are actually colliding, the number of comparisons is reduced. Spandan *et al.* [58] proposed a different approach where information is stored about whether Lagrangian markers are within one Eulerian grid cell. The collision response can be modelled, as example, by adding repulsive forces to nodes that get too close to each other [51].

To summarize this work, a SNM has been implemented in C++. For this, mesh data structures for 2D and 3D have been defined. The SNM in a standalone version has been

validated against multiple cantilever beam cases in 2D, which have shown good agreement with literature. The code has been coupled with a fluid solver based on a fractional step method to solve the NS equations and a direct-forcing IBM for FSI. The coupled solver was used to simulate a flexible filament in 2D and a flexible flag in 3D. The results were compared with literature, which has shown that the current method does not work. Since the 2D SNM works in a standalone version, the issues are likely with the coupling, which has to be further investigated.

---

## List of Figures

3.1	A 2D mesh consisting of nodes (red) and edges (black) and the corresponding Lagrangian markers (blue). . . . .	16
3.2	A 3D mesh consisting of nodes (red), edges (black) and faces (grey) and the corresponding Lagrangian markers (blue). . . . .	16
3.3	The setup for the computation of forces due to bending energy includes two edges connecting vertices <b>A</b> , <b>B</b> and <b>C</b> , the two unit normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ and the bending angle $\theta$ . . . . .	18
3.4	The setup for the computation of forces due to bending energy includes two adjacent triangles connecting vertices <b>A</b> , <b>B</b> , <b>C</b> and <b>D</b> , the two unit normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ and the bending angle $\theta$ . . . . .	20
3.5	An overview of the code structure of the SNM solver. . . . .	23
3.6	An overview of the mesh types supported by the mesh generator. . . . .	23
3.7	The IDs of the <i>Membran</i> instances are used to control memory storage. . . . .	24
3.8	The 2D mesh data structure includes vertices (red), linear edges (black), angular edges (blue) and boundary patches (green). . . . .	25
3.9	The class <i>MembranNodes</i> stores the vertices and nodal velocities. The class <i>MembranConnectivity</i> stores the linear edges, angular edges and boundary information. . . . .	25
3.10	The 3D mesh data structure includes vertices (red), edges (black), faces (blue) and boundary patches (green). . . . .	26
3.11	The class <i>MembranNodes</i> stores the vertices and nodal velocities. The class <i>MembranConnectivity</i> stores the edges, faces and boundary information. . . . .	26
4.1	Sketches of the different cantilever beam validation cases. . . . .	27
4.2	The x-position of the tip of the cantilever beam in comparison with the results obtained from Truong <i>et al.</i> [4]. . . . .	29
4.3	The relative error of the x-position of the tip of the cantilever beam with respect to the reference values obtained from Truong <i>et al.</i> [4]. . . . .	31
4.4	The y-position of the tip of the cantilever beam in comparison with the results obtained from Truong <i>et al.</i> [4]. . . . .	32

4.5	The relative error of the y-position of the tip of the cantilever beam with respect to the reference values obtained from Truong <i>et al.</i> [4]. . . . .	34
4.6	Setup of the flexible filament and flexible flag case. . . . .	34
4.7	Comparison of the trailing-edge location over time with results from Tullio and Pascazio [29]. . . . .	35
4.8	Comparison of the flag center cross-section for $B = \{10^{-4}, 10^{-3}, 10^{-2}\}$ . . . .	36
4.9	Comparison of the flag bottom cross-section for $B = \{10^{-4}, 10^{-3}, 10^{-2}\}$ . . . .	37
4.10	Comparison of the flag top cross-section for $B = \{10^{-4}, 10^{-3}, 10^{-2}\}$ . . . . .	38

---

## List of Tables

4.1	The simulation and material parameters of the cantilever beam validation case with damping. . . . .	28
4.2	The simulation and material parameters of the cantilever beam validation case without damping. . . . .	30
4.3	Results for the x-displacement of the cantilever beam validation case without damping in comparison with results obtained from Turek and Hron [54] and Engels [47]. Results are given as $mean \pm amplitude[frequency]$ . . . . .	30
4.4	Results for the y-displacement of the cantilever beam validation case without damping in comparison with results obtained from Turek and Hron [54] and Engels [47]. Results are given as $mean \pm amplitude[frequency]$ . . . . .	30
4.5	The simulation and material parameters of the flexible filament case. . . . .	33
4.6	The simulation and material parameters of the flexible flag case. . . . .	33

---

# List of Algorithms

1	Force integration . . . . .	22
---	-----------------------------	----

---

# Nomenclature

## Latin symbols

$A$  Area

$B$  Bending modulus

$C$  Constant scalar or tensor

$C$  Curvature

$E$  Young modulus

$E$  Green strain tensor

$F$  Force

$Fr$  Froude number

$G$  Shear modulus

$G$  Body force

$K$  Bulk modulus

$L$  Length

$N$  Normal vector

$P$  Surface force

$P$  Point load

$Re$  Reynolds number

$T$  Torque

$U$  Inlet Velocity

$V$  Volume

$W$  Elastic energy

$X$  Position of point in undeformed object

$\tilde{E}$  Scaled Young modulus

$a$  Acceleration

$e$  Almansi strain tensor

$f$  Volume force

$g$  Gravity

$h$  Thickness

$h$  Local average Eulerian mesh spacing

$k$  Spring stiffness

$m$  Mass

$n$  Unit normal vector

$n$  Number of ...

$p$  Pressure

$q$  Shearing stress

$r$  Position

$s$  Arclength

$t$  Time

$u$	Fluid velocity
$u$	Displacement
$v$	Velocity of mass point
$x$	Spring displacement
$x$	Position of point in deformed object

## Greek symbols

$\Gamma$	Model-specific bending coefficient
$\Phi$	Potential
$\Pi$	Total potential energy
$\alpha$	Angle
$\beta$	Nondimensional bending modulus
$\delta$	Kronecker Delta
$\epsilon$	Nondimensional elastic modulus
$\gamma$	Damping coefficient
$\gamma$	Density ratio
$\lambda$	Lame's first constant
$\mu$	Lame's second constant
$\nu$	Kinematic viscosity
$\nu$	Poisson's ratio
$\phi$	Shearing strain
$\rho$	Density
$\sigma$	Stress
$\theta$	Bending angle
$\varepsilon$	Strain

## Subscripts

0	Initial
$A, B, C, D$	Points of edge/triangle pair
$F$	Nonlinear bending energy
$G$	Body force
$L$	Linear bending energy
$P$	Surface force
$T$	Triangle
$a$	Area
$a, b, c$	Edges of a triangle
$b$	Bending
$cb$	Cantilever beam
$e$	Linear spring
$f$	Fluid
$f$	Face
$i, j, k, l, m, n, o$	Indices
$l$	Lagrangian marker
$s$	Solid
$x, y, z$	Coordinate dimensions
axial	Axial direction
trans	Normal direction

## Superscripts

$F$	Nonlinear bending energy
$L$	Linear bending energy
$a$	Area

*b* Bending

*e* Linear spring

ext External

int Internal

## Acronyms

**FSI** fluid-structure interaction

**GPGPU** General Purpose Computation on Graphics Processing Unit

**IBM** immersed boundary method

**LBM** Lattice Boltzmann method

**NS** Navier-Stokes

**SNM** spring-network model

---

## Bibliography

- [1] D. A. Fedosov, B. Caswell, and G. E. Karniadakis, “Systematic coarse-graining of spectrin-level red blood cell models,” *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 29–32, pp. 1937–1948, Jun. 2010. DOI: 10.1016/j.cma.2010.02.001.
- [2] M. Ju *et al.*, “A review of numerical methods for red blood cell flow simulation,” *Computer Methods in Biomechanics and Biomedical Engineering*, vol. 18, no. 2, pp. 130–140, Apr. 2013. DOI: 10.1080/10255842.2013.783574.
- [3] S. Kumar, J.-H. Seo, and R. Mittal, “Computational modelling and analysis of the coupled aero-structural dynamics in bat-inspired wings,” *Journal of Fluid Mechanics*, vol. 1010, May 2025. DOI: 10.1017/jfm.2025.356.
- [4] H. Truong, T. Engels, D. Kolomenskiy, and K. Schneider, “A mass-spring fluid-structure interaction solver: Application to flexible revolving wings,” *Computers & Fluids*, vol. 200, p. 104426, Mar. 2020. DOI: 10.1016/j.compfluid.2020.104426.
- [5] T. Nakata and H. Liu, “A fluid–structure interaction model of insect flight with flexible wings,” *Journal of Computational Physics*, vol. 231, no. 4, pp. 1822–1847, Feb. 2012. DOI: 10.1016/j.jcp.2011.11.005.
- [6] S. K. Chimakurthi, J. Tang, R. Palacios, C. E. S. Cesnik, and W. Shyy, “Computational aeroelasticity framework for analyzing flapping wing micro air vehicles,” *AIAA Journal*, vol. 47, no. 8, pp. 1865–1878, Aug. 2009. DOI: 10.2514/1.38845.
- [7] R. M. MacMECCAN, J. R. CLAUSEN, G. P. NEITZEL, and C. K. AIDUN, “Simulating deformable particle suspensions using a coupled lattice-boltzmann and finite-element method,” *Journal of Fluid Mechanics*, vol. 618, pp. 13–39, Jan. 2009. DOI: 10.1017/s0022112008004011.
- [8] T. Sawada and T. Hisada, “Fluid–structure interaction analysis of the two-dimensional flag-in-wind problem by an interface-tracking ALE finite element method,” *Computers & Fluids*, vol. 36, no. 1, pp. 136–146, Jan. 2007. DOI: 10.1016/j.compfluid.2005.06.007.

- [9] B. S. H. CONNELL and D. K. P. YUE, “Flapping dynamics of a flag in a uniform stream,” *Journal of Fluid Mechanics*, vol. 581, pp. 33–67, May 2007. DOI: 10.1017/S0022112007005307.
- [10] R. Mittal and G. Iaccarino, “IMMERSED BOUNDARY METHODS,” *Annual Review of Fluid Mechanics*, vol. 37, no. 1, pp. 239–261, Jan. 2005. DOI: 10.1146/annurev.fluid.37.061903.175743.
- [11] W. Kim and H. Choi, “Immersed boundary methods for fluid-structure interaction: A review,” *International Journal of Heat and Fluid Flow*, vol. 75, pp. 301–309, Feb. 2019. DOI: 10.1016/j.ijheatfluidflow.2019.01.010.
- [12] C. S. Peskin, “Flow patterns around heart valves: A numerical method,” *Journal of Computational Physics*, vol. 10, no. 2, pp. 252–271, Oct. 1972. DOI: 10.1016/0021-9991(72)90065-4.
- [13] L. J. Fauci and C. S. Peskin, “A computational model of aquatic animal locomotion,” *Journal of Computational Physics*, vol. 77, no. 1, pp. 85–108, Jul. 1988. DOI: 10.1016/0021-9991(88)90158-1.
- [14] R. P. Beyer, “A computational model of the cochlea using the immersed boundary method,” *Journal of Computational Physics*, vol. 98, no. 1, pp. 145–162, Jan. 1992. DOI: 10.1016/0021-9991(92)90180-7.
- [15] D. Goldstein, R. Handler, and L. Sirovich, “Modeling a no-slip flow boundary with an external force field,” *Journal of Computational Physics*, vol. 105, no. 2, pp. 354–366, Apr. 1993. DOI: 10.1006/jcph.1993.1081.
- [16] J. Mohd-Yusof, “Combined immersed-boundary / b-spline methods for simulations of flow in complex geometries,” NASA ARS/Stanford University CTR, Stanford, CA, USA, Tech. Rep., 1997. [Online]. Available: <https://web.stanford.edu/group/ctr/ResBriefs97/myusof.pdf>.
- [17] E. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof, “Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations,” *Journal of Computational Physics*, vol. 161, no. 1, pp. 35–60, Jun. 2000. DOI: 10.1006/jcph.2000.6484.
- [18] M. Uhlmann, “An immersed boundary method with direct forcing for the simulation of particulate flows,” *Journal of Computational Physics*, vol. 209, no. 2, pp. 448–476, Nov. 2005. DOI: 10.1016/j.jcp.2005.03.017.
- [19] M. Vanella and E. Balaras, “A moving-least-squares reconstruction for embedded-boundary formulations,” *Journal of Computational Physics*, vol. 228, no. 18, pp. 6617–6628, Oct. 2009. DOI: 10.1016/j.jcp.2009.06.003.
- [20] X. Provot, “Deformation constraints in a mass-spring model to describe rigid cloth behaviour,” en, *Proceedings of Graphics Interface '95*, vol. Québec, pp. 147–154, 1995. DOI: 10.20380/GI1995.17.

- [21] P. Volino and N. Thalmann, “Developing simulation techniques for an interactive clothing system,” in *Proceedings. International Conference on Virtual Systems and Multi-Media VSMM '97*, IEEE Comput. Soc, 1997, pp. 109–118. DOI: 10.1109/vsmm.1997.622337.
- [22] A. V. Gelder, “Approximate simulation of elastic membranes by triangulated spring meshes,” *Journal of Graphics Tools*, vol. 3, no. 2, pp. 21–41, Jan. 1998. DOI: 10.1080/10867651.1998.10487490.
- [23] Y. Ujihara, M. Nakamura, H. Miyazaki, and S. Wada, “Proposed spring network cell model based on a minimum energy concept,” *Annals of Biomedical Engineering*, vol. 38, no. 4, pp. 1530–1538, Jan. 2010. DOI: 10.1007/s10439-010-9930-8.
- [24] H.-G. Menz and K.-S. Choi, “Simulation of sutures for virtual surgery applications,” in *18th International Conference on Artificial Reality and Telexistence*, Yokohama, Japan, 2008. [Online]. Available: [https://icat.vrsj.org/ICAT2008\\_Proceedings/Papers/PS4\\_3.pdf](https://icat.vrsj.org/ICAT2008_Proceedings/Papers/PS4_3.pdf).
- [25] D. Wu, C. Lv, and Y. Bao, “An improved vascular model based on mass spring model and parameters optimization by gaussian processes,” in *2016 IEEE International Conference on Mechatronics and Automation*, IEEE, Aug. 2016, pp. 2425–2430. DOI: 10.1109/icma.2016.7558946.
- [26] T. Kawai, “New discrete models and their application to seismic response analysis of structures,” *Nuclear Engineering and Design*, vol. 48, no. 1, pp. 207–229, Jun. 1978. DOI: 10.1016/0029-5493(78)90217-0.
- [27] B. Choo, Y. K. Hwang, J. E. Bolander, and Y. M. Lim, “Rigid-body-spring network model for failure simulation of reinforced-concrete members under various loading rates,” *International Journal for Numerical and Analytical Methods in Geomechanics*, vol. 47, no. 12, pp. 2213–2230, May 2023. DOI: 10.1002/nag.3578.
- [28] J. Bolander and S. Saito, “Fracture analyses using spring networks with random geometry,” *Engineering Fracture Mechanics*, vol. 61, no. 5–6, pp. 569–591, Nov. 1998. DOI: 10.1016/s0013-7944(98)00069-1.
- [29] M. de Tullio and G. Pascazio, “A moving-least-squares immersed boundary method for simulating the fluid–structure interaction of elastic bodies with arbitrary thickness,” *Journal of Computational Physics*, vol. 325, pp. 201–225, Nov. 2016. DOI: 10.1016/j.jcp.2016.08.020.
- [30] V. Spandan *et al.*, “A parallel interaction potential approach coupled with the immersed boundary method for fully resolved simulations of deformable interfaces and membranes,” *Journal of Computational Physics*, vol. 348, pp. 567–590, Nov. 2017. DOI: 10.1016/j.jcp.2017.07.036.

- [31] A. Coclite, S. Ranaldo, G. Pascazio, and M. D. de Tullio, “A lattice boltzmann dynamic-immersed boundary scheme for the transport of deformable inertial capsules in low-re flows,” *Computers & Mathematics with Applications*, vol. 80, no. 12, pp. 2860–2876, Dec. 2020. DOI: 10.1016/j.camwa.2020.09.017.
- [32] B. Afra, A. Amiri Delouei, M. Mostafavi, and A. Tarokh, “Fluid-structure interaction for the flexible filament’s propulsion hanging in the free stream,” *Journal of Molecular Liquids*, vol. 323, p. 114941, Feb. 2021. DOI: 10.1016/j.molliq.2020.114941.
- [33] M. Zhang, J. Shen, R. Liu, P. Cui, H. Wu, and Z. Liu, “A numerical study of circulating tumor cell behavior in constricted microvessels based on the immersed boundary-lattice boltzmann method,” *Soft Matter*, vol. 21, no. 24, pp. 4956–4967, 2025. DOI: 10.1039/d5sm00363f.
- [34] D. A. Fedosov, “Multiscale modeling of blood flow and soft matter,” Ph.D. dissertation, Brown University, Providence, RI, USA, 2010. [Online]. Available: [https://www.dam.brown.edu/dpd/lib/exe/fetch.php/wiki:phd\\_thesis:fedosov\\_thesis.pdf](https://www.dam.brown.edu/dpd/lib/exe/fetch.php/wiki:phd_thesis:fedosov_thesis.pdf).
- [35] J. Young, S. M. Walker, R. J. Bomphrey, G. K. Taylor, and A. L. R. Thomas, “Details of insect wing design and deformation enhance aerodynamic function and flight efficiency,” *Science*, vol. 325, no. 5947, pp. 1549–1552, Sep. 2009. DOI: 10.1126/science.1175928.
- [36] D. J. Pines and F. Bohorquez, “Challenges facing future micro-air-vehicle development,” *Journal of Aircraft*, vol. 43, no. 2, pp. 290–305, Mar. 2006. DOI: 10.2514/1.4922.
- [37] G. Berti, “Generic software components for scientific computing,” Ph.D. dissertation, Brandenburgische Technische Universität Cottbus, Cottbus, Germany, 2000. [Online]. Available: [https://www.researchgate.net/publication/239065936\\_Generic\\_software\\_components\\_for\\_Scientific\\_Computing](https://www.researchgate.net/publication/239065936_Generic_software_components_for_Scientific_Computing).
- [38] D. Sieger and M. Botsch, “Design, implementation, and evaluation of the surface\_mesh data structure,” in *Proceedings of the 20th International Meshing Roundtable*. Springer Berlin Heidelberg, 2011, pp. 533–550. DOI: 10.1007/978-3-642-24734-7\_29.
- [39] B. G. Baumgart, “Winged-edge polyhedron representation,” Stanford University, Stanford, CA, USA, Tech. Rep. STAN-CS-72-320, Oct. 1972.
- [40] D. Muller and F. Preparata, “Finding the intersection of two convex polyhedra,” *Theoretical Computer Science*, vol. 7, no. 2, pp. 217–236, 1978. DOI: 10.1016/0304-3975(78)90051-8.
- [41] L. Guibas and J. Stolfi, “Primitives for the manipulation of general subdivisions and the computation of voronoi,” *ACM Transactions on Graphics*, vol. 4, no. 2, pp. 74–123, Apr. 1985. DOI: 10.1145/282918.282923.
- [42] M. Mäntylä, *An introduction to solid modeling* (Principles of Computer Science Series 13). Rockville, MD, USA: Computer Science Pr., 1988.

- [43] S. Pion and M. Yvinec, *2D triangulation data structure*, 6.0.1 ed. (2024), Accessed: Sep. 10, 2025. [Online]. Available: <https://doc.cgal.org/6.0.1/Manual/packages.html#PkgTDS2>.
- [44] J. R. Shewchuk, “Triangle: Engineering a 2d quality mesh generator and delaunay triangulator,” in *Applied Computational Geometry Towards Geometric Engineering*. Springer Berlin Heidelberg, 1996, pp. 203–222. DOI: 10.1007/bfb0014497.
- [45] M. Tanaka, S. Wada, and M. Nakamura, *Computational Biomechanics*. Osaka, Japan: Springer Japan, 2012, vol. 3. DOI: 10.1007/978-4-431-54073-1.
- [46] S. S. Bhavikatti, *Mechanics of solids*. New Delhi, India: New Age International Ltd, 2010.
- [47] T. Engels, “Numerical Modeling of Fluid-Structure Interaction in Bio-Inspired Propulsion,” Ph.D. dissertation, Aix-Marseille Université ; Technische Universität Berlin, 2015. [Online]. Available: [http://aifit.cfd.tu-berlin.de/wordpress/wp-content/uploads/2016/05/ENGELS\\_PhD\\_thesis\\_final.pdf](http://aifit.cfd.tu-berlin.de/wordpress/wp-content/uploads/2016/05/ENGELS_PhD_thesis_final.pdf).
- [48] B. Lloyd, G. Szekely, and M. Harders, “Identification of spring parameters for deformable object simulation,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 5, pp. 1081–1094, Sep. 2007. DOI: 10.1109/tvcg.2007.1055.
- [49] W. Helfrich, “Elastic properties of lipid bilayers: Theory and possible experiments,” *Zeitschrift für Naturforschung C*, vol. 28, no. 11–12, pp. 693–703, Dec. 1973. DOI: 10.1515/znc-1973-11-1209.
- [50] F. Moukalled, L. Mangani, and M. Darwish, *The Finite Volume Method in Computational Fluid Dynamics: An Advanced Introduction with OpenFOAM® and Matlab*. Springer International Publishing, 2016. DOI: 10.1007/978-3-319-16874-6.
- [51] I. Cimrak and I. Jancigova, *Computational Blood Cell Mechanics: Road Towards Models and Biomedical Applications* (Mathematical and Computational Biology Series). Zilina, Slovakia: Taylor & Francis Group, 2018.
- [52] *Programming languages — C++*, ISO/IEC 14882:2014, International Organization for Standardization, Geneva, Switzerland, 2014.
- [53] *The VTK User’s Guide*, 11th ed., Kitware, Inc., Clifton Park, NY, USA, 2010. [Online]. Available: <https://vtk.org/wp-content/uploads/2021/08/VTKUsersGuide.pdf>.
- [54] S. Turek and J. Hron, “Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow,” in *Fluid-Structure Interaction*, ser. Lecture Notes in Computational Science and Engineering, vol. 53, Springer Berlin Heidelberg, 2006, pp. 371–385. DOI: 10.1007/3-540-34596-5\_15.
- [55] W.-X. Huang, S. J. Shin, and H. J. Sung, “Simulation of flexible filaments in a uniform flow by the immersed boundary method,” *Journal of Computational Physics*, vol. 226, no. 2, pp. 2206–2228, Oct. 2007. DOI: 10.1016/j.jcp.2007.07.002.

- [56] M. Uhlmann, “Simulation of Particulate Flows on Multi-Processor Machines with Distributed Memory,” Centro de Investigaciones Energeticas Medioambientales y Tecnologicas (CIEMAT), Madrid, Spain, Tech. Rep. 1039, May 2004.
- [57] F. Viola *et al.*, “FSEI-GPU: GPU accelerated simulations of the fluid–structure–electrophysiology interaction in the left heart,” *Computer Physics Communications*, vol. 273, p. 108 248, Apr. 2022. DOI: 10.1016/j.cpc.2021.108248.
- [58] V. Spandan, D. Lohse, M. D. de Tullio, and R. Verzicco, “A fast moving least squares approximation with adaptive lagrangian mesh refinement for large scale immersed boundary simulations,” *Journal of Computational Physics*, vol. 375, pp. 228–239, Dec. 2018. DOI: 10.1016/j.jcp.2018.08.040.
- [59] M. R. Spiegel, *Mathematical handbook of formulas and tables* (Schaum’s Outline Series), 3rd ed. New York, NY, USA: McGraw-Hill, 1973.

# Appendix A

## Bending forces

### A.1 Bending forces in 2D

In this chapter, a derivation of equations 3.12 based on bending energy is presented. A sketch of the setup is shown in fig. 3.3. From equations 3.10 and 3.11 we know that in general the energy derivative can be written as

$$-\frac{\partial W^b}{\partial \mathbf{r}_i} = -k_b \Gamma \frac{\partial \theta}{\partial \mathbf{r}_i}. \quad (\text{A.1})$$

Using the definition of the dot product, we can write:

$$\frac{\partial \theta}{\partial \mathbf{r}_i} = \frac{\partial}{\partial \mathbf{r}_i} \arccos(\mathbf{n}_1 \cdot \mathbf{n}_2), \quad (\text{A.2})$$

where  $\mathbf{n}_1$  and  $\mathbf{n}_2$  are unit normal vectors. From a table of derivatives [59] follows:

$$\frac{\partial}{\partial \mathbf{r}_i} \arccos(\mathbf{n}_1 \cdot \mathbf{n}_2) = -\frac{1}{\sqrt{1 - (\mathbf{n}_1 \cdot \mathbf{n}_2)^2}} \frac{\partial}{\partial \mathbf{r}_i} (\mathbf{n}_1 \cdot \mathbf{n}_2) = -\frac{1}{\sin(\theta)} \frac{\partial}{\partial \mathbf{r}_i} (\mathbf{n}_1 \cdot \mathbf{n}_2). \quad (\text{A.3})$$

Using the derivative of the dot product [59], we can further write:

$$-\frac{1}{\sin(\theta)} \frac{\partial}{\partial \mathbf{r}_i} (\mathbf{n}_1 \cdot \mathbf{n}_2) = -\frac{1}{\sin(\theta)} \left( \mathbf{n}_1 \cdot \frac{\partial \mathbf{n}_2}{\partial r_i^{(1)}} + \mathbf{n}_2 \cdot \frac{\partial \mathbf{n}_1}{\partial r_i^{(1)}} \right), \quad (\text{A.4})$$

where the superscript denotes the vector component. The unit normal vectors are defined as

$$N_1 = \left(r_{AB}^{(2)}\right)^2 + \left(r_{BA}^{(1)}\right)^2, \quad (\text{A.5a})$$

$$N_2 = \left(r_{BC}^{(2)}\right)^2 + \left(r_{CB}^{(1)}\right)^2, \quad (\text{A.5b})$$

$$\mathbf{n}_1 = \frac{1}{\sqrt{N_1}} \begin{pmatrix} r_{AB}^{(2)} \\ r_{BA}^{(1)} \end{pmatrix}, \quad (\text{A.5c})$$

$$\mathbf{n}_2 = \frac{1}{\sqrt{N_2}} \begin{pmatrix} r_{BC}^{(2)} \\ r_{CB}^{(1)} \end{pmatrix}. \quad (\text{A.5d})$$

Since  $\mathbf{n}_1$  only depends on nodes A and B and  $\mathbf{n}_2$  only depends on nodes B and C, following derivatives evaluate to zero:

$$\frac{\partial \mathbf{n}_2}{\partial r_A^{(1)}} = \frac{\partial \mathbf{n}_2}{\partial r_A^{(2)}} = \frac{\partial \mathbf{n}_1}{\partial r_C^{(1)}} = \frac{\partial \mathbf{n}_1}{\partial r_C^{(2)}} = \mathbf{0}. \quad (\text{A.6})$$

Using the chain and product rule, we can write:

$$\frac{\partial \mathbf{n}_1}{\partial r_A^{(1)}} = \begin{pmatrix} N_1^{-\frac{3}{2}} r_{BA}^{(1)} r_{AB}^{(2)} \\ -N_1^{-\frac{1}{2}} + N_1^{-\frac{3}{2}} \left(r_{BA}^{(1)}\right)^2 \end{pmatrix}, \quad \frac{\partial \mathbf{n}_1}{\partial r_A^{(2)}} = \begin{pmatrix} N_1^{-\frac{1}{2}} - \left(r_{AB}^{(2)}\right)^2 N_2^{-\frac{3}{2}} \\ -N_1^{-\frac{3}{2}} r_{AB}^{(2)} r_{BA}^{(1)} \end{pmatrix}, \quad (\text{A.7a})$$

$$\frac{\partial \mathbf{n}_1}{\partial r_B^{(1)}} = \begin{pmatrix} -N_1^{-\frac{3}{2}} r_{BA}^{(1)} r_{AB}^{(2)} \\ N_1^{-\frac{1}{2}} - N_1^{-\frac{3}{2}} \left(r_{BA}^{(1)}\right)^2 \end{pmatrix}, \quad \frac{\partial \mathbf{n}_1}{\partial r_B^{(2)}} = \begin{pmatrix} -N_1^{-\frac{1}{2}} + \left(r_{AB}^{(2)}\right)^2 N_2^{-\frac{3}{2}} \\ N_1^{-\frac{3}{2}} r_{AB}^{(2)} r_{BA}^{(1)} \end{pmatrix}, \quad (\text{A.7b})$$

$$\frac{\partial \mathbf{n}_2}{\partial r_B^{(1)}} = \begin{pmatrix} N_2^{-\frac{3}{2}} r_{CB}^{(1)} r_{BC}^{(2)} \\ -N_2^{-\frac{1}{2}} + N_2^{-\frac{3}{2}} \left(r_{CB}^{(1)}\right)^2 \end{pmatrix}, \quad \frac{\partial \mathbf{n}_2}{\partial r_B^{(2)}} = \begin{pmatrix} -N_2^{-\frac{1}{2}} - N_2^{-\frac{3}{2}} \left(r_{BC}^{(2)}\right)^2 \\ -N_2^{-\frac{3}{2}} r_{BC}^{(2)} r_{CB}^{(1)} \end{pmatrix}, \quad (\text{A.7c})$$

$$\frac{\partial \mathbf{n}_2}{\partial r_C^{(1)}} = \begin{pmatrix} -N_2^{-\frac{3}{2}} r_{CB}^{(1)} r_{BC}^{(2)} \\ N_2^{-\frac{1}{2}} - N_2^{-\frac{3}{2}} \left(r_{CB}^{(1)}\right)^2 \end{pmatrix}, \quad \frac{\partial \mathbf{n}_2}{\partial r_C^{(2)}} = \begin{pmatrix} -N_2^{-\frac{1}{2}} + N_2^{-\frac{3}{2}} \left(r_{BC}^{(2)}\right)^2 \\ N_2^{-\frac{3}{2}} r_{BC}^{(2)} r_{CB}^{(1)} \end{pmatrix}. \quad (\text{A.7d})$$

From equations A.7 can be seen that some of the terms only differ in sign:

$$\frac{\partial \mathbf{n}_1}{\partial r_A^{(1)}} = -\frac{\partial \mathbf{n}_1}{\partial r_B^{(1)}}, \quad \frac{\partial \mathbf{n}_1}{\partial r_A^{(2)}} = -\frac{\partial \mathbf{n}_1}{\partial r_B^{(2)}}, \quad (\text{A.8a})$$

$$\frac{\partial \mathbf{n}_2}{\partial r_C^{(1)}} = -\frac{\partial \mathbf{n}_2}{\partial r_B^{(1)}}, \quad \frac{\partial \mathbf{n}_2}{\partial r_C^{(2)}} = -\frac{\partial \mathbf{n}_2}{\partial r_B^{(2)}}. \quad (\text{A.8b})$$

From relations A.8 and A.4 then follows:

$$\mathbf{F}_B^b = -\mathbf{F}_A^b - \mathbf{F}_C^b. \quad (\text{A.9})$$

Multiplying A.7 with the corresponding unit normal vector according to A.4 and some rearranging yields for vertex A:

$$\mathbf{n}_2 \cdot \frac{\partial \mathbf{n}_1}{\partial r_A^{(1)}} = \frac{r_{AB}^{(2)}}{(r_{AB}^{(2)})^2 + (r_{BA}^{(1)})^2} \frac{r_{BA}^{(1)} r_{BC}^{(2)} - r_{AB}^{(2)} r_{CB}^{(1)}}{\left[ (r_{AB}^{(2)})^2 + (r_{BA}^{(1)})^2 \right]^{\frac{1}{2}} \left[ (r_{BC}^{(2)})^2 + (r_{CB}^{(1)})^2 \right]^{\frac{1}{2}}}, \quad (\text{A.10a})$$

$$\mathbf{n}_2 \cdot \frac{\partial \mathbf{n}_1}{\partial r_A^{(2)}} = \frac{r_{BA}^{(1)}}{(r_{AB}^{(2)})^2 + (r_{BA}^{(1)})^2} \frac{r_{BA}^{(1)} r_{BC}^{(2)} - r_{AB}^{(2)} r_{CB}^{(1)}}{\left[ (r_{AB}^{(2)})^2 + (r_{BA}^{(1)})^2 \right]^{\frac{1}{2}} \left[ (r_{BC}^{(2)})^2 + (r_{CB}^{(1)})^2 \right]^{\frac{1}{2}}}. \quad (\text{A.10b})$$

The second fraction corresponds to the norm of the cross product between the unit normal vectors:

$$\frac{r_{BA}^{(1)} r_{BC}^{(2)} - r_{AB}^{(2)} r_{CB}^{(1)}}{\left[ (r_{AB}^{(2)})^2 + (r_{BA}^{(1)})^2 \right]^{\frac{1}{2}} \left[ (r_{BC}^{(2)})^2 + (r_{CB}^{(1)})^2 \right]^{\frac{1}{2}}} = |\mathbf{n}_1 \times \mathbf{n}_2| = \sin(\theta) \quad (\text{A.11})$$

From eq. A.5 and eq. A.11 follows:

$$\mathbf{n}_2 \cdot \frac{\partial \mathbf{n}_1}{\partial r_A^{(1)}} = \frac{\mathbf{n}_1^{(1)}}{N_1} \sin(\theta), \quad (\text{A.12a})$$

$$\mathbf{n}_2 \cdot \frac{\partial \mathbf{n}_1}{\partial r_A^{(2)}} = \frac{\mathbf{n}_1^{(2)}}{N_1} \sin(\theta). \quad (\text{A.12b})$$

Plugging equations A.12 into A.4, we can write for vertex A:

$$-\frac{\partial W^b}{\partial \mathbf{r}_i} = k_b \Gamma \frac{\mathbf{n}_1}{N_1}. \quad (\text{A.13})$$

After using the same procedure for vertex C, the equations can be summarized as

$$\mathbf{F}_A^b = k_b \Gamma \frac{\mathbf{n}_1}{N_1}, \quad (\text{A.14a})$$

$$\mathbf{F}_C^b = k_b \Gamma \frac{\mathbf{n}_2}{N_2}, \quad (\text{A.14b})$$

$$\mathbf{F}_B^b = -\mathbf{F}_A^b - \mathbf{F}_C^b. \quad (\text{A.14c})$$