TECHNISCHE UNIVERSITÄT MÜNCHEN

Ingenieurfakultät Bau Geo Umwelt

Lehrstuhl für Statik

Development of Co-Simulation Environment and Mapping Algorithms

Tianyang Wang

Vollständiger Abdruck der von der Ingenieurfakultät Bau Geo Umwelt der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender:

Univ.-Prof. Dr.-Ing. habil. Fabian Duddeck

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. Kai-Uwe Bletzinger
2. Prof. Dr. Riccardo Rossi,
   Universitat Politècnica de Catalunya - BarcelonaTech/Spanien

Die Dissertation wurde am 12.11.2015 bei der Technischen Universität München eingereicht und durch die Ingenieurfakultät Bau Geo Umwelt am 21.04.2016 angenommen.

Zusammenfassung

Ein partitionierter Ansatz zur Lösung multiphysikalischer Probleme resultiert in einer Co-Simulation. Vorhandene Softwareumgebungen hierfür sind beschränkt auf Kopplung von zwei Teilsystemen, wie z.B. in einer Fluid-Struktur-Interaktion, oder spezifische gekoppelte Probleme mehrerer Teilsysteme. Diese Arbeit präsentiert eine neue Softwareumgebung namens EMPIRE zur Lösung genereller multiphysikalischer Problemstellungen. Verwirklicht wird dies durch die Freiheit für den Nutzer flexible Co-Simuloation-Szenarien aufbauen zu können. Die Anwendung von Softwaretechniken, wie z.B. dem Client-Server-Modell, Programmierschnittstelle und Objektorientierung, garantiert zusätzlich Modularität, Erweiterbarkeit und Nutzbarkeit.

Die Anwendung der Kopplungsbedingungen am Interface zweier physikalischer Felder macht auf Grund der unterschiedlichen Netzauflösungen beider Seiten eine Mapping-Technik notwendig. Drei Mapping-Algorithmen für Oberflächennetze werden dafür untersucht: Nearest Element Interpolation, Standard-Mortar Methode und duale Mortar Methode. Probleme der Mortar Methoden bei gekrümmten Kanten werden durch den neu entwickelten Enforced-Consistency-Ansatz gelöst. Außerdem sei erwähnt, dass die duale Mortar Methode zu schwach oszilierenden Ergebnissen führen kann. Basierend auf den Analysen und Testbeispielen wird auch eine allgemeine Regel der Wahl eines passenden Mapping-Algorithmus vorgeschlagen.

Bei Modellierung von Strukturen mit eindimensionalen Balkenelementen werden auf Grund des fehlenden Oberflächennetzes spezielle Mapping-Algorithmen notwendig. Bereits vonhandene Methoden sind beschränkt auf lineare Balkenelementen und kleine Deformationen. Diese Arbeit entwickelt daher den Co-rotating Algorithmus, um auch mit großen Verschiebungen und Rotationen umgehen zu können.

Bisher fand EMPIRE Anwendung in vielen praktischen Problemen, wie diversen Fluid-Struktur-Interaktionen, Simulation einer Windkraftanlage unter Berücksichtigung von Fluid, Struktur, Getriebe/Generator und Regelung sowie der Optimierung von gekoppelten Fluid-Struktur-Problemen. Auf Grund der Flexibilität und leichten Erweiterbarkeit von EMPIRE ist darauf zu hoffen, dass es zu einer weit verbreiteten Plattform für die Entwicklung von neuen Algorithmen und für Simulationen multiphysikalischer Problemstellungen wird.

Abstract

Partitioned strategy for solving multiphysics problems results in co-simulation. Existing software environments of co-simulation are restricted to two-partitions coupled multiphysics problems such as fluid-structure interaction, or more-partitions coupled problems with specific scenarios. This work develops a co-simulation software environment named EMPIRE which can solve a general multiphysics problem. This is realized by allowing the user to set up co-simulation scenarios flexibly. Besides, software techniques such as server-client model, application program interface and object oriented programming are applied to obtain good modularity, extendability and usability.

When applying the coupling conditions on the interface between two physical domains, mapping is required due to different grid resolutions from both sides. Three mapping algorithms dealing with surface meshes are comparatively investigated which are nearest element interpolation, the standard mortar method and the dual mortar method. The problem of the mortar methods at curved edges is solved by the newly developed enforcing consistency approach. It is also found that the dual mortar method can give slightly oscillatory results. Based on the analysis and tests, a general rule of choosing a suitable mapping algorithm in practice is suggested.

When the structure is modeled by 1D beam elements, special mapping algorithms are needed since the surface mesh from the structure is missing. Existing methods in literature work only for linear beam elements with small deformation. This work develops a co-rotating algorithm which can handle large displacements and rotations.

So far, EMPIRE has been applied in many practical multiphysics problems including fluid-structure interaction, simulation of a wind turbine where fluid, structure, gearbox and a control unit are coupled, as well as optimization of fluid-structure coupled problems. Due to the flexibility and extendability of the software, it is hoped that EMPIRE could become a widely-used platform for both algorithm development and simulation of multiphysics problems.

Acknowledgments

I would like to use this opportunity to express my sincere gratitude to the persons who help in the finishing of my PhD work.

First of all, I would like to thank Prof. Dr.-Ing. Kai-Uwe Bletzinger for giving me the chance to do research at the Chair of Structural Analysis as well as to experience more the life in Germany. I am especially grateful for the biggest freedom he gave me in choosing the topics and scheduling the work. I have been learning to develop independent thinking in these years which will have an important influence to my future career. Moreover, I am also grateful to his tolerance for my shortcomings in many aspects. I also would like to thank Dr.-Ing. Roland Wüchner for the organization of my PhD study and the countless time and effort in discussions of research, publications and projects. He is always the first person whom I will look for when I have questions.

Secondly, I would like to thank Prof. Dr. Riccardo Rossi for being my co-examiner and for his interest in my work. Also I want to thank Univ.-Prof. Dr.-Ing. habil. Fabian Duddeck for charging the jury.

Furthermore, I would like to thank all my coworkers at the Chair of Structural Analysis. Thank Majid Hojjat for introducing me to the chair with the master thesis topic fitting my ability. Thank Benedikt Philipp, Michael Andre, Helmut Masching, Andreas Apostolatos, Christopher Lerch and others for the knowledge sharing and selfless helps. Especially I would like to thank Stefan Sicklinger for the cooperation throughout our PhD time from which I benefit so much. I am also thankful for his administration of the supercomputer in our chair so that running large simulations is possible.

Finally, I would like to thank my parents who give me invaluable love in my whole life. Most importantly, I would like to express my deepest gratitude to my wife Zhuoyan for her encouragement, waiting and trust.

Tianyang Wang
Technische Universität München
01.11.2015

Contents

Contents v

1 Introduction 1

2 Fluid-Structure Interaction and Co-Simulation 5
- 2.1 Structural Mechanics . . . 6
  - 2.1.1 Governing Equations . . . 6
  - 2.1.2 Weak Form . . . 8
  - 2.1.3 Discretization in Space and Time . . . 8
- 2.2 Fluid Dynamics . . . 10
  - 2.2.1 Governing Equations in Eulerian Form . . . 10
  - 2.2.2 Governing Equations in ALE Form . . . 12
  - 2.2.3 Discretization in Space and Time . . . 13
  - 2.2.4 Turbulence Modeling . . . 15
- 2.3 Fluid-Structure Interaction . . . 16
  - 2.3.1 Coupling Conditions . . . 16
  - 2.3.2 Coupling Algorithms . . . 17
- 2.4 General Co-Simulation . . . 22
- 2.5 Summary . . . 25

3 Co-Simulation Environment EMPIRE 27
- 3.1 Requirements . . . 29
- 3.2 Design Concepts . . . 29
  - 3.2.1 Client-Server Model . . . 29
  - 3.2.2 Flexible Co-Simulation Scenario . . . 30
  - 3.2.3 Flexible Data Operation . . . 34
- 3.3 Implementation . . . 34
  - 3.3.1 Data Storage . . . 34
  - 3.3.2 Co-Simulation Scenario . . . 35
  - 3.3.3 Coupling Functionalities . . . 36

3.4 Configuration and Runtime Behavior 40
3.5 Summary 41

4 Mapping with Surface Meshes 43
4.1 Interface Coupling Conditions and Discretization 45
4.2 Direct and Conservative Mapping 47
4.3 Mapping Algorithms 48
4.3.1 Nearest Element Interpolation 48
4.3.2 Standard Mortar Method 50
4.3.3 Dual Mortar Method 57
4.3.4 Implementation of Mortar Methods 60
4.3.5 Computational Cost 62
4.4 Convergence Tests 62
4.4.1 Evaluation of Discretization Error and Mapping Error 63
4.4.2 Geometries and Analytical Fields 63
4.4.3 Results and Conclusions 64
4.5 Examples 70
4.5.1 Circular Plate 70
4.5.2 Finer Structure Mesh 75
4.5.3 Wind Turbine Blade 77
4.6 Summary 82

5 Mapping with Beam Elements 83
5.1 Deformation Mapping 85
5.1.1 Determine Cross Section 85
5.1.2 Rigid Body Motion and Coordinates Transformation 88
5.1.3 Interpolation of Rigid Body Motion 90
5.2 Load Mapping 96
5.3 Convergence Tests 97
5.3.1 Bending 98
5.3.2 Bending and Twist 99
5.4 Practical Examples 106
5.5 Summary 111

6 Co-Simulation Examples 113
6.1 Flat Membrane in a Wind Tunnel 114
6.1.1 Experiments 114
6.1.2 Structure Model and Validation 116
6.1.3 Fluid Model and Validation 117

6.1.4 FSI Simulation and Results 124
6.2 NREL Phase VI Wind Turbine 131
6.2.1 Previous Simulation with Shell Elements 131
6.2.2 New Simulation with Beam Elements 135
6.3 Shape Optimization of a Prototypical Hydrofoil 144
6.3.1 Fluid Model 145
6.3.2 Structure Model 145
6.3.3 FSI Simulation 147
6.3.4 Optimization 147
6.4 Summary 154

7 Conclusion and Outlook **155**

A Proof of Inconsistency of Mortar Methods **157**
A.1 Standard Mortar Method 157
A.2 Dual Mortar Method 158

Bibliography **159**

Chapter 1

Introduction

Fluid-structure interaction (FSI) is the interaction between a structure and the fluid around it. Examples of application can be found in civil engineering where lightweight structures vibrate under wind load [59, 60, 95, 69, 11, 91], in aeronautical engineering to study the flutter of wings [89, 26, 18], in marine engineering where interaction happens between ocean waves and ships or other offshore structures [119, 142], and in medical engineering where interaction can happen between arteries and blood flow [9, 126] or between airway and airflow [135]. The works on FSI from the multibody community can be found in e.g. [4, 24, 85, 104]. An FSI problem contains a structure domain, a fluid domain and the interface between them. Each domain is described by its own governing equations, and the interaction is described by the coupling conditions at the interface.

FSI is a typical multiphysics problem with two partitions. Some multiphysics problems can have three and more partitions, such as acoustic-fluid-structure interaction [6, 88], control-fluid-structure interaction [120] and optimization of FSI [66, 93, 5, 65, 1]. Similar to FSI, each domain has its own governing equations, and different domains are coupled with each other through the coupling conditions at the interfaces. All the governing equations and the coupling conditions constitute the global system of equations of a multiphysics problem.

The global system of equations can either be solved with the *monolithic* or the *partitioned* strategy. The monolithic strategy solves the global system as a whole, and the global Jacobian matrix is usually derived to apply the Newton-Raphson method. Due to the complexity it is mainly applied in relatively simple problems, e.g. the geometry is in

1D/2D or the equations of the domains are linear. With the partitioned strategy, the two domains are solved separately and they exchange data at the interface according to the coupling conditions. Existing simulation programs of the single domains can be reused which brings big benefits: on the one hand, the time of software development is largely reduced; on the other hand, the scope of FSI problems that can be solved is broadened. The remaining task is to provide a *co-simulation* software environment where the simulation programs can be coupled together.

There are quite a number of successful co-simulation environments for FSI and general two-partitions coupled problems, which apply modern concepts and techniques such as object oriented programming, application program interface (API), server-client model and peer-to-peer model [76, 55, 57]. However, they either do not work for more-partitions coupled problems or can only deal with such problems of fixed types. This work aims at developing a flexible co-simulation environment which can solve general multiphysics problems. The flexibility means that there should be no restriction on the number of programs that are coupled as well as the coupling process.

Beside the development of a co-simulation environment, this work also researches on *mapping* algorithms. Due to separate discretizations, the grids of two coupled domains at the interface are usually non-matching, so data from one interface mesh has to be *mapped* to the other and vice versa. Note that the mapping here deals with surface coupled problems instead of volume coupled problems. Mapping algorithms usually perform interpolation or solve the weak form of the coupling conditions. When the structure in an FSI problem is modeled by 1D beam elements, special mapping algorithms are needed since the surface mesh from the structure model is missing. Existing mapping algorithms in literature are restricted to linear beam elements with small deformation. This work will investigate and improve mapping algorithms for surface meshes and develop a new mapping algorithm for nonlinear beam elements with large displacements and rotations.

The thesis is outlined as follows:

- Chapter 2 derives the governing equations of structure, fluid and FSI as well as the numerical models of them. Partitioned coupling algorithms for solving FSI problems are presented with different classifications. At the end of the chapter, from FSI co-simulation the concept of general co-simulations is abstracted which can also contain more than two partitions.

- Chapter 3 presents the newly developed co-simulation software environment EMPIRE. It starts with the requirements of the software, and continues with the design concepts that solve the requirements. The implementation is briefly shown with the class diagrams of object oriented programming. The configuration and the runtime behavior of EMPIRE are also presented.

- In Chapter 4, different mapping algorithms dealing with surface meshes are investigated including nearest element interpolation, standard mortar method and dual mortar method. The inconsistency problem of the mortar methods on curved edges is solved by the newly developed enforcing consistency approach. The algorithms are tested on representative geometries as well as on a wind turbine blade.

- In Chapter 5, special mapping algorithms dealing with 1D beam elements are developed including a linearized and a nonlinear algorithm. The nonlinear one is based on the idea of co-rotating formulation in nonlinear structural analysis. Both algorithms are tested in convergence tests on a beam with constant square cross section. The corotating algorithm is also applied on airfoil surfaces under artificially large displacements and rotations.

- Chapter 6 presents three co-simulation examples using EMPIRE. The first one is FSI simulation of a flat membrane in a wind tunnel, and the simulation results are compared with the measurements. The second one is FSI simulation of a wind turbine whose blades are modeled by linear beam elements. The rigid body rotation is realized by transforming the forces to a rotating coordinate system where the beam elements are modeled. The last example is the shape optimization of a prototypical hydrofoil where three codes are coupled including an optimizer, a fluid solver and a structure solver.

- The thesis is concluded in Chapter 7.

Chapter 2

Fluid-Structure Interaction and Co-Simulation

In FSI, the structure and the fluid domains maintain their own dynamical characteristics, meanwhile they interact with each other at the interface which is often called the *wet surface*. Both domains are modeled individually as a continuum which leads to partial differential equations describing the conservation of mass, momentum and energy. Numerical methods for solving the equations are studied by computational structural mechanics (CSM) and computational fluid dynamics (CFD). The finite element method (FEM) is the dominant numerical treatment in CSM while the finite volume method (FVM) is a major treatment in CFD. The interaction is realized by the kinematic and dynamic boundary conditions on the wet surface, which respectively describe the conformity of geometry and the balance of load. These coupling conditions together with the governing equations of both domains constitute the complete description of an FSI problem. The global equation system can be solved in a partitioned way where both domains are solved separately. This results in a co-simulation of programs of the individual domains.

This chapter presents firstly the derivation of governing equations of the structure and the fluid as well as the numerical treatments of them. Then the coupling conditions are derived to complete the establishment of the equation system of FSI, and the solving schemes of it are introduced thereafter. The chapter ends with a discussion of general multiphysics problems which can contain two or more than two partitions. The content of this chapter serves as theoretical background of

the co-simulation software environment EMPIRE, which will be presented in the next chapter.

2.1 Structural Mechanics

The structure is considered undergoing small stains which can be applied for many civil engineering problems. Despite the small strain assumption it can have large displacement and rotation, which requires geometrically nonlinear formulation. The material is further assumed as homogenous and isotropic so that the Saint-Venant-Kirchhoff material law is applied. The derivation of the governing equations is only briefly summarized and the details can be found in many text books e.g. [7], [14] and [140].

2.1.1 Governing Equations

Kinematics

The kinematics describe the deformation of the structure. Here, the Lagrange description is applied where physical fields are defined on the material points of the structure. These material points form the structure domain Ω_s corresponding to the initial configuration. Given a material point $\mathbf{x} \in \Omega_\mathrm{s}$, if its position at the current configuration is denoted by $\mathbf{x}_t$, then the displacement vector on $\mathbf{x}$ is

$$\mathbf{u} = \mathbf{x}_t - \mathbf{x}. \tag{2.1}$$

The relation between an infinitesimal length $\mathrm{d}\mathbf{x}$ at the initial configuration and the length after deformation $\mathrm{d}\mathbf{x}_t$ can be described by the deformation gradient $\mathbf{F}$ as

$$\mathbf{F} = \frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}\mathbf{x}}. \tag{2.2}$$

The Green-Lagrange strain tensor $\mathbf{E}$ that allows large displacement and rotation is defined as

$$\mathbf{E} = \frac{1}{2}(\mathbf{F}^T\mathbf{F} - \mathbf{I}), \tag{2.3}$$

where $\mathbf{I}$ is an identity matrix.

Constitutive equation

The constitutive equation describes the relation between stress and strain. The 2nd Piola-Kirchhoff stress tensor $\mathbf{S}$ is the stress measure that

is defined on the initial configuration. The corresponding strain measure of $\mathbf{S}$ is the Green-Lagrange strain $\mathbf{E}$. Since the linear Saint-Venant-Kirchhoff material is assumed in this work, the constitutive equation is derived as

$$\mathbf{S} = \lambda_\mathrm{s}\, tr(\mathbf{E})\, \mathbf{I} + 2\mu_\mathrm{s}\mathbf{E}. \tag{2.4}$$

The Lamé constants λ_s and μ_s are directly related to the Young's modulus E_s and the Poisson's ratio ν_s as

$$\lambda_\mathrm{s} = \frac{E_\mathrm{s}\nu_\mathrm{s}}{(1+\nu_\mathrm{s})(1-2\nu_\mathrm{s})}, \qquad \mu_\mathrm{s} = \frac{E_\mathrm{s}}{2(1+\nu_\mathrm{s})}. \tag{2.5}$$

Dynamics

The dynamics describe the force balance on the structure which can be derived as

$$\rho_\mathrm{s}\frac{\mathrm{d}^2\mathbf{u}}{\mathrm{d}t^2} = \nabla\cdot(\mathbf{F}\cdot\mathbf{S}) + \rho_\mathrm{s}\mathbf{b}_\mathrm{s}, \tag{2.6}$$

where ρ_s is the density of the structure and $\mathbf{b}_\mathrm{s}$ is the body force. The equation states the balance among the inertia, the internal elastic force and the external body force.

Boundary and initial conditions

The boundaries are classified into two different types namely the *Dirichlet boundaries* Γ_s^u and the *Neumann boundaries* Γ_s^p. The displacement field is specified on Γ_s^u while the traction field is specified on Γ_s^p. The boundary conditions are written as

$$\mathbf{u} = \bar{\mathbf{u}} \qquad \text{at } \Gamma_\mathrm{s}^u, \tag{2.7a}$$

$$\mathbf{n}\cdot(\mathbf{F}\cdot\mathbf{S}) = \bar{\mathbf{p}} \qquad \text{at } \Gamma_\mathrm{s}^p, \tag{2.7b}$$

where $\bar{\mathbf{u}}$ and $\bar{\mathbf{p}}$ are respectively the displacement and traction specified on the corresponding boundaries, while $\mathbf{n}$ is the normal of Γ_s^p. Note that Γ_s^u and Γ_s^p are boundaries of Ω_s so they are also defined at the initial configuration.

Since (2.6) contains a second order time derivative, two initial conditions are needed where both the initial displacement $\mathbf{u}_{\mathrm{s}0}$ and the initial velocity $\mathbf{v}_{\mathrm{s}0}$ are specified as

$$\mathbf{u}(t=0) = \mathbf{u}_{\mathrm{s}0} \qquad \text{in } \Omega_\mathrm{s}, \tag{2.8a}$$

$$\left.\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t}\right|_{t=0} = \mathbf{v}_{\mathrm{s}0} \qquad \text{in } \Omega_\mathrm{s}. \tag{2.8b}$$

2.1.2 Weak Form

In order to solve the problem numerically, the weak form of (2.6) needs to be formulated first as

$$\int_{\Omega_\mathrm{s}} \rho \frac{\mathrm{d}^2\mathbf{u}}{\mathrm{d}t^2} \cdot \delta\mathbf{u}\,\mathrm{d}\Omega_\mathrm{s} + \int_{\Omega_\mathrm{s}} \mathbf{S} : \delta\mathbf{E}\,\mathrm{d}\Omega_\mathrm{s} = \int_{\Omega_\mathrm{s}} \rho\mathbf{b}_\mathrm{s} \cdot \delta\mathbf{u}\,\mathrm{d}\Omega_\mathrm{s} + \int_{\Gamma_\mathrm{s}^p} \bar{\mathbf{p}} \cdot \delta\mathbf{u}\,\mathrm{d}\Gamma_\mathrm{s}^p, \tag{2.9}$$

where $\delta\mathbf{u}$ is the test function which is also interpreted as virtual displacement, and $\delta\mathbf{E}$ can be expressed by $\delta\mathbf{u}$. The terms in (2.9) are individually defined as

$$\delta W_{dyn} = \int_{\Omega_\mathrm{s}} \rho \frac{\mathrm{d}^2\mathbf{u}}{\mathrm{d}t^2} \cdot \delta\mathbf{u}\,\mathrm{d}\Omega_\mathrm{s}, \tag{2.10a}$$

$$\delta W_{int} = \int_{\Omega_\mathrm{s}} \mathbf{S} : \delta\mathbf{E}\,\mathrm{d}\Omega_\mathrm{s}, \tag{2.10b}$$

$$\delta W_{ext} = \int_{\Omega_\mathrm{s}} \rho\mathbf{b}_\mathrm{s} \cdot \delta\mathbf{u}\,\mathrm{d}\Omega_\mathrm{s} + \int_{\Gamma_\mathrm{s}^p} \bar{\mathbf{p}} \cdot \delta\mathbf{u}\,\mathrm{d}\Gamma_\mathrm{s}^p, \tag{2.10c}$$

where δW_{dyn} contains the inertia part of the virtual work, δW_{int} is the internal virtual work and δW_{ext} the external virtual work. Obviously

$$\delta W_{dyn} + \delta W_{int} = \delta W_{ext}. \tag{2.11}$$

2.1.3 Discretization in Space and Time

Spatial discretization

To apply the FEM, the continuous domain Ω_s is disretized into a mesh with elements and nodes. The continuous displacement field $\mathbf{u}(\mathbf{x})$ is approximated by interpolating the discrete displacements $\mathbf{U}$ with the help of shape functions $\mathbf{N}(\mathbf{x})$ as

$$\mathbf{u}_i(\mathbf{x}) = \mathbf{N}(\mathbf{x})^T \mathbf{U}_i, \tag{2.12}$$

where $i = x, y$ or z which denotes a component of the 3D vector.

Applying (2.12) in (2.10) the following equations can be finally obtained:

$$\delta W_{dyn} = \delta\mathbf{U}^T\,\mathbf{M}\ddot{\mathbf{U}}, \tag{2.13a}$$

$$\delta W_{int} = \delta\mathbf{U}^T\,\mathbf{f}^{int}(\mathbf{U}), \tag{2.13b}$$

$$\delta W_{ext} = \delta\mathbf{U}^T\,\mathbf{f}^{ext}, \tag{2.13c}$$

where $\ddot{\mathbf{U}}$ represents the second order time derivative of $\mathbf{U}$, and the expression $\dot{\mathbf{U}}$ that will be introduced later represents the first order time derivative. $\mathbf{M}$ is the mass matrix and $\mathbf{f}^{ext}$ the external force. The internal force $\mathbf{f}^{int}$ is a nonlinear function of $\mathbf{U}$. So the semi-discrete form (only discretized in space) of (2.9) can be written as

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{f}^{int}(\mathbf{U}) = \mathbf{f}^{ext}. \tag{2.14}$$

In practice (2.12) is defined on each element and (2.14) is obtained by assembling the elemental evaluations. Damping effect in the structure can be modeled by adding a damping matrix $\mathbf{C}$ in (2.14) as

$$\mathbf{M}\ddot{\mathbf{U}} + \mathbf{C}\dot{\mathbf{U}} + \mathbf{f}^{int}(\mathbf{U}) = \mathbf{f}^{ext}, \tag{2.15}$$

which results in a more general description.

Time discretization

Up to now, the time derivatives in (2.15) have not been numerically treated yet. Time discretization is also called 'time integration' since the variables evolute to the next time step by integrating the derivatives over the time step length. Two widely-used time integration methods for structural dynamics are presented here namely the Newmark-β method [101] and the Generalized-α method [28].

The Newmark-β method supplements (2.15) with two additional equations:

$$\mathbf{M}\ddot{\mathbf{U}}_n + \mathbf{C}\dot{\mathbf{U}}_n + \mathbf{f}^{int}(\mathbf{U}_n) = \mathbf{f}_n^{ext}, \tag{2.16a}$$

$$\mathbf{U}_n = (\mathbf{U}_{n-1} + \Delta t\dot{\mathbf{U}}_{n-1} + (\frac{1}{2} - \beta)\Delta t^2\ddot{\mathbf{U}}_{n-1}) + \beta\Delta t^2\ddot{\mathbf{U}}_n, \tag{2.16b}$$

$$\dot{\mathbf{U}}_n = (\dot{\mathbf{U}}_{n-1} + (1-\gamma)\Delta t\ddot{\mathbf{U}}_{n-1}) + \gamma\Delta t\ddot{\mathbf{U}}_n, \tag{2.16c}$$

where n is the current time step number with $n = 1, 2, ..., n_{end}$, Δt is the time step length which is assumed constant, and β and γ are the factors that determine the proportion between the quantities at $n-1$ and n.

The generalized-α method reuses 2.16b and 2.16c but modifies 2.16a

as

$$\begin{aligned}
&\mathbf{M}\left[(1-\alpha_m)\ddot{\mathbf{U}}_n + \alpha_m \ddot{\mathbf{U}}_{n-1}\right] \\
&+ \mathbf{C}\left[(1-\alpha_f)\dot{\mathbf{U}}_n + \alpha_f \dot{\mathbf{U}}_{n-1}\right] \\
&+ (1-\alpha_f)\mathbf{f}^{int}(\mathbf{U}_n) + \alpha_f \mathbf{f}^{int}(\mathbf{U}_{n-1}) \\
&= (1-\alpha_f)\mathbf{f}_n^{ext} + \alpha_f \mathbf{f}_{n-1}^{ext},
\end{aligned} \tag{2.17}$$

where α_m and α_f are two additional factors to control the proportion between the quantities at $n-1$ and n.

Both methods are stable and of second order accuracy. Moreover, the generalized-α method owns a characteristic of numerical damping on the high frequency modes. For more details please refer to [82].

After spatial and time discretization, the numerical model of the structure is completely established. Regardless of which time integration method is chosen, the final system of equations is solved repeatedly at each time step with the increment of n. Initial values $\mathbf{U}_0$ and $\dot{\mathbf{U}}_0$ are used in the first iteration to start the solving process. The boundary conditions are applied by fixing the boundary values $\bar{\mathbf{U}}$ and $\bar{\mathbf{f}}^{ext}$. The nonlinear system of equations is solved by the Newton-Raphson method, which is out of the scope of this thesis.

2.2 Fluid Dynamics

This section derives the numerical model of incompressible Newtonian flow. The incompressibility assumption which means that the flow has a constant density can also be valid for compressible flow. For example, wind with velocity under $0.3\,c \approx 100\,\mathrm{m/s}$ can be modeled as an incompressible fluid, where c is the speed of sound. The governing equations of fluids are usually based on the Eulerian description, i.e. the variables are defined on fixed spacial points. But for FSI problems, the fluid boundary at the wet surface is deformable, so the equations based on the Euler description have to be converted to those based on the arbitrary Lagrangian-Eulerian (ALE) description, where the variables are defined on moving points.

2.2.1 Governing Equations in Eulerian Form

The governing equations of incompressible flow are briefly introduced below. For more details the reader is referred to standard textbooks on

fluid mechanics e.g. [83] and [132]. Following the Eulerian description, the fluid fields are defined on fixed spatial points $\mathbf{x}$ with $\mathbf{x} \in \Omega_\mathrm{f}$, where Ω_f denotes the fluid domain.

Conservation of mass

The conservation of mass is written as

$$\nabla \cdot \mathbf{v} = 0, \tag{2.18}$$

where $\mathbf{v}$ denotes the flow velocity. The equation specifies that the amount of fluid coming into an infinitesimal volume per unit time is equal to that coming out of the volume.

Conservation of momentum

The conservation of momentum is written as

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot (\mathbf{v}\mathbf{v}) = \frac{1}{\rho_\mathrm{f}} \nabla \cdot \sigma_\mathrm{f} + \mathbf{b}_\mathrm{f}, \tag{2.19}$$

where ρ_f denotes the flow density, σ_f the Cauchy stress tensor and $\mathbf{b}_\mathrm{f}$ the body force. For Newtonian flow, the Cauchy stress tensor is consisted of a pressure part and a viscous stress part as

$$\sigma_\mathrm{f} = -p\mathbf{I} + \mu_\mathrm{f} \left[\nabla \mathbf{v} + (\nabla \mathbf{v})^T - \frac{2}{3} (\nabla \cdot \mathbf{v}) \mathbf{I} \right], \tag{2.20}$$

where p is the pressure and μ_f the dynamic viscosity. Combine 2.19 and 2.20 it is obtained that

$$\frac{\partial \mathbf{v}}{\partial t} + \nabla \cdot (\mathbf{v}\mathbf{v}) = \nabla \cdot (\nu_\mathrm{f} \nabla \mathbf{v}) - \frac{1}{\rho_\mathrm{f}} \nabla p + \mathbf{b}_\mathrm{f}, \tag{2.21}$$

with the kinematic viscosity $\nu_\mathrm{f} = \mu_\mathrm{f} / \rho_\mathrm{f}$.

Boundary and initial conditions

As in structural mechanics, the boundaries are also classified into Dirichlet boundaries Γ_f^v where velocity $\bar{\mathbf{v}}$ is specified and Neumann boundaries Γ_f^p where traction $\bar{\mathbf{p}}$ is specified. The boundary conditions are written as

$$\mathbf{v} = \bar{\mathbf{v}} \qquad \text{at } \Gamma_\mathrm{f}^v, \tag{2.22a}$$

$$\mathbf{n} \cdot \sigma_\mathrm{f} = \bar{\mathbf{p}} \qquad \text{at } \Gamma_\mathrm{f}^p. \tag{2.22b}$$

The initial conditions set the initial velocity and pressure fields as

$$\mathbf{v}(t=0) = \mathbf{v}_{\mathrm{f}0} \qquad \text{in } \Omega_{\mathrm{f}}, \tag{2.23a}$$

$$p(t=0) = p_{\mathrm{f}0} \qquad \text{in } \Omega_{\mathrm{f}}. \tag{2.23b}$$

2.2.2 Governing Equations in ALE Form

The arbitrary Lagrangian-Eulerian (ALE) description [64, 38, 39] introduces new observation points which can move in space. And the fluid fields are defined on the moving points. If these points follow the motion of the material points, it becomes the Lagrangian description as in structural mechanics; if they are fixed in space, it becomes the Eulerian description.

The displacement of an observation point is defined as

$$\mathbf{u}_g = \mathbf{x}_t - \mathbf{x}_0, \tag{2.24}$$

where $\mathbf{x}_t$ is the current spatial position and $\mathbf{x}_0$ the initial position. Note that $\mathbf{x}_0$ is understood as a fixed observation point while $\mathbf{x}_t$ the current spatial coordinates. The velocity of an observation point is defined by

$$\mathbf{v}_g = \frac{\mathrm{d}\,\mathbf{u}_g}{\mathrm{d}\,t}. \tag{2.25}$$

The conservation of momentum in (2.21) can be transformed into ALE form as

$$\left.\frac{\partial \mathbf{v}}{\partial t}\right|_{\mathbf{x}_0} + (\mathbf{v} - \mathbf{v}_g)\cdot\nabla\mathbf{v} = \nabla\cdot(\nu_{\mathrm{f}}\nabla\mathbf{v}) - \frac{1}{\rho_{\mathrm{f}}}\nabla p + \mathbf{b}_{\mathrm{f}}. \tag{2.26}$$

The following points are to be noted:

- The time derivative is defined on a fixed observing point $\mathbf{x}_0$.
- The spatial derivatives are evaluated with respect to $\mathbf{x}_t$ instead of $\mathbf{x}_0$ because of their definitions. Although fluid variables are defined on $\mathbf{x}_0$, but they can be converted onto $\mathbf{x}_t$ using the relation between $\mathbf{x}_0$ and $\mathbf{x}_t$.

Redefine the fluid domain Ω_{f} by the current positions of the observation points with $\Omega_{\mathrm{f}} \ni \mathbf{x}_t$, so Ω_{f} can move now. The conservation of mass in (2.18), the boundary conditions in (2.22) and the initial conditions in (2.23) which are defined in Ω_{f} remain the same form.

To complete (2.26), $\mathbf{v}_g$ or $\mathbf{u}_g$ should be provided. This work considers only the computation of $\mathbf{u}_g$. Normally, the displacement of the moving boundaries Γ_{g}^u is given, so the task is to find an equation that governs the boundary value problem. One choice is the Laplace's equation as

$$\nabla \cdot (\Gamma_g \nabla \mathbf{u}_g) = 0, \tag{2.27}$$

where Γ_g is the diffusivity. It can be uniform or a function of the distance to the boundaries [75]. The boundary condition is written as

$$\mathbf{u}_g = \bar{\mathbf{u}}_g \qquad \text{on } \Gamma_{\mathrm{f}}^u, \tag{2.28}$$

where Γ_{f}^u denotes the moving boundaries. In case of FSI, Γ_{f}^u are the wet surfaces. Moreover, they are part of Γ_{f}^v, because the fluid velocity on the wet surfaces should be equal to the velocity of the "walls". This can be expressed as

$$\mathbf{v} = \frac{\mathrm{d}\,\bar{\mathbf{u}}_g}{\mathrm{d}\,t} \qquad \text{on } \Gamma_{\mathrm{f}}^u \ (\Gamma_{\mathrm{f}}^u \in \Gamma_{\mathrm{f}}^v). \tag{2.29}$$

The movement of the fluid domain is also called *mesh motion*, since the observing points become the nodes after meshing. An equation governing the grid motion like 2.27 is often called a *mesh motion solver*.

2.2.3 Discretization in Space and Time

Spatial discretization

The FVM is based on the integral form of the governing equations. An integral form of (2.21) based on the Eulerian description can be obtained by integrating each term in the equation on a finite volume V as

$$\int_V \frac{\partial \mathbf{v}}{\partial t}\,\mathrm{d}V + \int_V \nabla \cdot (\mathbf{v}\mathbf{v})\,\mathrm{d}V = \int_V \nabla \cdot (\nu_{\mathrm{f}} \nabla \mathbf{v})\,\mathrm{d}V - \int_V \frac{1}{\rho_{\mathrm{f}}} \nabla p\,\mathrm{d}V + \int_V \mathbf{b}_{\mathrm{f}}\,\mathrm{d}V. \tag{2.30}$$

Let V_t denote a moving finite volume. With the Reynold's transport theorem

$$\frac{\partial}{\partial t}\int_{V_t} \mathbf{v}\,\mathrm{d}V_t = \int_{V_t} \frac{\partial \mathbf{v}}{\partial t}\,\mathrm{d}V_t + \int_{V_t} \nabla \cdot (\mathbf{v}_g \mathbf{v})\,\mathrm{d}V_t, \tag{2.31}$$

the integral form of the conservation of momentum based on the ALE description is obtained as

$$\frac{\partial}{\partial t}\int_{V_t}\mathbf{v}\,\mathrm{d}V_t+\int_{V_t}\nabla\cdot[(\mathbf{v}-\mathbf{v}_g)\mathbf{v}]\,\mathrm{d}V_t= \int_{V_t}\nabla\cdot(\nu_\mathrm{f}\nabla\mathbf{v})\,\mathrm{d}V_t-\int_{V_t}\frac{1}{\rho_\mathrm{f}}\nabla p\,\mathrm{d}V_t+\int_{V_t}\mathbf{b}_\mathrm{f}\,\mathrm{d}V_t. \quad (2.32)$$

And the integral form of the conservation of mass is written as

$$\int_{V_t}\nabla\cdot\mathbf{v}\,\mathrm{d}V_t=0. \quad (2.33)$$

Gauss' divergence theorem is applied in (2.33) and (2.32) so they can be simplified as

$$\int_{S_t}\mathbf{v}\cdot\mathbf{n}\,\mathrm{d}S_t=0, \quad (2.34a)$$

$$\frac{\partial}{\partial t}\int_{V_t}\mathbf{v}\,\mathrm{d}V_t+\int_{S_t}[(\mathbf{v}-\mathbf{v}_g)\mathbf{v}]\cdot\mathbf{n}\,\mathrm{d}S_t= \int_{S_t}(\nu_\mathrm{f}\nabla\mathbf{v})\cdot\mathbf{n}\,\mathrm{d}S_t-\int_{S_t}\frac{1}{\rho_\mathrm{f}}p\mathbf{n}\,\mathrm{d}S_t+\int_{V_t}\mathbf{b}_\mathrm{f}\,\mathrm{d}V_t, \quad (2.34b)$$

where S_t is the surface of V_t and $\mathbf{n}$ the normal of $\mathrm{d}S_t$.

The grid motion equation in (2.27) can also be rewritten in the same way, which results in

$$\int_S(\Gamma_g\nabla\mathbf{u}_g)\cdot\mathbf{n}\,\mathrm{d}S=0. \quad (2.35)$$

With FVM, the fluid domain is spatially descritized into cells of certain types, e.g. tetrahedron or hexahedron. Equations (2.34) and (2.35) are evaluated on each cell and all cell-based evaluations are finally assembled to construct the global equation. The methods of evaluating the integrals are briefly explained below:

- Volume integrals are approximated by the multiplication between the integrand evaluated at the cell center and the volume of the cell.
- Surface integrals are calculated by summing up the evaluations on all faces of the cell. On each face, $\mathrm{d}S_t$ is approximated by the area of a face and $\mathbf{n}$ becomes the face normal. The values at the

face centers are interpolated with schemes such as central differencing, upwind differencing, the hybrid method, total variation diminishing and so on. For details the reader is referred to [132].

Time discretization

Time discretization needs to be applied for the time derivative term in (2.34b). Two methods are presented here, namely the first and the second order backward differencing formula (BDF1 and BDF2).

Let n denote the current time step, and Δt the constant time step length. The time derivative of an arbitrary variable ϕ is approximated via BDF1 as

$$\frac{\partial \phi}{\partial t} = \frac{1}{\Delta t}(\phi^n - \phi^{n-1}). \tag{2.36}$$

Note that BDF1 is also called backward Euler. While it is approximated via BDF2 as

$$\frac{\partial \phi}{\partial t} = \frac{1}{2\Delta t}(3\phi^n - 4\phi^{n-1} + \phi^{n-2}). \tag{2.37}$$

In (2.36) and (2.37), ϕ can be replaced by $\int_{V_t} \mathbf{v}\,\mathrm{d}V_t$ to approximate the time derivative term in (2.34b). Also, it can be replaced by $\mathbf{u}_g$ to approximate $\mathbf{v}_g$ in (2.25).

The nonlinear discretized momentum equation can be solved with the pressure-correction algorithms such as SIMPLE, PISO or their variants. The PIMPLE algorithm [72] as a variant of PISO is the default choice in this work.

2.2.4 Turbulence Modeling

Flow turbulence is a complex phenomenon and still a challenging topic in CFD. According to the Boussinesq assumption, it can be modeled by additional viscous stresses in the momentum equation, i.e. ν_f is replaced by $\nu_\mathrm{f} + \nu_\mathrm{ft}$ in (2.21), where ν_ft is the *turbulence viscosity*. ν_ft can be calculated with methods of reynolds-averaged Navier-Stokes (RANS) or methods of large-eddy simulation (LES). Different methods of turbulence modeling are reviewed in e.g. [107], [123] and [138]. As a method of RANS, the k-ω-SST [94] is chosen as the default turbulence model in this work which can give satisfactory results in both the near-wall region and the fully turbulent region.

2.3 Fluid-Structure Interaction

The numerical model of an FSI problem is established by coupling the fluid and the structure at their interface. The coupling conditions and the solution schemes are presented in this section.

2.3.1 Coupling Conditions

The kinematic and the dynamic coupling conditions on the wet surface denoted by Γ are respectively defined as

$$\mathbf{u}_\mathrm{f} = \mathbf{u}_\mathrm{s} \qquad \text{on } \Gamma, \tag{2.38a}$$

$$\mathbf{p}_\mathrm{f} = \mathbf{p}_\mathrm{s} \qquad \text{on } \Gamma, \tag{2.38b}$$

where $\mathbf{u}_\mathrm{f}$ and $\mathbf{u}_\mathrm{s}$ are the displacements from the fluid and the structure respectively, $\mathbf{p}_\mathrm{f}$ and $\mathbf{p}_\mathrm{s}$ the tractions from them respectively. The kinematic condition in (2.38a) describes that the fluid and the structure domains deform consistently at the wet surface without overlaps or gaps. It can be alternatively expressed with velocity which is equivalent. The dynamic condition in (2.38b) specifies the load balance at the interface. One important remark is that Γ corresponds to the initial configuration. Therefore, the traction in the fluid domain should be transformed to the initial configuration if necessary before applying the dynamic coupling condition.

The coupling conditions are applied to the individual domains as boundary conditions. But it should be noted that, only one of the two conditions in (2.38) is allowed on a boundary, since Dirichlet boundaries and Neumann boundaries cannot overlap. This results in a Dirichlet-Neumann partitioning. The structure domain usually becomes the Neumann partition, i.e. it takes $\mathbf{p}_\mathrm{f}$ as the Neumann boundary condition (see (2.7b)) and $\mathbf{u}_\mathrm{s}$ as the unknown to be solved; the fluid domain becomes the Dirichlet partition, i.e. it takes $\mathbf{u}_\mathrm{s}$ as the Dirichlet boundary condition not only for the mesh motion solver in (2.28) but also for the momentum equation in (2.29), and $\mathbf{p}_\mathrm{f}$ becomes the unknown to be solved.

Mapping between non-matching grids

Due to separate discretizations, the grids at the wet surface from the fluid and the structure are generally non-matching. Therefore, a *mapping* technique is required to apply the coupling conditions on the non-

matching grids. Mapping is one main topic of this thesis, which will be presented in detail in Chapter 4 and 5.

2.3.2 Coupling Algorithms

The FSI problem can be solved either with the monolithic strategy or the partitioned strategy. With the monolithic strategy, the global Jacobian matrix is usually derived so that the two partitions are solved simultaneously or in a block-sequential way by Newton-Raphson iterations [48, 125, 115]. Since the Jacobian matrices are not always easy to derive, the monolithic strategy is often applied to specific FSI problems e.g. the piston problem in 1D [17, 96, 130] and arterial simulations in 1D/2D [63, 68]. With the partitioned strategy, the two partitions are solved separately, i.e. they advance separately in time. As a result, the fluid and the structure partitions can be simply expressed as black box solvers whose inputs and outputs are the interface variables as

$$\mathbf{f} = F(\mathbf{s}), \tag{2.39a}$$

$$\mathbf{s} = S(\mathbf{f}). \tag{2.39b}$$

In the equation, F represents the fluid solver that takes the motion of the wet surface $\mathbf{s}$ (displacements or velocities) as input and outputs the load $\mathbf{f}$ (nodal forces or tractions), while S represents the structure solver that takes $\mathbf{f}$ as input and outputs $\mathbf{s}$. Note that when the output from one solver is assigned to the other solver as input, mapping is needed. Compared with the monolithic strategy, the advantage of the partitioned strategy is that existing fluid and structure codes can be reused.

A solution scheme of (2.39) is called a *coupling algorithm* here. There are different ways to categorize coupling algorithms:

- According to whether the Jacobian matrices are used, they can be categorized into fixed point iteration and Newton-Raphson iteration.
- According to whether the solvers are run sequentially or in parallel, they can be categorized into block Gauss-Seidel and block Jacobi.
- According to the number of iterations, they can be categorized into loose coupling and iterative coupling.

The details are given below.

Fixed point iteration vs. Newton-Raphson iteration

A fixed point iteration can be formulated either with one of $\mathbf{s}$ and $\mathbf{f}$ or both. For example, a fixed point iteration can be formulated as

$$\mathbf{s} = S \circ F(\mathbf{s}). \tag{2.40}$$

The equation means that the fluid partition advances in time first with given $\mathbf{s}$, then the structure partition advances in time using the output from the fluid partition $\mathbf{f}$. The output $\mathbf{s}$ is fed back in an iterative way. The residual vector of one iteration is defined as

$$\mathbf{r} = S \circ F(\mathbf{s}) - \mathbf{s}. \tag{2.41}$$

When $\|\mathbf{r}\|$ is below a given tolerance ϵ, the fixed point iteration is converged and all the variables at the current time step are solved. The fixed point iteration can also be defined with $\mathbf{f}$, but the one in (2.40) is used more often in practice, since an FSI simulation usually starts with initial displacements equal to zero. This is also regarded as a *block Gauss-Seidel* process, since the two solvers run in sequence.

Another fixed point iteration can be formulated with both $\mathbf{s}$ and $\mathbf{f}$ as

$$\mathbf{f} = F(\mathbf{s}), \tag{2.42a}$$
$$\mathbf{s} = S(\mathbf{f}). \tag{2.42b}$$

The equations look the same as the definition of the black boxes in (2.39), but they are interpreted here in a way that the outputs of the solvers are fed back into them iteratively. In this case the residual vectors can be defined as

$$\mathbf{r}_\mathrm{f} = \mathbf{f} - F(\mathbf{s}), \tag{2.43a}$$
$$\mathbf{r}_\mathrm{s} = \mathbf{s} - S(\mathbf{f}). \tag{2.43b}$$

When $\|\mathbf{r}_\mathrm{f}\|$ and $\|\mathbf{r}_\mathrm{s}\|$ are below given tolerances ϵ_f and ϵ_s, the fixed point iteration is converged and all the variables at the current time step are solved. This is also regarded as a *block Jacobi* process, since both solvers run in parallel. Compared with the block Gauss-Seidel method, one solver does not use the new output from the other immediately. Although the solvers can run in parallel, but this method is still slower if the computation time for the structure partition is much smaller than that for the fluid partition, which is a more usual case in FSI.

The Newton-Raphson iteration converges normally faster than the fixed point iteration since the Jacobian matrices are used. There are also a block Gauss-Seidel way and a block Jocobi way to formulate the Newton-Raphson iteration. The block Gauss-Seidel way uses the definition of residual in (2.41), and the corresponding Jacobian matrix can be derived as

$$\mathbf{J_r} = \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\mathbf{s}} = \frac{\mathrm{d}S \circ F(\mathbf{s})}{\mathrm{d}\mathbf{s}} - \mathbf{I} = \frac{\mathrm{d}S}{\mathrm{d}\mathbf{f}} \frac{\mathrm{d}F}{\mathrm{d}\mathbf{s}} - \mathbf{I}, \tag{2.44}$$

so the Newton-Raphson equation can be written as

$$\mathbf{J_r}\, \Delta\mathbf{s} = -\mathbf{r}. \tag{2.45}$$

The block Jacobi way uses the residuals defined in (2.43), and the corresponding global Jacobian matrix can be derived as

$$\begin{bmatrix} \mathbf{J}_{\mathbf{r}_{\mathrm{f,f}}} & \mathbf{J}_{\mathbf{r}_{\mathrm{f,s}}} \\ \mathbf{J}_{\mathbf{r}_{\mathrm{s,f}}} & \mathbf{J}_{\mathbf{r}_{\mathrm{s,s}}} \end{bmatrix} = \begin{bmatrix} \frac{\partial \mathbf{r}_{\mathrm{f}}}{\partial \mathbf{f}} & \frac{\partial \mathbf{r}_{\mathrm{f}}}{\partial \mathbf{s}} \\ \frac{\partial \mathbf{r}_{\mathrm{s}}}{\partial \mathbf{f}} & \frac{\partial \mathbf{r}_{\mathrm{s}}}{\partial \mathbf{s}} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\frac{\mathrm{d}F}{\mathrm{d}\mathbf{s}} \\ -\frac{\mathrm{d}S}{\mathrm{d}\mathbf{f}} & \mathbf{I} \end{bmatrix}, \tag{2.46}$$

so the Newton-Raphson equation in this case can be written as

$$\begin{bmatrix} \mathbf{J}_{\mathbf{r}_{\mathrm{f,f}}} & \mathbf{J}_{\mathbf{r}_{\mathrm{f,s}}} \\ \mathbf{J}_{\mathbf{r}_{\mathrm{s,f}}} & \mathbf{J}_{\mathbf{r}_{\mathrm{s,s}}} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{f} \\ \Delta\mathbf{s} \end{pmatrix} = \begin{pmatrix} -\mathbf{r}_{\mathrm{f}} \\ -\mathbf{r}_{\mathrm{s}} \end{pmatrix}. \tag{2.47}$$

Note that if setting $\mathbf{r}_{\mathrm{f}} = 0$ in the above equation, the block Jacobi method becomes the block Gauss-Seidel method.

Equations above are very similar to the equations resulting from the monolithic strategy. However, these equations are expressed with the interface variables but the equations from the monolithic strategy contain also the variables inside the fluid and the structural domains. It can be seen that the Jacobian matrices $\frac{\mathrm{d}F}{\mathrm{d}\mathbf{s}}$ and $\frac{\mathrm{d}S}{\mathrm{d}\mathbf{f}}$ are the key of building the Newton-Raphson equations. However, they are usually not available so they have to be approximated with certain algorithms, e.g. the interface block quasi-Newton method [133], interface quasi-Newton methods [35, 34], vector extrapolation [84] and interface Newton-Krylov/GMRES methods [97, 116, 92, 80]. With these algorithms, the solvers are still regarded as black boxes, and the Jacobian matrices are approximated based on the results from several runs.

Block Gauss-Seidel vs. block Jacobi

Regardless of fixed point iteration or Newton-Raphson iteration, the black box solvers run iteratively, and a general work flow of block Gauss-Seidel fashioned coupling algorithms can be extracted which is presented

in Algorithm 1. Analogously, a general work flow of block Jacobi fashioned coupling algorithms is also extracted which is presented in Algorithm 2. In these algorithms, n denotes the current time step and m the current iteration, and the inputs have additional hats in the notations such as $\hat{\mathbf{s}}$ and $\hat{\mathbf{f}}$.

Algorithm 1 General work flow of a block Gauss-Seidel coupling algorithm

for $n = 1$ to n_{end} **do**
  m=1
  Compute initial guess ${}^{m}\hat{\mathbf{s}}^{n}$ (extrapolation)
  while $\|{}^{m}\mathbf{r}^{n}\| > \epsilon$ **do**
    ${}^{m}\mathbf{f}^{n} = F({}^{m}\hat{\mathbf{s}}^{n})$
    ${}^{m}\mathbf{s}^{n} = S({}^{m}\mathbf{f}^{n})$
    ${}^{m}\mathbf{r}^{n} = {}^{m}\mathbf{s}^{n} - {}^{m}\hat{\mathbf{s}}^{n}$
    Compute ${}^{m+1}\hat{\mathbf{s}}^{n}$ based on $\{ {}^{1}\mathbf{r}^{n}, {}^{2}\mathbf{r}^{n}, \ldots , {}^{m}\mathbf{r}^{n} \}$ and $\{ {}^{1}\mathbf{s}^{n}, {}^{2}\mathbf{s}^{n}, \ldots , {}^{m}\mathbf{s}^{n} \}$ (relaxation, Newton-Raphson, etc.)
    m=m+1
  end while
end for

Algorithm 2 General work flow of a block Jacobi coupling algorithm

for $n = 1$ to n_{end} **do**
  m=1
  Compute initial guess ${}^{m}\hat{\mathbf{s}}^{n}$ and ${}^{m}\hat{\mathbf{f}}^{n}$ (extrapolation)
  while $\|{}^{m}\mathbf{r}_{\mathrm{s}}^{n}\| > \epsilon_{\mathrm{s}}$ or $\|{}^{m}\mathbf{r}_{\mathrm{f}}^{n}\| > \epsilon_{\mathrm{f}}$ **do**
    ${}^{m}\mathbf{f}^{n} = F({}^{m}\hat{\mathbf{s}}^{n})$
    ${}^{m}\mathbf{s}^{n} = S({}^{m}\hat{\mathbf{f}}^{n})$
    ${}^{m}\mathbf{r}_{\mathrm{s}}^{n} = {}^{m}\mathbf{s}^{n} - {}^{m}\hat{\mathbf{s}}^{n}$
    ${}^{m}\mathbf{r}_{\mathrm{f}}^{n} = {}^{m}\mathbf{f}^{n} - {}^{m}\hat{\mathbf{f}}^{n}$
    Compute ${}^{m+1}\hat{\mathbf{s}}^{n}$ and ${}^{m+1}\hat{\mathbf{f}}^{n}$ based on $\{ {}^{1}\mathbf{r}_{\mathrm{s}}^{n}, {}^{2}\mathbf{r}_{\mathrm{s}}^{n}, \ldots , {}^{m}\mathbf{r}_{\mathrm{s}}^{n} \}$, $\{ {}^{1}\mathbf{s}^{n}, {}^{2}\mathbf{s}^{n}, \ldots , {}^{m}\mathbf{s}^{n} \}$, $\{ {}^{1}\mathbf{r}_{\mathrm{f}}^{n}, {}^{2}\mathbf{r}_{\mathrm{f}}^{n}, \ldots , {}^{m}\mathbf{r}_{\mathrm{f}}^{n} \}$ and $\{ {}^{1}\mathbf{f}^{n}, {}^{2}\mathbf{f}^{n}, \ldots , {}^{m}\mathbf{f}^{n} \}$ (relaxation, Newton-Raphson, etc.)
    m=m+1
  end while
end for

Loose coupling vs. iterative coupling

The coupling algorithms can be further categorized into two types:

- loose/staggered/explicit coupling which means that only one iteration is performed for each time step (the while loops in Algorithm 1 or 2 are executed only once);
- iterative/implicit coupling which means that iterations are performed until convergence.

Loose coupling algorithms suffer from instability problems [110, 17, 109, 54, 77] since the interface residual is not converged. To improve the stability, an initial guess at the beginning of each time step is computed by extrapolating from the data of previous time steps, as shown at line 3 of Algorithm 1 and 2. This procedure is termed as *extrapolation* or *prediction* in literature. Examples of prediction can be found in the improved serial staggered algorithm with a leap-frog type procedure [41, 109], or in the generalized serial staggered algorithm [43] where velocity and acceleration of the wet surface are used. These algorithms are found to be more stable from practical aeroelastic tests than those without prediction. Obviously, a closer initial guess results in a smaller interface residual and hence bigger stability. A simple but effective linear extrapolation is used in this work as (time step length is assumed constant):

$$^{1}\hat{\mathbf{s}}^{n} = 2\mathbf{s}^{n-1} - \mathbf{s}^{n-2}. \tag{2.48}$$

Iterative coupling algorithms can also use extrapolation to improve the initial guess. Moreover, relaxation or Newton-Raphson can be used during the iterative process to reduce the number of iterations, as shown at line 8 in Algorithm 1 and at line 9 in Algorithm 2. Examples of effective iterative coupling algorithms in literature include Aitken relaxation [71, 99], vector extrapolation, interface Newton-Krylov methods and interface quasi-Newton methods. Comparison of these algorithms with benchmark examples can be found in [99, 35, 55, 36], where the results show that the Aitken relaxation is competitive with the others in many cases. Due to its simplicity of implementation and efficiency, the Aitken relaxation is chosen as the default iterative coupling algorithm in this work. It is implemented in a block Gauss-Seidel fashion, and computes the new input $^{m+1}\hat{\mathbf{s}}^{n}$ with a relaxation factor $^{m}\omega$ as

$$^{m+1}\hat{\mathbf{s}} = (1 - {}^{m}\omega)\,{}^{m}\hat{\mathbf{s}} + {}^{m}\omega\,{}^{m}\mathbf{s} = {}^{m}\hat{\mathbf{s}} + {}^{m}\omega{}^{m}\mathbf{r}, \tag{2.49}$$

where the superscript n is dropped for simplicity in notation. When $m = 1$, a user defined initial relaxation factor ${}^{1}\omega$ is used; otherwise, the relaxation factor is calculated dynamically as

$$ {}^{m}\omega = -{}^{m-1}\omega \frac{{}^{m-1}\mathbf{r}^{T}({}^{m}\mathbf{r} - {}^{m-1}\mathbf{r})}{({}^{m}\mathbf{r} - {}^{m-1}\mathbf{r})^{T}({}^{m}\mathbf{r} - {}^{m-1}\mathbf{r})}. \tag{2.50} $$

If the relaxation factor is fixed instead of computed dynamically, it is called *constant relaxation*.

2.4 General Co-Simulation

FSI is a typical multiphysics problem containing two coupled physical domains. This work also interests in structure- or fluid-related multiphysics problems with more than two physical domains. Two examples are given first.

For aero-elasticity analysis of wind turbines, usually the rotational velocity is fixed in FSI simulations e.g. in [62, 69]. As shown in [136] and [120], wind turbines with dynamical rotational velocity can be simulated through co-simulation with an additional generator solver. The framework of the three solvers coupled co-simulation is shown in Fig. 2.1. The fluid solver $\mathbf{f} = F(\mathbf{s})$ takes the displacements of the blade surfaces $\mathbf{s}$ as input, and outputs the wind load on the blades $\mathbf{f}$. The structure solver $\mathbf{s} = S(\mathbf{f}, g)$ takes $\mathbf{f}$ and the resistance torque g due to power generation as input, and outputs $\mathbf{s}$. The generator model extracts the rotational velocity out of $\mathbf{s}$ to compute g. Note that g is a scalar computed from an ordinary differential equation. The coupled problem can be solved in a general way by nested loops, i.e. two partitions are coupled together as a combined partition which is converged in an inner loop, and the combined partition is coupled to the remaining partition in an outer loop. For example, the structure and the generator solvers can be coupled in the inner loop while the structure and the fluid are coupled in the outer loop. However, this problem has its specialty that the fluid partition and the generator unit are independent, so it is defined as a bi-coupled problem in [128]. The best solution of a bi-coupled problem is to group the independent partitions together, then the need of an inner loop is removed due to the independence. Note that in both cases there is an outermost time step loop.

Another example is the optimization of a fluid-structure interacted problem. Besides a fluid and a structure solver, the co-simulation contains an optimizer where the optimization algorithm is implemented,

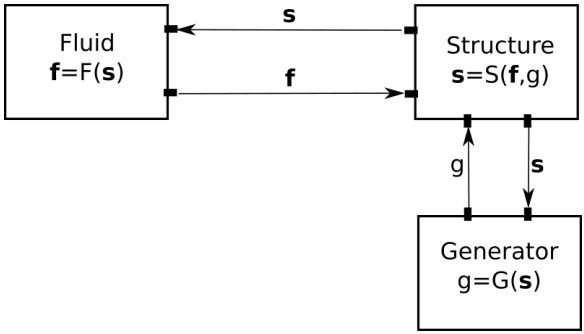


Figure 2.1: Co-simulation of a wind turbine with dynamic velocity. Three solvers are coupled in the co-simulation including a fluid, a structure and a generator solver.

as shown in Fig. 2.2. A constraint optimization problem is considered here, which can be formulated as $min\ p = P(\mathbf{q})$, with equality constraints $P_e(\mathbf{q}) = 0$ and inequality constraints $P_i(\mathbf{q}) < 0$ [102]. Here, $p = P(\mathbf{q})$ represents the fluid-structure coupled system, where $\mathbf{q}$ denotes the vector of design variables and p the scalar of objective, which can be computed out of the fluid or structure fields. The optimizer outputs $\mathbf{q}$ to the fluid or the structure solver or both, and takes p and the gradient ∇p as inputs. The gradient helps to decide the direction of optimal searching, and can be approximated with finite difference approach or be computed through sensitivity analysis [49, 66, 37]. The problem is solved in nested loops where the optimization iteration is the outermost loop.

In [47], structure related mulitiphysics problems appearing in practice are reviewed, including

- fluid-structure-interaction,
- thermal-structure-interaction,
- control-structure-interaction,
- control-fluid-structure-interaction,
- fluid-structure-combustion-thermal-interaction,
- fluid-structure-optimization.

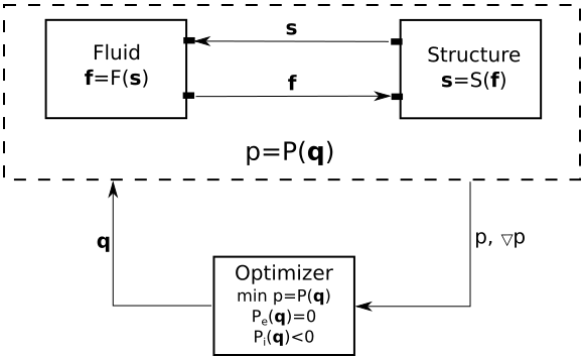


Figure 2.2: Co-simulation of fluid-structure-optimization. Three solvers are coupled in the co-simulation including a fluid solver, a structure solver and an optimizer.

FSI simulations where structures are modeled as rigid bodies (multi-body systems) can be found in e.g. [103].

Partitioned analysis of a general multiphysics problem is summarized as follows:

- A coupled problem is governed by the equations of the individual physical domains and the coupling conditions between them.
- Each partition can be rewritten by a black-box solver with inputs and outputs, which are the variables appearing in the coupling conditions. In this way, the global system is reformulated by the black-box solvers and the communications (equations) between inputs and outputs.
- A global solution scheme specifies in which sequence the solvers and the communications are performed. The fixed point iteration in FSI is one example. If there are more than two solvers, the problem can be solved in nested loops. An alternative is to use the interface Jacobian-based co-simulation algorithm (IJCSA) proposed in [120]. This algorithm requires Jacobian matrices regarding the interface variables form the individual domains to formulate the global Newton-Raphson equation, but the Jacobian matrices are not always available from black box solvers. No matter which algorithm is used, more physical domains result in lower efficiency. Therefore, possibilities of simplification should be considered as

much as possible, such as elimination of unimportant partitions or coupling conditions, using loose coupling or even one-way communication to replace iterative coupling.

2.5 Summary

This chapter aims at presenting the theoretical background of the co-simulation software environment developed in this work. The content is summarized below.

The governing equations of structure and fluid are derived as well as the boundary and initial conditions. Methods of space and time discretizaton are also presented so that the numerical models of both problems are established.

Regarding FSI, the coupling conditions between the fluid and the structure domains are defined at the interface. The governing equations of the individual domains together with the coupling conditions constitute the mathematical description of FSI. Due to separate discretizations, a mapping technique is required for applying the coupling conditions. The partitioned strategy enables to solve the coupled equations in a co-simulation where each partition becomes a black box solver. Different coupling algorithms for solving the coupled problem are reviewed which can be categorized in different ways: fixed point iteration or Newton-Raphson iteration; block Gauss-Seidel or block Jacobi; loose coupling or iterative coupling. Beside mapping and coupled equations solving, extrapolation (prediction) is also an important technique which helps to stabilize co-simulations.

A general co-simulation can also contain more than two partitions. Similar to partitioned FSI, each partition is rewritten into a black box solver with inputs and outputs, and the coupling conditions specify the relations between the inputs and outputs. More-partitions coupled problems can be either solved as nested two-partitions coupled problems or by the IJCSA which requires Jacobian matrices. But with more solvers and more coupling conditions, the solution process will become more lengthy. To reduce the complexity and improve the efficiency, it is suggested to remove unimportant partitions or connections, and use loose coupling or even one-way communication to replace iterative coupling.

Chapter 3

Co-Simulation Environment EMPIRE

Some commercial software codes are able to solve multiphysics problems within a single program, since they usually own the source codes of solving the individual physical domains. But this work aims at solving multiphysics problems in a co-simulation, where different simulation programs run in parallel with individually stored data and the programs exchange data in the runtime. This is a typical parallel process of multiple program multiple data (MPMD).

There exist several software projects that provide a co-simulation environment for partitioned FSI analysis, including commercial codes such as MPCCI [76] and in-house research codes such as COMA [55, 56] and preCICE [57, 58]. An overview and comparison of the existing works is given in [57]. From these projects the following trends can be found:

- The message passing interface (MPI) is favored for data communication across programs due to its high-speed communication implemented by hardware and high-level interface functions.

- The generality is more and more considered in the software development, i.e. an environment is more compatible to new algorithms and new solvers. Besides, the same co-simulation environment can be used in general coupled problems with two partitions instead of only in FSI.

- There are mainly two ideas for software interfacing, which are called the *library* and the *framework* approaches in [57]. With

the library approach, a software library of coupling functionalities with application program interface (API) is provided. The developer of a solver needs to write an adapter by inserting functions of the API at certain places in the old code to participate in a co-simulation. With the framework approach, an interface is defined which is to be implemented by the developer of a solver. And the adapter is already written and provided which calls only the interface functions. The benefit of this approach is that the synchronization among programs can be guaranteed. One example of the framework approach is the functional mock-up interface (FMI) [52], which is implemented by several commercial codes so that they can be easily adapted for different uses. Shortly spoken, one approach provides an interface of the coupling software and the other approach provides the interface of the software being coupled. But the two approaches can be used in combination since they are independent.

- Client-server model is widely used, where the solvers are the clients and a new coupling supervisor is the server. The coupling functionalities including mapping, extrapolation, coupled equations solving are implemented in the server, which is an additional program that runs in parallel with the clients. The clients cannot communicate directly with each other but only via the server. An alternative is the peer to peer (P2P) model, where the solvers directly communicate. And the coupling functionalities are implemented in a library which is integrated into the individual solvers instead of in an additional server program. This model is more efficient in the sense that the communication via the server is avoided.

Although some of the works in literature provide big generality for co-simulation with two solvers, they have not considered the case with more than two solvers or they can only work for customized cases with fixed numbers of solvers and fixed scenarios of co-simulation.

In this work, a co-simulation environment EMPIRE (Enhanced MultiPhysics Interface Research Engine) is developed to solve general multiphysics problems. This chapter starts with the requirements of EMPIRE, which are followed by the design concepts and the implementation. The configuration of co-simulations and the run-time behavior are presented at the end.

3.1 Requirements

In software engineering, requirements elicitation is the first step in the development circle which defines the purpose of the software [22]. Important requirements of EMPIRE are listed below:

1. Flexibility: it should be flexible to allow a general co-simulation scenario, i.e. the number of solvers to be coupled and the sequence of communications among them are flexible. This serves as the main novelty of the new co-simulation environment.

2. Usability: it should provide necessary coupling functionalities including mapping, extrapolation and coupled equations solving (coupling algorithms); besides, the effort of adapting a solver code in order to participate in a co-simulation should be small.

3. Extendability: it should be extendable for new functionalities.

3.2 Design Concepts

The design concepts that solve the requirements are presented here.

3.2.1 Client-Server Model

The client-server model is chosen which allows an arbitrary number of clients, as shown in Fig. 3.1. The clients are the solvers of individual partitions and the server is an additional coupling program released by EMPIRE with the name Emperor. As mentioned at the beginning of this chapter, the clients communicate with each other indirectly through the server. As a result, one solver can be replaced by an alternative that solves the same physical problem without affecting the others. The client-sever model is chosen instead of the P2P model, because the coupling functionalities are "naturally" centralized into the server code. In case of P2P model, these functionalities must be distributed among the clients, and it is not obvious that which functionalities belong to which clients. What's more, high-speed hardware communication on modern computers makes the time spent on communication negligible, which also facilitates the client-server model.

The library approach is used, i.e. a library EMPIRE_API is provided to the clients. The interface of the library defines the types of data to be communicated and provides the functions of communicating (sending or receiving) these data. The predefined data types include mesh, field

and signal. Field is the type of data defined on a mesh and signal represents a number or an array of numbers. A solver needs to be adapted by inserting the API functions at certain places e.g. receiving data before solving the domain and sending data after that. An adapter needs to be compiled with the library EMPIRE_API. The library is internally written in C++ but offers an interface (header file) in C, so that solvers written in languages compatible with C, e.g. C, C++, Fortran, Python, Java, MATLAB and so on can be compiled with it. The framework approach is not applied, since it is impossible to define all possible behaviors of different solvers in different co-simulations by interface functions. However, if the type of a co-simulation is fixed, the framework approach can be used in combination with the library approach. But this work does not go further in this direction.

The server-client model is realized via MPI-2.2, where functions of the connection between different programs and functions of communication on raw data types such as int and double are provided.

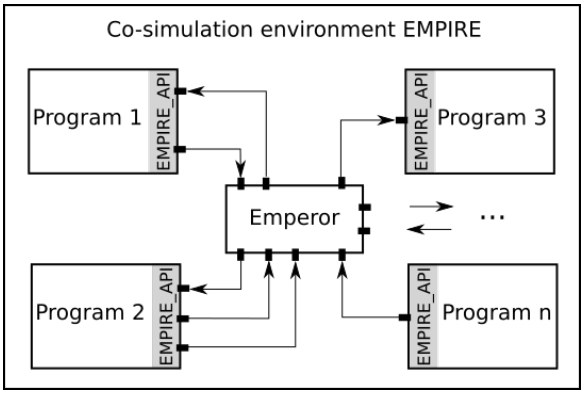


Figure 3.1: Co-simulation within EMPIRE.

3.2.2 Flexible Co-Simulation Scenario

The scenario of a co-simulation can contain all activities in the process of solving the coupled problem. However, the concept *Co-simulation scenario* defined here concentrates on the communication process. An artificial coupling scenario is given as an example in Fig. 3.3, where all *connections* (communications) between the server and the clients are defined both statically and dynamically. The index of a connection

indicates its order in a sequence of connections. Note that connections 1a and 1b actually belong to the same connection, where the data flow from Program 1 to Program 2 through Emperor. The same applies to connections 2a and 2b. The statical diagram specifies the sender and receiver of each connection, and the dynamical diagram specifies the order of communications. With both diagrams, the co-simulation scenario is clearly defined.

Two questions need to be considered:

- How to model a flexible co-simulation scenario?
- How to synchronize the server and the clients in a flexible co-simulation scenario?

From the example in Fig. 3.3 it can be seen that a co-simulation scenario can be modeled by loops and sequences of connections. Example of loops can be time step loop, iterative coupling loop and optimization loop. Inside a loop, there can be a sequence of communications or even inner loops. One example of sequence of connections is the displacement and the force exchanges during an FSI co-simulation.

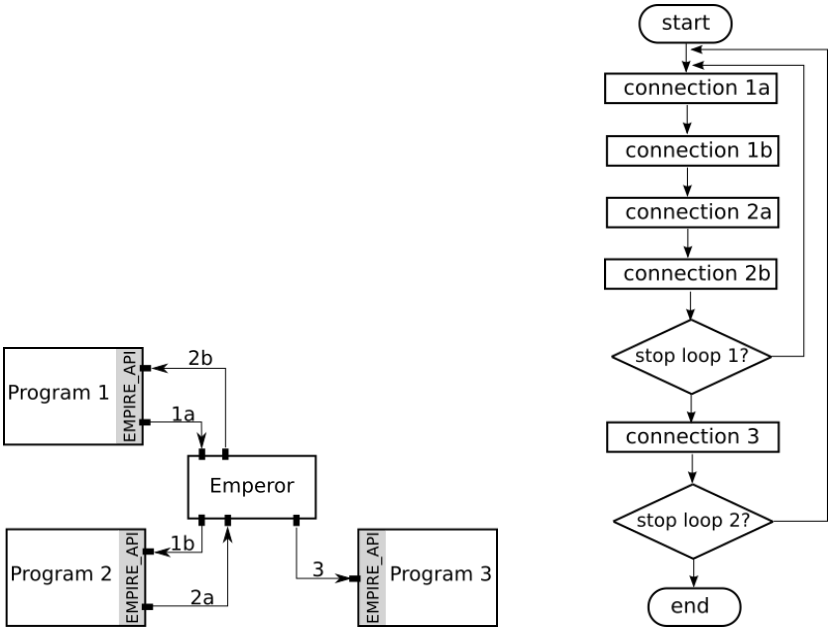
Given a defined co-simulation scenario, the order of communication calls in all programs can be derived. Fig. 3.4 shows the communication calls in the individual clients corresponding to the example in Fig. 3.3. It can be seen that the sending and receiving must be synchronized as well as the exiting-loop actions. The runtime behavior of the server could be automatically configured according to the co-simulation scenario. However, the runtime behavior of the clients cannot be controlled since they are written by other persons, so the synchronization among the server and the clients cannot be guaranteed automatically. To help avoid non-synchronization mistakes, pseudo codes are provided to the persons writing the clients. The pseudo codes can be automatically generated according to a given co-simulation scenario. They are just the text representation of what is shown in Fig. 3.4, which contain only communication calls.

The flexibility in defining co-simulation scenarios also enables a free choice between the block Gauss-Seidel process and the block Jacobi process, as shown in Fig. 3.2, where the index of a connection locates above the arrow and the notation of the communicated data locates below it. The indexes still indicate the orders of the connections that are executed in a sequence. Note that although connection 1 is called before connection 2 in Fig. 3.2b, the two solvers can be regarded as running in parallel since the connections take very little time.

(a) Block Gauss-Seidel.

(b) Block Jacobi.

Figure 3.2: Set up connections in block Gauss-Seidel and block Jacobi processes for FSI.



(a) (b)

Figure 3.3: An artificial co-simulation scenario where all connections are defined both statically and dynamically.(a) Statical diagram. (b) Dynamical diagram.

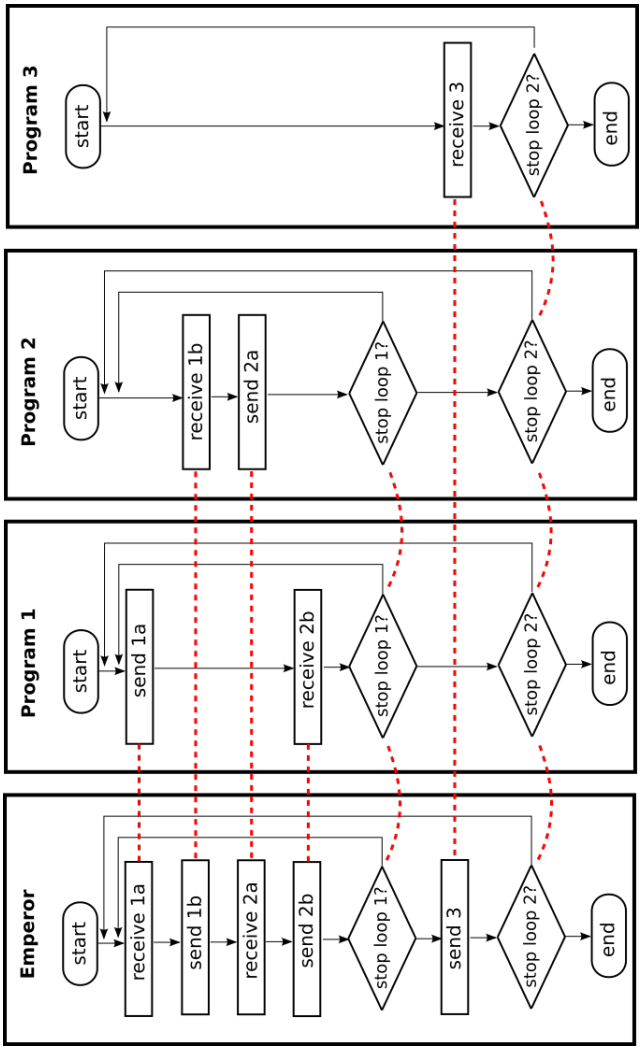


Figure 3.4: Communication flows in all programs derived from the co-simulation scenario. The sending and receiving must synchronize between programs as well as the exit-loop check. The red-dotted lines represent synchronizations.

3.2.3 Flexible Data Operation

The data structure of a connection in Emperor allows multiple inputs and multiple outputs to provide biggest flexibility. Besides, data operations inside a connection are modeled as *filters*, which can also have multiple inputs and outputs, as shown in Fig. 3.5. Examples of filter include the mapping operation and some arithmetic operations such as addition of two fields.

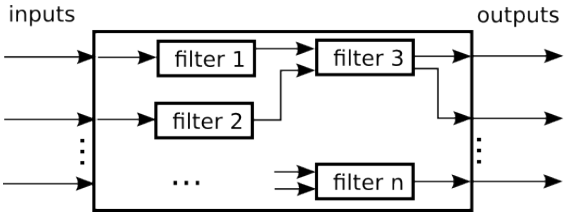


Figure 3.5: A connection with multiple inputs and multiple outputs and arbitrary filters.

3.3 Implementation

The software EMPIRE has two parts of codes namely the server Emperor and the library EMPIRE_API. The latter is straightforward to implement and is much smaller than the server code, so the implementation of it is omitted here. All the coupling functionalities are implemented in Emperor in the way of object oriented programming. The design of the classes in Emperor is presented here with class diagrams in order to offer a general picture of the whole program. The class names are written in the serif font.

3.3.1 Data Storage

The interface data of all clients are also stored in Emperor. There exist three types of data including mesh, field and signal. They are implemented by the classes AbstractMesh, DataField and Signal respectively. The sub-classes FEMesh and IGAMesh are derived from AbstractMesh, which implement the finite element mesh and the isogeometric mesh [70, 10] respectively. These data types correspond to those in EMPIRE_API because data are communicated between Emperor and EMPIRE_API. Objects of DataField are contained in objects

of AbstractMesh, since fields are defined on meshes. Interface data of each client are stored in a container which is represented by the class ClientCode. Communication functions are called within ClientCode to communicate with external clients. The class diagram of these classes is shown in Fig. 3.6.

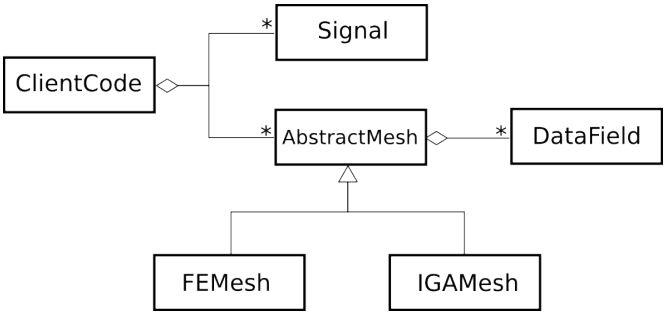


Figure 3.6: Data storage in Emperor (UML Class diagram).

3.3.2 Co-Simulation Scenario

The sequences and loops during coupling are implemented by the class AbstractCouplingLogic. It is written following the *composite pattern* i.e. an object of AbstractCouplingLogic can contain a list of objects of the same type, as shown in Fig. 3.7. When one object of AbstractCouplingLogic is executed, the sub-objects that it contains are executed in sequence iteratively.

The sub-classes of AbstractCouplingLogic are listed below:

- Connection. It is the basic unit of coupling which does not contain any sub-objects of AbstractCouplingLogic to execute. It implements a communication/connection which can have multiple inputs and outputs, as shown in Fig. 3.5. An input defines that the data flow from a client to the server, and an output defines that the data flow from the server to a client. Between receiving inputs and sending outputs, a sequence of data operations implemented by the class AbstractFilter can be performed.

- IterativeCouplingLoop. It implements the iterative coupling for solving coupled equations. Take iteratively coupling for FSI as an example, two objects of Connection which are communications

of displacements and forces can be added to an object of IterativeCouplingLoop. The connections are executed iteratively until convergence. Class AbstractCouplingAlgorithm implements the coupling algorithms used in iterative coupling, e.g. Aitken relaxation. The residual is implemented by the class Residual. When the L2 norm of the residual is below a user defined limit, the server will inform the clients to exit the loop. AbstractCouplingAlgorithm needs Residual because most coupling algorithms use residuals to update the values for the next iteration.

- TimeStepLoop. It implements the time step loop. Unlike IterativeCouplingLoop, the number of iterations to be performed is fixed by the number of time steps. Transient FSI solved by iterative coupling can be modeled by an object of TimeStepLoop wrapping an object of IterativeCouplingLoop. The class AbstractExtrapolator implements extrapolation algorithms and is called by TimeStepLoop.

- OptimizationLoop. It implements the loop of an optimization process. An object of ClientCode which is the optimizer will notify the server whether the optimal is found or not. Then this will be further broadcast from the server to the other clients.

- CouplingLogicSequence. The number of iterations is set to one. One instance of this class will be initialized by default in each co-simulation, which is called coSimulation. It is the root where all objects of AbstractCouplingLogic are added in a recursive way. In the runtime, the server executes coSimulation to start a co-simulation.

3.3.3 Coupling Functionalities

The basic coupling functionalities implemented in Emperor are mapping, extrapolation, coupled equation solving (coupling algorithms). Besides, the code provides an interface for data operations through the filter concept so that new operations can be easily added.

Mapping

The base class of mapping is AbstractMapper, which requires two objects of AbstractMesh to initialize the mapping operator. During mapping, it operates on objects of DataField which are defined in the objects of AbstractMesh, so that one field is computed according to the

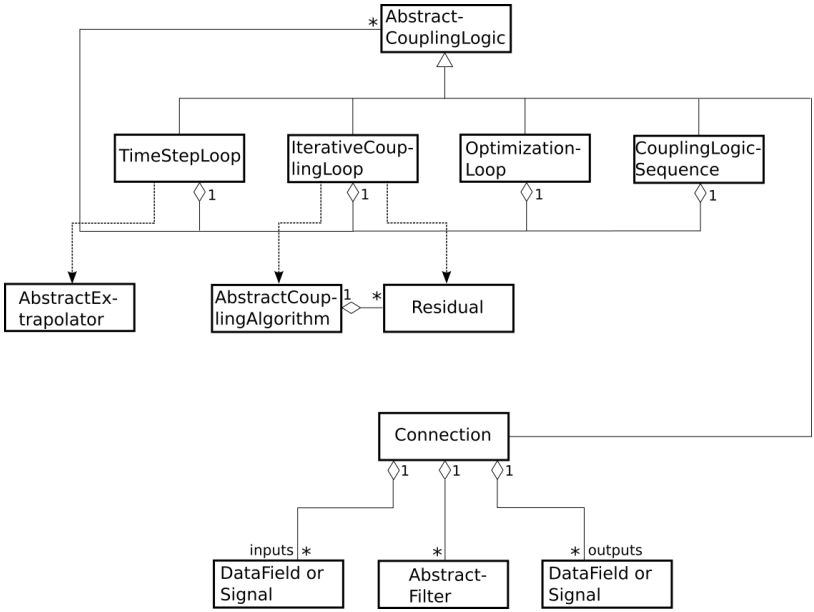


Figure 3.7: UML Class diagram of AbstractCouplingLogic.

other. The mapping algorithms are implemented in the derived classes of AbstractMapper, as shown in Fig. 3.8. The classes are listed in the following:

- NearestNeighborMapper. The nearest neighbor interpolation is implemented. It has low accuracy since the data on a point is directly assigned from that on its nearest neighbor. But it is the default choice for the case of matching interface grids.

- NearestElementMapper. The nearest element interpolation is implemented where the data is computed through projection on the nearest element and interpolation within it. It is more accurate and usually used in a consistent way.

- BarycentricInterpolationMapper. The barycentric interpolation is implemented which is very similar to the nearest element interpolation. The nearest element is replaced by the triangle constructed by the nearest three nodes.

- MortarMapper. The standard and the dual mortar algorithms are implemented in this class which are usually used in a conservative way.
- CurveSurfaceMapper. The linearized and co-rotating algorithms for mapping with structural beam elements are implemented. The details are presented in Chapter 5.
- IGAMortarMapper. The algorithm for mapping between an isogeometrical structural surface mesh and a FEM fluid surface mesh is implemented. The development and implementation of the algorithm is not done by the author, but it is the work of [3]. The algorithm is applied in the optimization example in Chapter 6.

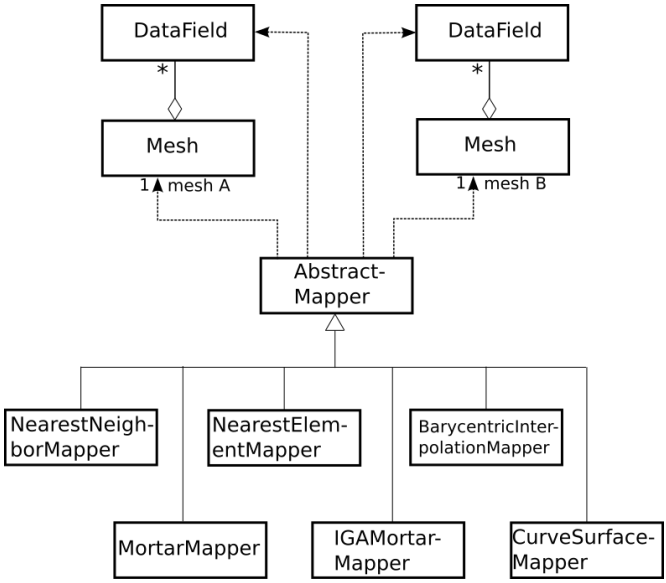


Figure 3.8: UML class diagram of AbstractMapper.

Extrapolation

The base class of extrapolation is AbstractExtrapolator. One derived class of it is LinearExtrapolator which implements the linear extrapolation. But new extrapolation algorithms can be easily added by extending AbstractExtrapolator.

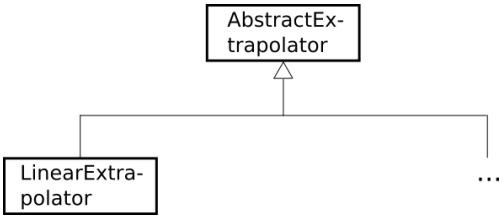


Figure 3.9: UML class diagram of AbstractExtrapolator.

Coupling algorithms

The base class of coupling algorithms for solving coupled equations is AbstractCouplingAlgorithm. The derived classes ConstantRelaxation and Aitken implement the constant and Aitken relaxation algorithms respectively. Other derived classes including GMRES which implements the Newton-Krylov algorithm and IJCSA which implements the IJCSA algorithm are not part of this work. New coupling algorithms can be easily added by extending AbstractCouplingAlgorithm.

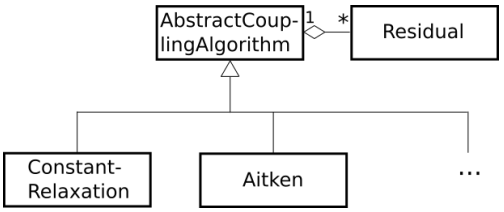


Figure 3.10: UML class diagram of AbstractCouplingAlgorithm.

Filters

The base class of filters is AbstractFilter. One derived class is MappingFilter, which calls an object of AbstractMapper to perform mapping during a connection. Filters doing arithmetic operations or realizing other purposes are skipped. New filters can be easily added by extending AbstractFilter.

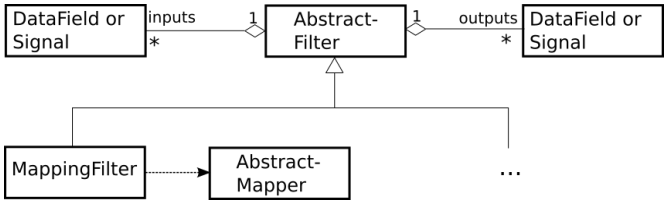


Figure 3.11: UML class diagram of AbstractFilter.

3.4 Configuration and Runtime Behavior

The server is configured by an XML input file, which defines the instances of the classes introduced in 3.3. In the input file, an instance of a class is denoted by the class name with the first letter of lowercase, e.g. an instance of ClientCode is a clientCode. Moreover, the word "abstract" is removed from the name of an instance of a base class. Such an instance has an attribute *type* which specifies the actual type. For example, an instance of AbstractMapper is a mapper, and its attribute *type* can be assigned as MortarMapper. The input file has six important blocks that are listed below:

1. Definition of clientCodes. An arbitrary number of clientCodes is allowed. Each clientCode can contain multiple meshes and signals and each mesh can contain multiple dataFields. In the runtime, the server waits until all clients defined in the input file are connected; each client sends meshes (if defined) to the sever; and the sever allocates memory for the meshes, dataFields and signals. Since the interface data are defined here, the remaining blocks in the input file will refer to these data.

2. Definition of mappers. Each mapper refers to two meshes defined previously to initialize the mapping operators.

3. Definition of couplingAlgorithms. In each couplingAlgorithm, residuals are defined.

4. Definition of extrapolators.

5. Definition of connections. Inputs and outputs and a list of filters that will be executed in sequence are defined in each connection. If necessary, a mappingFilter can be defined which calls a mapper defined previously.

6. Definition of couplingLogics. As mentioned before, the instance coSimulation exists in each input file by default. It is the root couplingLogic under which all sub-couplingLogics are defined recursively.

During the runtime, the server and the clients cooperate in the following sequence:

1. The server opens a port and the clients are connected to the server through it.

2. The server initializes clientCodes and receives meshes from the clients if defined. Then the storage of all interface data is allocated in the server.

3. The server initializes mappers, couplingAlgorithms, extrapolators, connections and couplingLogics.

4. The server executes the instance coSimulation then the co-simulation is started. The server runs through the recursively defined couplingLogics, and the clients should send and receive data in synchronization with the server. The co-simulation is finished after coSimulation is finished.

5. All clients disconnect from the server. All programs are exited.

The co-simulation scenario and the functionalities are completely defined by the XML input file. And the runtime behavior of the server is automatically configured. The runtime behavior of the clients should also follow the defined co-simulation scenario, but this is not assured automatically. Mistakes can happen in manually written codes and non-synchronization is caused. As mentioned before, the server is able to output pseudo codes for the clients as a help to check mistakes in the communication calls.

3.5 Summary

The co-simulation environment EMPIRE provides a software solution for general multiphysical co-simulations. During the design and the implementation three requirements are concerned namely flexibility, usability and extendability.

The ingredients of co-simulation include black box solvers with interface data (inputs and outputs), data exchange and data operations

during data exchange. Client-server model is applied in EMPIRE. The clients exchange data with each other through an additional server program, which is also in charge of data operations such as mapping, prediction or relaxation. A library named EMPIRE_API is provided to the clients which defines the types of interface data including mesh, field and signal. It also provides the send and receive functions of these data. To participate in a co-simulation, each solver can be adapted easily by inserting these functions at proper places in the existing control sequence.

The biggest novelty of EMPIRE is that it allows the user to define a co-simulation scenario flexibly, which consists of loops and sequences of connections. Given a defined co-simulation scenario, the runtime behavior of the server and the clients regarding communication can be derived. And in practice, it is used to configure the runtime behavior of the server program and write pseudo codes for the clients. Flexible data operations during connections are realized by the *filter* concept. Moreover, regular coupling functionalities such as mapping, extrapolation and coupled equations solving are provided.

The current implementation of EMPIRE already supports various types of co-simulations. Moreover, the concepts and the data structures allow the software to be extended for future needs, i.e. new classes can be added to the existing code structure and new functionalities can be added by extending the abstract classes.

Chapter 4

Mapping with Surface Meshes

Mapping is needed in general surface coupled multiphysics problems, but it is discussed here in the context of FSI. As introduced in Chapter 2, a numerical FSI model consists of three main parts namely the fluid equations, the structure equations and the coupling conditions at the fluid-structure interface which define the equivalence of the interface displacement and traction. The fluid and structure meshes at the fluid-structure interface are usually non-matching due to the need of separate discretization with resolutions which are adapted to the requirements of the different physics (see Fig. 4.1). Therefore, a mapping technique is required to apply the interface boundary conditions between non-matching interface meshes.

Different widely used mapping algorithms for surface meshes are introduced in [32, 46] which can be differentiated into two main types. Algorithms of the first kind use interpolation. According to the nature of interpolation the sum of the coefficients is usually equal to 1, so the consistency of mapping is guaranteed, i.e., a constant data field can be exactly recovered. The accuracy of mapping is dependent on the polynomial order and the number of data points used in an interpolation model. Nearest neighbor interpolation is the simplest example where only a single neighboring node is used. In barycentric interpolation, three neighboring nodes are searched to build up a barycentric coordinate system (i.e. a linear triangular element) where the shape functions are used for interpolation. It is similar to nearest element interpolation where the nearest element is searched. RBF interpolation [12, 112, 122] can apply an arbitrary number of neighboring nodes to form a higher order radial basis function model. Within the second type of map-

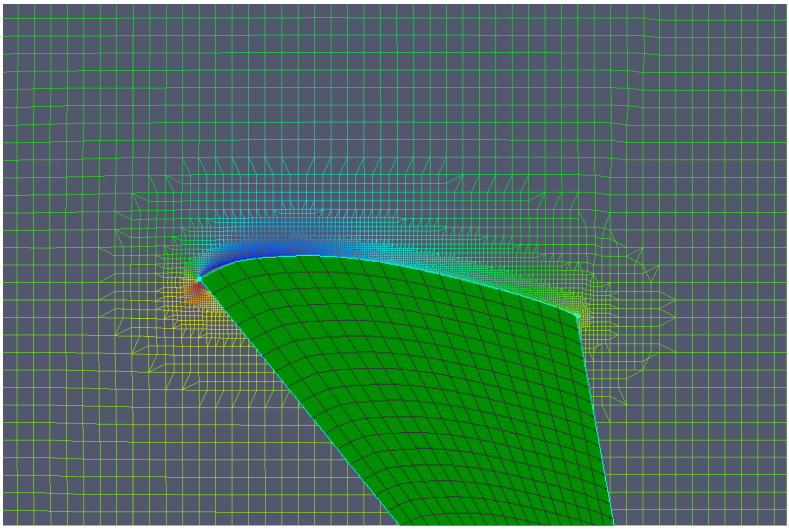


Figure 4.1: Non-matching meshes at the fluid-structure interface in an FSI simulation of an airfoil. (Only a vertical slice of the fluid mesh is shown.)

ping algorithms, a weak form of the interface boundary conditions is constructed and solved. The standard mortar method [15] applies the Galerkin approach which can minimize the L2 norm of the deviation between two fields [21]. A dual mortar method is introduced in [139] which diagonalizes the coefficient matrix (which has the form of a mass matrix) to reduce the computational cost. The standard and the dual mortar methods are not only used in contact mechanics [113, 114] but also in FSI [46, 73, 129, 79].

Consistency is the basic criterion for mapping algorithms which specifies that a constant field should be mapped exactly. Another criterion is the conservation of energy which helps to derive a special mapping operator for traction/force as presented in [42]. In [32], the direct use of the mapping algorithms is called *consistent mapping* while using a mapping operator derived from the energy conservation is called *conservative mapping*. The consistent mapping is renamed as *direct mapping* in our work to avoid ambiguity in the discussion of consistency. The mortar algorithms are usually used in a conservative way e.g. in [42] for partitioned FSI and in [46, 129, 79] for monolithic FSI. But the conservative mapping in combination with interpolation based methods can

result in unphysical traction fields as shown in [32]. In this chapter, the direct and conservative mapping of nearest element interpolation and the mortar algorithms are assessed and compared because these algorithms have similarity in formulation and popularity in practice as well.

Mortar algorithms are not consistent at curved edges of the wet surface due to the discrepancy between the integration domains. To tackle this problem, state of the art works [129, 29, 111] evaluate the integrations on the common contact area and the latter two papers use a scaling approach to improve the condition of the resulting matrix. The scaling approach does not work if a fluid element is totally not overlapped by structure elements. This can happen in the case of high mesh density ratios, which is a typical scenario for FSI with highly turbulent flow simulation on very fine meshes. To solve the problem, a robust enforcing consistency approach is developed in this work which embeds inter-/extrapolation into the mortar matrices.

This chapter starts with the interface coupling conditions and the discretization in Section 4.1. Criteria of consistency and energy conservation are introduced in Section 4.2. Section 4.3 introduces the nearest element interpolation, the standard and the dual mortar methods comparatively with the consideration of consistency and energy conservation. Moreover, the newly developed enforcing consistency algorithm is presented. The convergence behavior of different algorithms is tested in Section 4.4. In Section 4.5, special examples including a wind turbine blade are tested to demonstrate the inconsistency of the mortar methods and the oscillating problems happening when using conservative mapping. Finally, this chapter is concluded in Section 4.6.

4.1 Interface Coupling Conditions and Discretization

The kinematic and dynamic coupling conditions at the continuous interface Γ corresponding to the initial configuration are:

$$u_{\mathrm{f}}(\mathbf{x}) = u_{\mathrm{s}}(\mathbf{x}) \qquad (\mathbf{x} \in \Gamma), \tag{4.1a}$$

$$p_{\mathrm{s}}(\mathbf{x}) = p_{\mathrm{f}}(\mathbf{x}) \qquad (\mathbf{x} \in \Gamma), \tag{4.1b}$$

where $u_{\mathrm{f}}(\mathbf{x})$ and $u_{\mathrm{s}}(\mathbf{x})$ are the displacement fields while $p_{\mathrm{f}}(\mathbf{x})$ and $p_{\mathrm{s}}(\mathbf{x})$ are the traction fields at the wet surface from the fluid and the structure respectively. The following points have to be clarified:

- The notations denote scalar fields which represent the x-, y- or zth component of the corresponding vector fields, respectively. Since the vector fields are treated componentwise in the same way in mapping, i.e. the same mapping operator is used for each component, scalar fields are used in the derivation of mapping operators instead of vector fields for the purpose of conciseness.
- Γ corresponds to the initial configuration. Usually the traction from the fluid is computed from the Cauchy stress which is defined on the current configuration, in this case transformation is needed.

The conservation of the interface energy is expressed as

$$\int_{\Gamma} u_{\mathrm{f}}(\mathbf{x}) p_{\mathrm{f}}(\mathbf{x}) \,\mathrm{d}\Gamma = \int_{\Gamma} u_{\mathrm{s}}(\mathbf{x}) p_{\mathrm{s}}(\mathbf{x}) \,\mathrm{d}\Gamma. \tag{4.2}$$

The equation is still expressed by the scalar fields and it enforces that each individual part of the energy computed from a component (x-, y- or zth) is equal rather than only the sum is equal. The equation is satisfied obviously if (4.1) is satisfied.

The continuous interface Γ is discretized separately into Γ_{f} from the fluid side and Γ_{s} from the structure side. In the following, Γ_{f} and Γ_{s} will be called the *fluid mesh* and the *structure mesh* respectively. Note that they are interface meshes used in mapping. The continuous fields $u_{\mathrm{f}}(\mathbf{x})$, $p_{\mathrm{f}}(\mathbf{x})$, $u_{\mathrm{s}}(\mathbf{x})$ and $p_{\mathrm{s}}(\mathbf{x})$ are discretized into $u_{\mathrm{f}}^h(\mathbf{x})$, $p_{\mathrm{f}}^h(\mathbf{x})$, $u_{\mathrm{s}}^h(\mathbf{x})$ and $p_{\mathrm{s}}^h(\mathbf{x})$ as

$$u_{\mathrm{f}}(\mathbf{x}) \approx u_{\mathrm{f}}^h(\mathbf{x}) = \mathbf{N}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_{\mathrm{f}} \qquad (\mathbf{x} \in \Gamma_{\mathrm{f}}), \tag{4.3a}$$

$$p_{\mathrm{f}}(\mathbf{x}) \approx p_{\mathrm{f}}^h(\mathbf{x}) = \mathbf{N}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\mathbf{P}_{\mathrm{f}} \qquad (\mathbf{x} \in \Gamma_{\mathrm{f}}), \tag{4.3b}$$

$$u_{\mathrm{s}}(\mathbf{x}) \approx u_{\mathrm{s}}^h(\mathbf{x}) = \mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\mathbf{U}_{\mathrm{s}} \qquad (\mathbf{x} \in \Gamma_{\mathrm{s}}), \tag{4.3c}$$

$$p_{\mathrm{s}}(\mathbf{x}) \approx p_{\mathrm{s}}^h(\mathbf{x}) = \mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\mathbf{P}_{\mathrm{s}} \qquad (\mathbf{x} \in \Gamma_{\mathrm{s}}), \tag{4.3d}$$

where $\mathbf{N}_{\mathrm{f}}(\mathbf{x})$, $\mathbf{U}_{\mathrm{f}}$, $\mathbf{P}_{\mathrm{f}}$ are respectively the shape functions, nodal displacements and nodal tractions of the fluid mesh, while $\mathbf{N}_{\mathrm{s}}(\mathbf{x})$, $\mathbf{U}_{\mathrm{s}}$, $\mathbf{P}_{\mathrm{s}}$ are the same terms of the structure mesh.

The fluid and the structure nodal forces $\mathbf{F}_{\mathrm{f}}$ and $\mathbf{F}_{\mathrm{s}}$ are defined by

$$\mathbf{F}_{\mathrm{f}} = \int_{\Gamma_{\mathrm{f}}} \mathbf{N}_{\mathrm{f}}(\mathbf{x}) p_{\mathrm{f}}^h(\mathbf{x}) \,\mathrm{d}\Gamma_{\mathrm{f}}, \tag{4.4a}$$

$$\mathbf{F}_{\mathrm{s}} = \int_{\Gamma_{\mathrm{s}}} \mathbf{N}_{\mathrm{s}}(\mathbf{x}) p_{\mathrm{s}}^h(\mathbf{x}) \,\mathrm{d}\Gamma_{\mathrm{s}}. \tag{4.4b}$$

They can be rewritten as

$$\mathbf{F}_\mathrm{f} = \int_{\Gamma_\mathrm{f}} \mathbf{N}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x})\mathbf{P}_\mathrm{f}\,\mathrm{d}\Gamma_\mathrm{f} = \mathbf{M}_\mathrm{ff}\mathbf{P}_\mathrm{f}, \tag{4.5a}$$

$$\mathbf{F}_\mathrm{s} = \int_{\Gamma_\mathrm{s}} \mathbf{N}_\mathrm{s}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\mathbf{P}_\mathrm{s}\,\mathrm{d}\Gamma_\mathrm{s} = \mathbf{M}_\mathrm{ss}\mathbf{P}_\mathrm{s}, \tag{4.5b}$$

where $\mathbf{M}_\mathrm{ff}$ and $\mathbf{M}_\mathrm{ss}$ are the mass matrices of the fluid and the structure mesh respectively. Inserting (4.3) into (4.2), the discrete form of the energy conservation at the interface can be obtained as

$$\mathbf{U}_\mathrm{f}^\mathrm{T}\mathbf{F}_\mathrm{f} = \mathbf{U}_\mathrm{s}^\mathrm{T}\mathbf{F}_\mathrm{s}. \tag{4.6}$$

4.2 Direct and Conservative Mapping

A mapping algorithm derives a discrete form of the boundary conditions in (4.1) which results in a mapping matrix. The direct displacement mapping matrix $\mathbf{H}_\mathrm{fs}$ defines the discrete form of (4.1a) as

$$\mathbf{U}_\mathrm{f} = \mathbf{H}_\mathrm{fs}\mathbf{U}_\mathrm{s}. \tag{4.7}$$

The direct traction mapping matrix $\mathbf{H}_\mathrm{sf}$ defines the discrete form of (4.1b) analogously as

$$\mathbf{P}_\mathrm{s} = \mathbf{H}_\mathrm{sf}\mathbf{P}_\mathrm{f}. \tag{4.8}$$

If all row-sums in a mapping matrix are equal to 1, the mapping is *consistent*, i.e. a constant field can be mapped exactly.

The *conservative* traction mapping matrix can be derived from the interface energy conservation equation and an existing displacement mapping matrix: given $\mathbf{H}_\mathrm{fs}$, it can be derived from (4.6) that

$$\mathbf{F}_\mathrm{s} = \mathbf{H}_\mathrm{fs}^\mathrm{T}\mathbf{F}_\mathrm{f}. \tag{4.9}$$

It can be rewritten with tractions as

$$\mathbf{P}_\mathrm{s} = \mathbf{M}_\mathrm{ss}^{-1}\mathbf{H}_\mathrm{fs}^\mathrm{T}\mathbf{M}_\mathrm{ff}\mathbf{P}_\mathrm{f}. \tag{4.10}$$

The conservative mapping saves the explicit computation of $\mathbf{H}_\mathrm{sf}$, since it is constructed by transposing the existing $\mathbf{H}_\mathrm{fs}$. If $\mathbf{H}_\mathrm{fs}$ is consistent, the conservative mapping also leads to the conservation of the resultant

force, since

$$\sum_{i=1}^{n_\mathrm{s}} \mathbf{F}_\mathrm{s}(i) = \sum_{i=1}^{n_\mathrm{s}} \sum_{j=1}^{n_\mathrm{f}} \mathbf{H}_\mathrm{fs}^\mathrm{T}(i,j)\mathbf{F}_\mathrm{f}(j)$$
$$= \sum_{j=1}^{n_\mathrm{f}} \mathbf{F}_\mathrm{f}(j) \sum_{i=1}^{n_\mathrm{s}} \mathbf{H}_\mathrm{fs}^\mathrm{T}(i,j) = \sum_{j=1}^{n_\mathrm{f}} \mathbf{F}_\mathrm{f}(j) \tag{4.11}$$

where n_f and n_s are the numbers of the nodes of the fluid and the structure mesh respectively.

The displacement is usually mapped directly in practice. Regarding the traction, it must be chosen between the direct mapping in (4.8) and the conservative mapping in (4.10). They can give different results since generally $\mathbf{H}_\mathrm{sf} \neq \mathbf{M}_\mathrm{ss}^{-1}\mathbf{H}_\mathrm{fs}^\mathrm{T}\mathbf{M}_\mathrm{ff}$. Both methods have pros and cons. On the one hand, conservative traction mapping guarantees the conservation of the resultant force and energy whereas it can lead to inconsistent traction fields. On the other hand, direct traction mapping is usually consistent but it results in difference in the resultant force and energy.

4.3 Mapping Algorithms

An interpolation algorithm constructs the mapping matrix by the weights of interpolation. Mortar algorithms solve the the weak form of (4.1a) as

$$\int_\Gamma \mathbf{N}_t(\mathbf{x})(u_\mathrm{f}(\mathbf{x}) - u_\mathrm{s}(\mathbf{x}))\,\mathrm{d}\Gamma = 0, \tag{4.12}$$

where $\mathbf{N}_t(\mathbf{x})$ is the vector of test functions. The discretized form is

$$\int_\Gamma \mathbf{N}_t(\mathbf{x})\mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{U}_\mathrm{f} = \int_\Gamma \mathbf{N}_t(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{U}_\mathrm{s}. \tag{4.13}$$

The difference of the mortar algorithms lies in different test functions.

The nearest element interpolation, the standard and the dual mortar methods are comparatively introduced and discussed in the following. All algorithms are implemented regarding linear triangular elements and bilinear quadrilateral elements. Elements of other types can be converted into the two types before mapping as an approximation.

4.3.1 Nearest Element Interpolation

In the displacement mapping with the nearest element interpolation, each fluid node $\mathbf{x}_{\mathrm{f}i}$ (for $i = 1, ..., n_f$) is projected to its nearest element

in the structure mesh and the unknown value is assigned as the result of interpolating the projection point $\mathbf{x}'_{\mathrm{f}i}$ inside the element (see Fig. 4.2a). The weights of interpolation are

$$w_l = \mathbf{N}_{\mathrm{s}}^{l}(\mathbf{x}'_{\mathrm{f}i}), \tag{4.14}$$

for each node with index l in the nearest element. w_l is assigned to the lth column in the ith row of $\mathbf{H}_{\mathrm{fs}}$. The row-sum is equal to 1 since the sum of the weights is 1, so the consistency is assured. One remark is that the projection lying outside of the nearest element as shown in Fig. 4.2b usually happens at the curved edges. In this case, the domain where the shape function is defined is expanded for extrapolation which results in some negative weights.

The process above can be alternatively formulated by setting the test functions in (4.13) as Dirac delta functions [46]:

$$\int_{\Gamma} \boldsymbol{\Delta}_{\mathrm{f}}(\mathbf{x})\mathbf{N}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{U}_{\mathrm{f}} = \int_{\Gamma} \boldsymbol{\Delta}_{\mathrm{f}}(\mathbf{x})\mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{U}_{\mathrm{s}}. \tag{4.15}$$

The Dirac delta functions are defined as

$$\boldsymbol{\Delta}_{\mathrm{f}i}(\mathbf{x}) = \begin{cases} \infty & \text{if } \mathbf{x} = \mathbf{x}_{\mathrm{f}i} \\ 0 & \text{if } \mathbf{x} \neq \mathbf{x}_{\mathrm{f}i} \end{cases}, \quad \text{with } i = 1, ..., n_f, \tag{4.16}$$

where $\mathbf{x}_{\mathrm{f}i}$ represents the ith fluid node. Since $\int_{\Gamma} \boldsymbol{\Delta}_{\mathrm{fi}}(\mathbf{x})\mathbf{N}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma = 1$, there is

$$\mathbf{H}_{\mathrm{fs}} = \int_{\Gamma} \boldsymbol{\Delta}_{\mathrm{f}}(\mathbf{x})\mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma. \tag{4.17}$$

To evaluate $\mathbf{H}_{\mathrm{fs}}$, the fluid node $\mathbf{x}_{\mathrm{f}i}$ is projected to the structure element to get the projection $\mathbf{x}'_{\mathrm{f}i}$. Then $\mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x}'_{\mathrm{f}i})$ are put into the ith row of $\mathbf{H}_{\mathrm{fs}}$. This is exactly the same process as the projection and interpolation inside the nearest element. The direct traction mapping operator $\mathbf{H}_{\mathrm{sf}}$ can be obtained in an analogous way.

The conservative traction mapping can be derived from (4.10) as

$$\int_{\Gamma} \mathbf{N}_{\mathrm{s}}(\mathbf{x})\mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{P}_{\mathrm{s}} = \int_{\Gamma} \mathbf{N}_{\mathrm{s}}(\mathbf{x})\boldsymbol{\Delta}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{F}_{\mathrm{f}}. \tag{4.18}$$

The equation can be interpreted in a way that a new weak form of the traction field is constructed where the structure shape functions become the test functions and the Dirac delta functions are used as interpolation functions of the discrete field $\mathbf{F}_{\mathrm{f}}$. The non-continuity of the Dirac delta functions explains the high oscillations in $\mathbf{P}_{\mathrm{s}}$ found by [32].

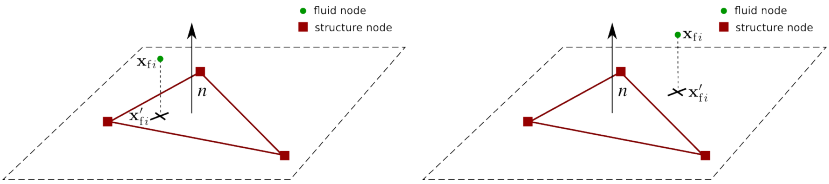
In Fig. 4.3 the algorithm is tested with an 1D example, where the data are mapped from the structure to the fluid mesh using direct mapping. Both meshes have equidistant nodes and linear shape functions. The resulting $\mathbf{H}_{\mathrm{fs}}$ is

$$\mathbf{H}_{\mathrm{fs}} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{5} & \frac{2}{5} & 0 \\ \frac{1}{5} & \frac{4}{5} & 0 \\ 0 & \frac{4}{5} & \frac{1}{5} \\ 0 & \frac{2}{5} & \frac{3}{5} \\ 0 & 0 & 1 \end{pmatrix}.$$

The corresponding conservative mapping matrix is

$$\mathbf{M}_{\mathrm{ss}}^{-1}\mathbf{H}_{\mathrm{fs}}^{\mathrm{T}}\mathbf{M}_{\mathrm{ff}} = \begin{pmatrix} \frac{29}{50} & \frac{17}{25} & \frac{1}{25} & -\frac{1}{5} & -\frac{1}{25} & \frac{3}{50} \\ -\frac{3}{25} & \frac{2}{25} & \frac{12}{25} & \frac{12}{25} & \frac{2}{25} & -\frac{3}{25} \\ \frac{3}{50} & -\frac{1}{25} & -\frac{1}{5} & \frac{1}{25} & \frac{17}{25} & \frac{29}{50} \end{pmatrix}.$$

Conservative mapping of a given constant traction field $\mathbf{P}_{\mathrm{f}}^{\mathrm{T}} = (1, 1, 1, 1, 1, 1)$ results in $\mathbf{P}_{\mathrm{s}}^{\mathrm{T}} = (\frac{28}{25}, \frac{22}{25}, \frac{28}{25})$, which demonstrates the inconsistency of the conservative mapping with nearest element interpolation.



(a) Inside. (b) Outside.

Figure 4.2: Projection to the nearest element in nearest element interpolation.

4.3.2 Standard Mortar Method

Standard mortar method sets $\mathbf{N}_t(\mathbf{x}) = \mathbf{N}_{\mathrm{f}}(\mathbf{x})$ in (4.13) as

$$\int_\Gamma \mathbf{N}_{\mathrm{f}}(\mathbf{x})\mathbf{N}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{U}_{\mathrm{f}} = \int_\Gamma \mathbf{N}_{\mathrm{f}}(\mathbf{x})\mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma\,\mathbf{U}_{\mathrm{s}}. \tag{4.19}$$

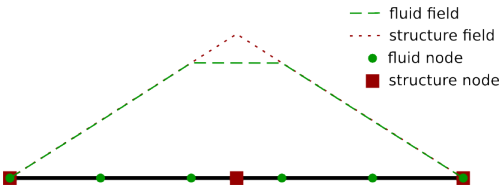


Figure 4.3: Using nearest element interpolation in an 1D test.

In this work, the integration domains corresponding to Γ are chosen as

$$\int_{\Gamma_\mathrm{f}} \mathbf{N}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x})\,\mathrm{d}\Gamma_\mathrm{f}\,\mathbf{U}_\mathrm{f} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}\,\mathbf{U}_\mathrm{s}, \tag{4.20}$$

where $\Gamma_{\mathrm{s}\to\mathrm{f}}$ is the overlap area between the fluid elements and the projection of the structure elements. The structure elements are projected on the fluid elements so that $\mathbf{N}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})$ can be evaluated. Rewrite (4.20) in matrix-vector form:

$$\mathbf{M}_{\mathrm{ff}}\mathbf{U}_\mathrm{f} = \mathbf{M}_{\mathrm{fs}}\mathbf{U}_\mathrm{s}, \tag{4.21}$$

where $\mathbf{M}_{\mathrm{ff}}$ is the same mass matrix of the fluid mesh as in (4.5a). With (4.7) the direct displacement mapping matrix is obtained:

$$\mathbf{H}_{\mathrm{fs}} = \mathbf{M}_{\mathrm{ff}}^{-1}\mathbf{M}_{\mathrm{fs}}. \tag{4.22}$$

The direct traction mapping matrix can be derived analogously as

$$\mathbf{H}_{\mathrm{sf}} = \mathbf{M}_{\mathrm{ss}}^{-1}\mathbf{M}_{\mathrm{sf}}. \tag{4.23}$$

However, the standard mortar algorithm is usually applied in a conservative way.

Inserting (4.22) into (4.10) the conservative traction mapping operator can be derived:

$$\mathbf{M}_{\mathrm{ss}}\mathbf{P}_\mathrm{s} = \mathbf{M}_{\mathrm{fs}}^\mathrm{T}\mathbf{M}_{\mathrm{ff}}^{-1}\mathbf{M}_{\mathrm{ff}}\mathbf{P}_\mathrm{f} = \mathbf{M}_{\mathrm{fs}}^\mathrm{T}\mathbf{P}_\mathrm{f}. \tag{4.24}$$

Comparing (4.23) and (4.24), the only difference between $\mathbf{M}_{\mathrm{sf}}$ and $\mathbf{M}_{\mathrm{fs}}^\mathrm{T}$ lies in the integration domains, which are $\Gamma_{\mathrm{f}\to\mathrm{s}}$ and $\Gamma_{\mathrm{s}\to\mathrm{f}}$ respectively. If $\Gamma_\mathrm{f} = \Gamma_\mathrm{s}$ (which means the integration domains are the same but not necessarily the meshes are matching), e.g. both meshes are generated from the same rectangle, the direct and the conservative traction mapping matrices are equal. If $\Gamma_\mathrm{f} \neq \Gamma_\mathrm{s}$ which is true for general geometries,

the matrices are not exactly but only approximately equal. The error comes from the facet error both inside the domain and at curved edges. The detailed proof of the inconsistency of the standard mortar method is put in A.1.

The same 1D example is also tested for this algorithm as shown in Fig. 4.4, where $\mathbf{H}_{\mathrm{fs}}$ is

$$\mathbf{H}_{\mathrm{fs}} = \begin{pmatrix} \frac{219}{220} & \frac{1}{110} & -\frac{1}{220} \\ \frac{67}{110} & \frac{21}{55} & \frac{1}{110} \\ \frac{37}{220} & \frac{19}{22} & -\frac{7}{220} \\ -\frac{7}{220} & \frac{19}{22} & \frac{37}{220} \\ \frac{1}{110} & \frac{21}{55} & \frac{67}{110} \\ -\frac{1}{220} & \frac{1}{110} & \frac{219}{220} \end{pmatrix}.$$

It is interesting to compare Fig. 4.4 with Fig. 4.3: although the standard mortar algorithm cannot interpolate the values exactly, it is able to minimize the deviation between the two dotted curves using the Galerkin approach. The corresponding conservative mapping matrix is

$$\mathbf{M}_{\mathrm{ss}}^{-1}\mathbf{H}_{\mathrm{fs}}^{\mathrm{T}}\mathbf{M}_{\mathrm{ff}} = \begin{pmatrix} \frac{29}{50} & \frac{17}{25} & -\frac{1}{50} & -\frac{13}{50} & -\frac{1}{25} & \frac{3}{50} \\ -\frac{3}{25} & \frac{2}{25} & \frac{27}{50} & \frac{27}{50} & \frac{2}{25} & -\frac{3}{25} \\ \frac{3}{50} & -\frac{1}{25} & -\frac{13}{25} & -\frac{1}{50} & \frac{17}{25} & \frac{29}{50} \end{pmatrix}.$$

Conservative mapping of a given constant traction field $\mathbf{P}_{\mathrm{f}}^{\mathrm{T}} = (1, 1, 1, 1, 1, 1)$ results in $\mathbf{P}_{\mathrm{s}}^{\mathrm{T}} = (1, 1, 1)$, so the conservative mapping with the standard mortar algorithm is consistent for this special test (since the domain is flat and the boundaries are matching exactly).

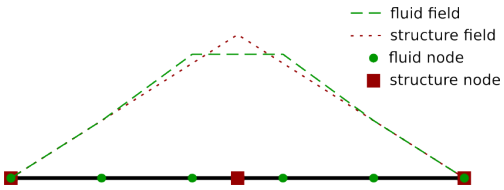
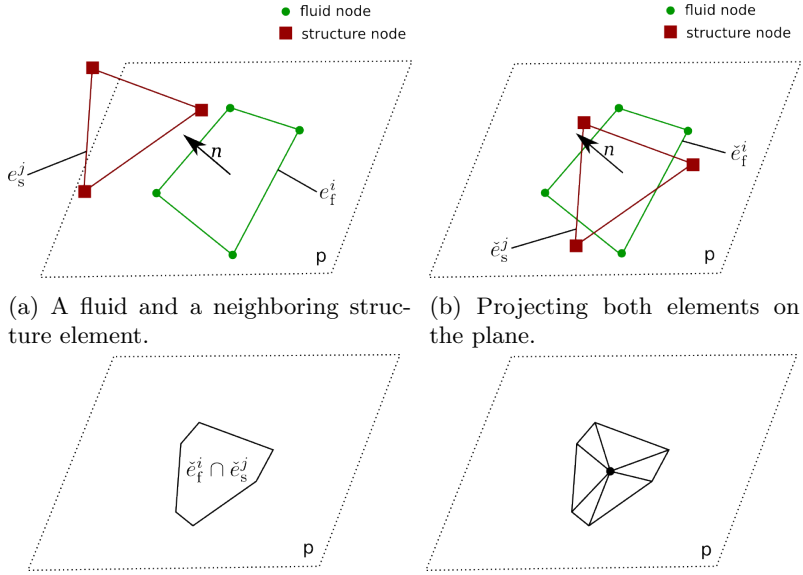


Figure 4.4: Using the standard mortar method for the 1D test.

Enforcing consistency

The integration on the right hand side (RHS) of (4.20) can be computed by assembling element-wise evaluations. Fig. 4.5 shows one part of

the global integration domain $\Gamma_{s \to f}$, where the shape functions of a projected structure element and the shape functions of a fluid element are evaluated and then multiplied. The details of the clipping process will be presented in Section 4.3.4.



(a) A fluid and a neighboring structure element.

(b) Projecting both elements on the plane.

(c) Clipping to get the overlap area.

(d) Triangulation.

Figure 4.5: Procedure of computing one part of the integration domain $\Gamma_{s \to f}$ which is the overlap between a fluid element and a neighboring structure element (this figure is an adapted presentation of the one in [114]).

As shown in A.1, the standard mortar method is not exactly consistent since $\Gamma_f \neq \Gamma_{s \to f}$ due to curved edges of an interface. An example is given in Fig. 4.6a, where a planar surface with a curved edge is shown. Since the fluid elements at the curved edge are clipped by the structure elements, the integral on the area without overlap is equal to 0, so the displacement field cannot be transferred to the fluid elements completely. The problem gets more severe when the mesh density ratio is higher as shown in Fig. 4.6b where some fluid elements are totally out of contact with the structure mesh. The same phenomenon also happens on 3D curved surfaces, where fluid elements are not fully overlapped

by *projected* structure elements. The best solution is to expand the shape functions of structure elements beyond its discretized boundary and compute the mortar integral also outside the discretized boundary. However, automatically detecting the fluid elements at curved edges and using a different clipping algorithm for these elements raise big complexities for implementation. Since the problem happens only to a small part of the wet surface, it can be solved by directly adjusting the numbers in the matrices using the consistency constraint. The state of the art is to scale up the structure shape functions to compensate the deficit due to the area out of contact [29, 111]. However, the method cannot handle fluid elements totally out of contact with structure elements as shown in Fig. 4.6b, because the scaling factor becomes infinity. These elements can be regarded as out of contact in contact problems, but they must move together with other fluid elements in FSI problems so that the wet surface moves as a whole. Moreover, this effect can naturally arise in FSI, where highly turbulent flows on fine meshes are coupled to structural discretizations which are typically much coarser.

The idea of scaling is also used in this work as the first step of enforcing the consistency. The scaling factor can be derived from the consistency constraint which requires that all row-sums of $\mathbf{M}_{\mathrm{ff}}$ and $\mathbf{M}_{\mathrm{fs}}$ in (4.21) are the same (this condition is obtained by assuming that constant fields $\mathbf{U}_{\mathrm{f}} = \mathbf{1}_{\mathrm{f}}$ and $\mathbf{U}_{\mathrm{s}} = \mathbf{1}_{\mathrm{s}}$ can satisfy (4.21)). The scaling factor for each row i of $\mathbf{M}_{\mathrm{fs}}$ is defined as

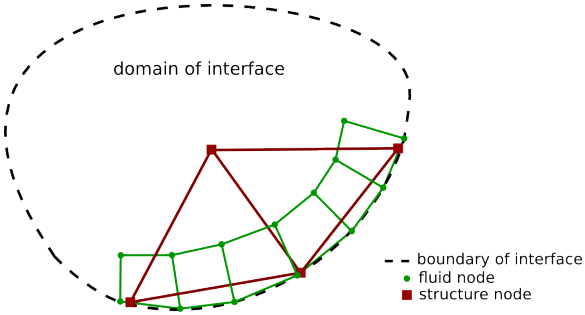
$$\alpha^i = \frac{\sum_{j=1}^{n_{\mathrm{f}}} \mathbf{M}_{\mathrm{ff}}^{ij}}{\sum_{j=1}^{n_{\mathrm{s}}} \mathbf{M}_{\mathrm{fs}}^{ij}} = \frac{\int_{\Gamma_{\mathrm{f}}} \mathbf{N}_{\mathrm{f}}^i(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{f}}}{\int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_{\mathrm{f}}^i(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}}, \quad \text{for } i = 1, ..., n_{\mathrm{f}}. \tag{4.25}$$

And $\mathbf{M}_{\mathrm{fs}}$ is updated as

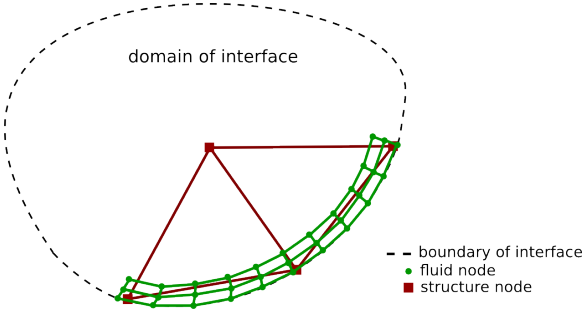
$$\mathbf{M}_{\mathrm{fs}}^{ij} = \mathbf{M}_{\mathrm{fs}}^{ij} \times \alpha^i, \quad \text{for } i = 1, ..., n_{\mathrm{f}}, j = 1, ..., n_{\mathrm{s}}. \tag{4.26}$$

It can be interpreted in a way that the structure shape functions in ith row are scaled up by α^i in (4.20). The consequence is that the structure displacement field is amplified to compensate the loss of data on the area out of contact. However, the scaling factor approaches infinity if related fluid elements are totally out of contact.

The newly developed enforcing consistency approach embeds the nearest element inter-/extrapolation into the weak form. It is based on the computed scaling factors. If the scaling factor α^k of the k-th fluid node has $\alpha^k < C$, scaling in (4.26) is used; Otherwise, nearest element inter-/extrapolation will be activated. The weights w_l are computed as



(a) Moderate mesh density ratio.



(b) High mesh density ratio.

Figure 4.6: Different discretizations of a planar surface with a curved edge. Only part of the elements are drawn to highlight the contact relation.

in (4.14) for each node l in the nearest element. They are then used to modify $\mathbf{M}_{\text{fs}}$: all entries in the k-th row of $\mathbf{M}_{\text{fs}}$ are set to 0 except $\mathbf{M}_{\text{fs}}^{kl}$ which is calculated by

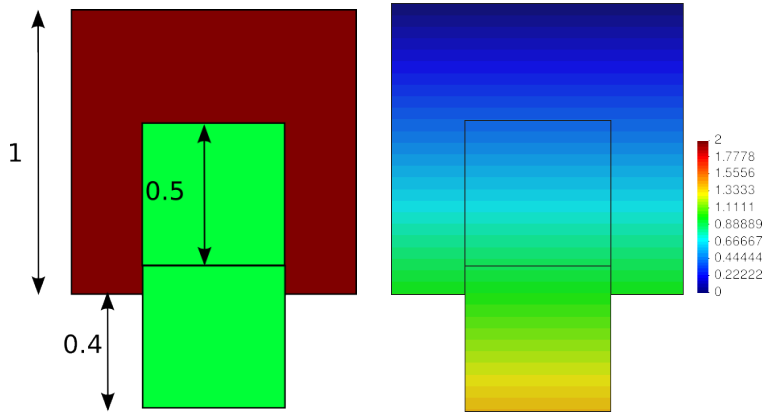
$$\mathbf{M}_{\text{fs}}^{kl} = w_l \int_{\Gamma_{\text{f}}} \mathbf{N}_{\text{f}}^{k}(\mathbf{x}) \, \mathrm{d}\Gamma_{\text{f}}. \tag{4.27}$$

The method is consistent since the row-sums are equal. In (4.27), w_l can be moved into the integration then it takes the place of the structure shape function, so the embedded approach can be interpreted as replacing the structure shape function by a constant function with the value w_l.

If $C = 1$, the scaling is never used. But it is found that the scaling can give acceptable result if the portion in contact is big enough.

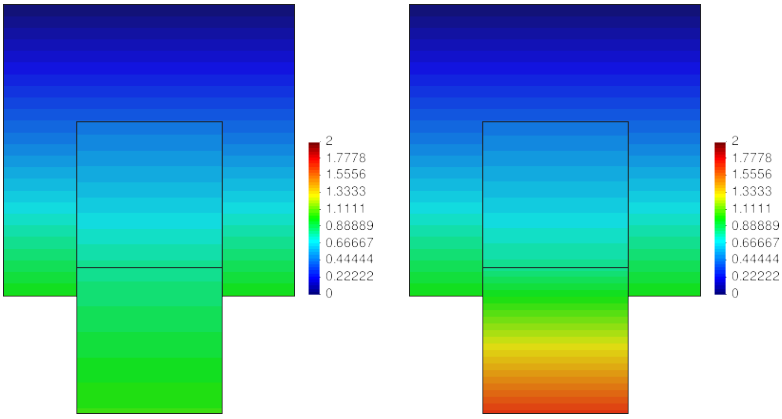
Therefore, it is chosen as $C = 1.1$ based on many practical examples.

To compare the performance between scaling and the new algorithm, a simple example with only one structure and two fluid elements is designed, as shown in Fig. 4.7a. The displacement is set to 0 at the upper edge and 1 at the lower edge of the structure element. The reference result in Fig. 4.7b comes from using the nearest element interpolation which is consistent due to its ability of extrapolation. Different results of using scaling and the new enforcing consistency algorithm are shown in Fig. 4.8. It can be seen that the scaling gives almost constant field on the area out of overlap, while the embedded inter-/extrapolation is able to represent the gradient but it is overestimated. The overestimation is a result of using constant functions at the places of structure shape functions in the weak form. The error due to the overestimation is bounded since the value of the constant function is interpolated from the structure shape function. Moreover, the embedded method still has the advantage in handling fully non-overlapped fluid elements in which case the scaling would fail.



(a) Test example. (b) Reference result.

Figure 4.7: Simple example with one structure element and two fluid elements. The structure element has edges of length 1 and the fluid elements has edges of length 0.5 as shown in (a). The source field and the reference mapping result are shown together in (b).



(a) Scaling.

(b) Embedded inter- and extrapolation.

Figure 4.8: Different results from applying different enforcing consistency algorithms in the standard mortar method.

4.3.3 Dual Mortar Method

The vector of dual shape functions $\hat{\mathbf{N}}_\mathrm{f}(\mathbf{x})$ is defined as

$$\int_{\Gamma_\mathrm{f}} \hat{\mathbf{N}}_\mathrm{f}^i(\mathbf{x}) \mathbf{N}_\mathrm{f}^j(\mathbf{x}) \, \mathrm{d}\Gamma_\mathrm{f} = \delta^{ij} \int_{\Gamma_\mathrm{f}} \mathbf{N}_\mathrm{f}^j(\mathbf{x}) \, \mathrm{d}\Gamma_\mathrm{f}, \quad \text{for } i,j = 1,...,n_\mathrm{f} \tag{4.28}$$

where δ^{ij} is the kronecker delta, which is equal to 0 if $i \neq j$ and 1 if $i = j$. $\hat{\mathbf{N}}_\mathrm{f}(\mathbf{x})$ is computed element-wise so that it is not continuous across elements. The reader can refer to [113] for the details of its computation.

In (4.20), replacing the test functions by $\hat{\mathbf{N}}_\mathrm{f}$ results in

$$\int_{\Gamma_\mathrm{f}} \hat{\mathbf{N}}_\mathrm{f}(\mathbf{x}) \mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x}) \, \mathrm{d}\Gamma_\mathrm{f} \, \mathbf{U}_\mathrm{f} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \hat{\mathbf{N}}_\mathrm{f}(\mathbf{x}) \mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x}) \, \mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}} \, \mathbf{U}_\mathrm{s}. \tag{4.29}$$

This can be rewritten in matrix-vector form as

$$\mathbf{D}_\mathrm{ff} \mathbf{U}_\mathrm{f} = \hat{\mathbf{M}}_\mathrm{fs} \mathbf{U}_\mathrm{s}. \tag{4.30}$$

$\mathbf{D}_\mathrm{ff}$ is a diagonal matrix according to the definition in (4.28), so (4.30) can be solved directly by inverting $\mathbf{D}_\mathrm{ff}$, which is the well-known advantage compared to the standard mortar method.

With (4.7) the direct displacement mapping matrix is obtained:

$$\mathbf{H}_{\mathrm{fs}} = \mathbf{D}_{\mathrm{ff}}^{-1}\hat{\mathbf{M}}_{\mathrm{fs}}. \tag{4.31}$$

And the direct traction mapping matrix can be obtained analogously as

$$\mathbf{H}_{\mathrm{sf}} = \mathbf{D}_{\mathrm{ss}}^{-1}\hat{\mathbf{M}}_{\mathrm{sf}}. \tag{4.32}$$

However, the dual mortar algorithm is usually applied in a conservative way as the standard one.

The conservative traction mapping is obtained by applying (4.31) in (4.10) as

$$\mathbf{M}_{\mathrm{ss}}\mathbf{P}_{\mathrm{s}} = \hat{\mathbf{M}}_{\mathrm{fs}}^{\mathrm{T}}\mathbf{D}_{\mathrm{ff}}^{-1}\mathbf{M}_{\mathrm{ff}}\mathbf{P}_{\mathrm{f}}, \tag{4.33}$$

which can be rewritten as

$$\int_{\Gamma_{\mathrm{s}}} \mathbf{N}_{\mathrm{s}}(\mathbf{x})\mathbf{N}_{\mathrm{s}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}}\,\mathbf{P}_{\mathrm{s}} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_{\mathrm{s}}(\mathbf{x})\hat{\mathbf{N}}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}\,\hat{\mathbf{P}}_{\mathrm{f}}. \tag{4.34}$$

$\hat{\mathbf{P}}_{\mathrm{f}}$ is an average of $\mathbf{P}_{\mathrm{f}}$ defined as

$$\hat{\mathbf{P}}_{\mathrm{f}} = \mathbf{D}_{\mathrm{ff}}^{-1}\mathbf{M}_{\mathrm{ff}}\mathbf{P}_{\mathrm{f}}, \tag{4.35}$$

whose entries are explicitly computed by

$$\hat{\mathbf{P}}_{\mathrm{f}i} = \frac{\int_{\Gamma_{\mathrm{f}}} \mathbf{N}_{\mathrm{f}i}(\mathbf{x})p_{\mathrm{f}}^{h}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{f}}}{\int_{\Gamma_{\mathrm{f}}} \mathbf{N}_{\mathrm{f}i}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{f}}}. \tag{4.36}$$

In (4.34) a new weak form is obtained where $\mathbf{N}_{\mathrm{s}}(\mathbf{x})$ becomes the test functions and $\hat{\mathbf{P}}_{\mathrm{f}}$ is interpolated by $\hat{\mathbf{N}}_{\mathrm{f}}^{\mathrm{T}}(\mathbf{x})$ which is the non-continuous dual shape functions. As a result, non-continuous traction field is mapped to the structure. Whether the non-continuity appears in the structure traction field depends on the grid density of the structure mesh. If the structure mesh is coarser than the fluid mesh, the non-continuity is blurred out. Otherwise, it will appear in the form of slight oscillations.

It can also be seen that the conservative traction mapping in (4.33) is not equal to the direct traction mapping in (4.32). However, the conservative mapping is consistent if $\Gamma_{\mathrm{f}} = \Gamma_{\mathrm{s}}$. The detailed proof of the inconsistency of the dual mortar method is put in A.2.

Using the dual mortar method for the same 1D test in Fig. 4.4 gives almost the same result as the standard one and $\mathbf{H}_{\mathrm{fs}}$ of the dual version

is

$$\mathbf{H}_{\mathrm{fs}} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{5} & \frac{2}{5} & 0 \\ \frac{7}{40} & \frac{17}{20} & -\frac{1}{40} \\ -\frac{1}{40} & \frac{17}{20} & \frac{7}{40} \\ 0 & \frac{2}{5} & \frac{3}{5} \\ 0 & 0 & 1 \end{pmatrix}.$$

The corresponding conservative mapping matrix is

$$\mathbf{M}_{\mathrm{ss}}^{-1}\mathbf{H}_{\mathrm{fs}}^{\mathrm{T}}\mathbf{M}_{\mathrm{ff}} = \begin{pmatrix} \frac{29}{50} & \frac{67}{100} & -\frac{1}{100} & -\frac{1}{4} & -\frac{1}{20} & \frac{3}{50} \\ -\frac{3}{25} & \frac{9}{100} & \frac{53}{100} & \frac{53}{100} & \frac{9}{100} & -\frac{3}{25} \\ \frac{3}{50} & -\frac{1}{20} & -\frac{1}{4} & -\frac{1}{100} & \frac{67}{100} & \frac{29}{50} \end{pmatrix}.$$

Conservative mapping of a given constant traction field $\mathbf{P}_{\mathrm{f}}^{\mathrm{T}} = (1, 1, 1, 1, 1, 1)$ results in $\mathbf{P}_{\mathrm{s}}^{\mathrm{T}} = (1, 1, 1)$, so the conservative mapping with the dual mortar method is consistent for this special test (since the domain is flat and the boundaries are matching exactly).

Enforcing consistency

The dual mortar method has also the inconsistency problem at curved edges as the standard version. The same enforcing consistency algorithm is applied to modify $\hat{\mathbf{M}}_{\mathrm{fs}}$. The scaling factor for each row i of $\hat{\mathbf{M}}_{\mathrm{fs}}$ is defined as

$$\alpha^i = \frac{\mathbf{D}_{\mathrm{ff}}^{ii}}{\sum_{j=1}^{n_{\mathrm{s}}} \hat{\mathbf{M}}_{\mathrm{fs}}^{ij}} = \frac{\int_{\Gamma_{\mathrm{f}}} \mathbf{N}_{\mathrm{f}}^i(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{f}}}{\int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \hat{\mathbf{N}}_{\mathrm{f}}^i(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}} = \frac{\int_{\Gamma_{\mathrm{f}}} \hat{\mathbf{N}}_{\mathrm{f}}^i(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{f}}}{\int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \hat{\mathbf{N}}_{\mathrm{f}}^i(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}}, \quad (4.37)$$

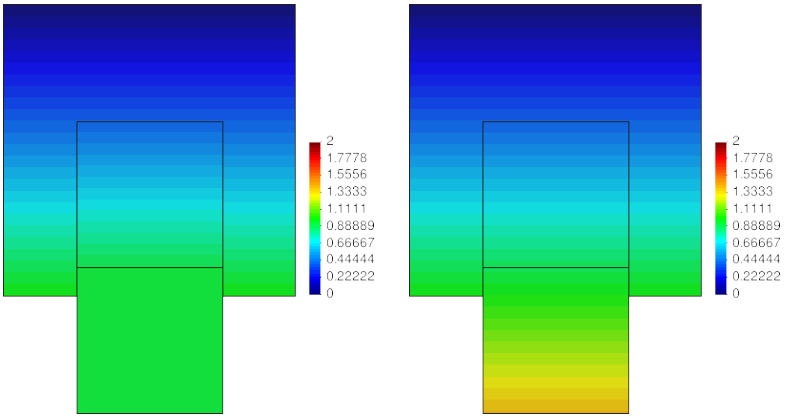
for $i = 1, ..., n_{\mathrm{f}}$. If a scaling factor α^k satisfies $0 < \alpha^k < 1.1$ (1.1 is an empirical number as in the standard mortar and α^k can be negative since dual shape functions have negative range), then $\hat{\mathbf{M}}_{\mathrm{fs}}^{ij}$ is updated by scaling as

$$\hat{\mathbf{M}}_{\mathrm{fs}}^{ij} = \hat{\mathbf{M}}_{\mathrm{fs}}^{ij} \times \alpha^i, \quad \text{for } i = 1, ..., n_{\mathrm{f}}, j = 1, ..., n_{\mathrm{s}}. \quad (4.38)$$

Otherwise, nearest element inter-/extrapolation will be performed to compute the weights w_l for each node l in the nearest element. All entries in the k-th row of $\hat{\mathbf{M}}_{\mathrm{fs}}$ are set to 0 except $\hat{\mathbf{M}}_{\mathrm{fs}}^{kl}$ which is calculated by

$$\hat{\mathbf{M}}_{\mathrm{fs}}^{kl} = w_l \int_{\Gamma_{\mathrm{f}}} \mathbf{N}_{\mathrm{f}}^k(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{f}}. \quad (4.39)$$

Due to the diagonal matrix in the dual mortar method, the value on the fluid node is equal to the inter-/extrapolated result. Applying the dual mortar method on the same example in Fig. 4.7 gives the result shown in Fig. 4.9. The scaling gives again almost constant field on the area out of contact and the embedded inter-/extrapolation can represent the gradient more accurately since the values of the four lower nodes are actually inter-/extrapolated.



(a) Scaling.

(b) Embedded inter- and extrapolation.

Figure 4.9: Different results from applying different enforcing consistency algorithms in the dual mortar method.

4.3.4 Implementation of Mortar Methods

The implementation realized in this contribution allows mixed meshes which may consist of both linear triangular and bilinear quadrilateral elements. Elements whose geometry is a concave quadrilateral or a polygon with more than four edges can be first converted into allowed types before mapping.

The computation of the direct displacement mapping matrix $\mathbf{H}_{\mathrm{fs}}$ according to (4.22) or (4.31) will be presented. The conservative traction mapping matrix can be obtained thereafter by transposing $\mathbf{H}_{\mathrm{fs}}$. For each fluid element, the elemental contribution to $\mathbf{M}_{\mathrm{ff}}$ or $\mathbf{D}_{\mathrm{ff}}$ is computed by Gauss integration and then assembled. One remark is that bilinear quadrilateral elements are always projected on the normal plane at the

center when computing $\mathbf{M}_{\mathrm{ff}}$ or $\mathbf{D}_{\mathrm{ff}}$. This treatment avoids inconsistency caused by implementation. The algorithm for computing $\mathbf{M}_{\mathrm{fs}}$ or $\hat{\mathbf{M}}_{\mathrm{fs}}$ is similar to that in [114] which is listed below (see Fig. 4.5):

1. Loop over the fluid elements, for the i-th fluid element e_{f}^i:
   a) Determine the search radius for neighbor searching. The radius should be big enough to contain all structure elements which are in contact with e_{f}^i. It is computed based on the edge lengths of e_{f}^i and its neighboring elements.
   b) Search for the structural nodes within the radius and structure elements containing these nodes are regarded as neighboring elements. Attention should be paid on thin structures such as an airfoil, where fluid elements on the upper surface may be falsely regarded as in contact with some structure elements on the lower surface, or vice verse, since there will be overlapped area after projection. This can be avoided by comparing the normal directions of the structure elements and the normal direction of e_{f}^i. Structure elements that have wrong normal directions are regarded as out of contact and are removed from the set of neighboring elements.
   c) Loop over the remaining neighboring elements, for the j-th element e_{s}^j:
      i. Form plane p from the normal n and the geometric center of e_{f}^i, see Fig. 4.5a.
      ii. Project both e_{s}^j and e_{f}^i to p to get $\check{e}_{\mathrm{s}}^j$ and $\check{e}_{\mathrm{f}}^i$, see Fig. 4.5b. (If e_{f}^i is a triangle, it does not have to be projected since $e_{\mathrm{f}}^i = \check{e}_{\mathrm{f}}^i$.)
      iii. Clip $\check{e}_{\mathrm{f}}^i$ by $\check{e}_{\mathrm{s}}^j$ to get $\check{e}_{\mathrm{f}}^i \cap \check{e}_{\mathrm{s}}^j$ using the Sutherland-Hodgman algorithm in [53], see Fig. 4.5c.
      iv. The polygon formed by $\check{e}_{\mathrm{f}}^i \cap \check{e}_{\mathrm{s}}^j$ is triangulated by its centroid, see Fig. 4.5d.
      v. Perform Gauss integration on the triangles to compute one part of $\mathbf{M}_{\mathrm{fs}}$ or $\hat{\mathbf{M}}_{\mathrm{fs}}$. Add the result into the global $\mathbf{M}_{\mathrm{fs}}$ or $\hat{\mathbf{M}}_{\mathrm{fs}}$.
   d) End loop.
2. End loop.
3. Apply the enforcing consistency algorithm. The scaling factors are computed according to (4.25) or (4.37), which are used to

determine whether to scale the entries in the mortar matrix or to reset them through inter-/extrapolation. The details can be found in Section 4.3.2 and 4.3.3.

$\mathbf{H}_{\mathrm{fs}}$ does not have to be computed explicitly. Instead, the equations (4.21) and (4.30) are solved with given $\mathbf{U}_{\mathrm{s}}$. The sparse matrices are stored in the compressed row storage (CRS) format to be storage efficient.

4.3.5 Computational Cost

The computation includes the construction of mapping matrices during the initialization stage and data mapping during the running stage. The computational cost of constructing a mapping matrix is dominated by the neighbor searching procedure. The neighbor searching library FLANN ([100]) is used which has efficient tree-like data structures. Once a tree is constructed, the neighbor searching process can be parallelized with multiple threads. Assume that a single qualitative number N can represent the number of nodes or elements of both the structure and the fluid meshes, then the construction of a tree costs $\mathcal{O}(N \cdot \log N)$, and the neighbor searching also costs $\mathcal{O}(N \cdot \log N)$ with a single thread. One data mapping operation costs $\mathcal{O}(N)$ except for the standard mortar method where additional factorization and equation solving are required. In this work, the solver PARDISO [117] is used for solving sparse linear systems.

Highly fine meshes are generated from the catenoid surface in Section 4.4.2 to investigate the computation time. The fluid mesh has 2000×2000 quadrilateral elements and the structure mesh has 700×700 quadrilateral elements. The test is performed on a machine with Intel® Core™ i7-2600 processors. The time used in direct displacement mapping with different mapping algorithms is given in Table 4.1. It can be seen that the dual mortar algorithm is as efficient as the nearest element interpolation in data mapping. The standard mortar algorithm takes more time but it is still small compared with the time that would be taken in solving the whole equation system of FSI.

4.4 Convergence Tests

The convergence behavior of different mapping algorithms is tested on three prototypic geometries with increasing geometrical complexity.

Computation time (sec.)	Initialization	Mapping
Nearest element interpolation	9.03	0.05
Standard mortar method	108.95	0.7
Dual mortar method	69.18	0.04

Table 4.1: Computation time of different algorithms on the test case.

4.4.1 Evaluation of Discretization Error and Mapping Error

Assume a continuous field $g(\mathbf{x})$ is defined on a curved surface Γ which is meshed into Γ_a, as illustrated in Fig. 4.10. The deviation between $g(\mathbf{x})$ and a corresponding discrete field on Γ_a is

$$\epsilon(\mathbf{x}) = g(\mathbf{x}) - \mathbf{N}_a(\mathbf{x}_a)^{\mathrm{T}}\mathbf{G}_a, \tag{4.40}$$

where $\mathbf{x}_a$ is the projection of $\mathbf{x}$ on Γ_a, $\mathbf{N}_a(\mathbf{x}_a)$ is the shape function vector of Γ_a evaluated at $\mathbf{x}_a$ and $\mathbf{G}_a$ are the nodal evaluations of $g(\mathbf{x})$ on Γ_a. (Because $\mathbf{N}_a(\mathbf{x})$ with $\mathbf{x} \in \Gamma$ cannot be evaluated directly, it is approximated by $\mathbf{N}_a(\mathbf{x}_a)$ with $\mathbf{x}_a \in \Gamma_a$.) The global error is defined by the root mean square (RMS) of the deviations on n sampling points as

$$\epsilon = \sqrt{\frac{\sum_{i=1}^{n}(g(\mathbf{x}^i) - \mathbf{N}_a(\mathbf{x}_a^i)^{\mathrm{T}}\mathbf{G}_a)^2}{n}}. \tag{4.41}$$

The larger n is, the more accurate is the error estimation. The evaluation of $\mathbf{N}_a(\mathbf{x}_a^i)^{\mathrm{T}}\mathbf{G}_a$ is exactly the same process of mapping $\mathbf{G}_a$ from Γ_a to the sampling mesh using nearest element interpolation. If $\mathbf{G}_a$ results from the discretization of $g(\mathbf{x})$, ϵ is defined as the discretization error; if it results from mapping, ϵ is defined as the mapping error.

The error evaluation approach has the following advantages:

- The error over the whole continuous domain is reflected by dense sampling.
- Discretization and mapping errors are evaluated on the same sampling mesh, so they can be compared together.

4.4.2 Geometries and Analytical Fields

The convergence behavior of different mapping algorithms is tested with three different geometries of increasing complexity: flat, singly curved

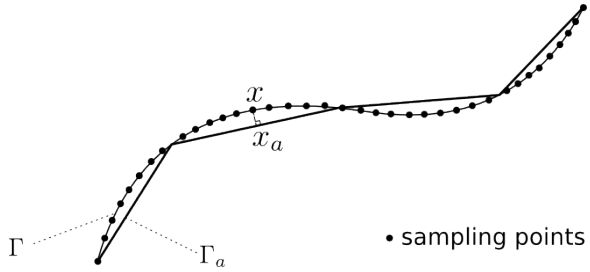
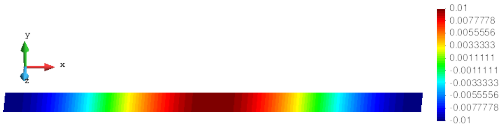


Figure 4.10: Discretization of a curved surface (represented by a 2D curve) with linear elements. Sampling points are used in error evaluation. In fact, the sampling points form an additional sampling mesh.

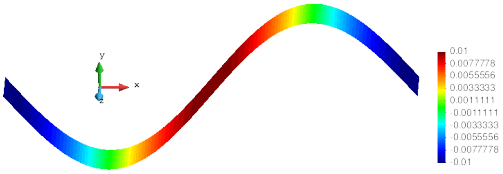
and doubly curved. The geometries and the analytical fields on them are shown in Fig. 4.11, where the first two come from [32]. The *extruded line* is defined by $y = 0, x \in [-0.5, 0.5]$ and the *extruded curve* is defined by $y = 0.2\sin(2\pi x), x \in [-0.5, 0.5]$. Both are extruded in the z-direction to form a surface. The displacement and traction fields are defined by the same analytical function $g(x, y, z) = 0.01\cos(2\pi x)$. Both geometries are meshed equidistantly into 26×2^k fluid elements and 5×2^k structure elements along the length direction, with $k \in \{0, 1, 2, 3, 4, 5\}$. The sampling mesh corresponding to each geometry has 10000 equidistant elements along the length direction. The *catenoid* is described parametrically by $x = \cos(u)\cosh(v)$, $y = v$ and $z = \sin(u)\cosh(v)$ with $u \in [0, \pi]$ and $v \in [-1.5, 1.5]$. The displacement and traction fields are defined by $g(x, y, z) = \sin(3x + 3y)$. The geometry is meshed into $(11 \times 2^k)^2$ fluid elements and $(5 \times 2^k)^2$ structure elements, with $k \in \{0, 1, 2, 3, 4, 5\}$. The sampling mesh has 2000^2 elements. All meshes are structured and consist of quadrilateral elements. It needs to be emphasized that the fluid elements are not fully overlapped by the structure elements at curved edges of the catenoid, as shown in Fig. 4.12. Another remark is that the curved edges of the extruded curve do not cause inconsistent displacements because the fluid elements are fully overlapped by the *projected* structure elements.

4.4.3 Results and Conclusions

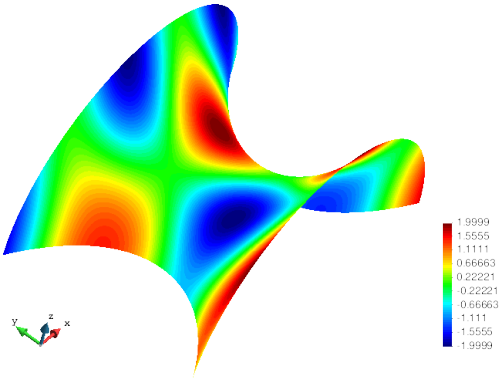
The prescribed fields are mapped between the structure and fluid meshes at the same refinement level, i.e. with the same value of k. Direct mapping of displacement is carried out with all algorithms. The trac-



(a) Extruded line.



(b) Extruded curve.



(c) Catenoid.

Figure 4.11: Test geometries and the analytical fields defined on them.

tion is mapped both directly and conservatively using the nearest element interpolation but it is only mapped conservatively with the mortar methods. The results are shown in Fig. 4.13, 4.14 and 4.15, and the conclusions are given in the following:

- The discretization error converges with 2nd order on all geometries as a result of bilinear shape functions.
- With nearest element interpolation, the direct traction mapping gives always good results but the conservative traction mapping is not convergent and the error stagnates from a certain level of refinement on.

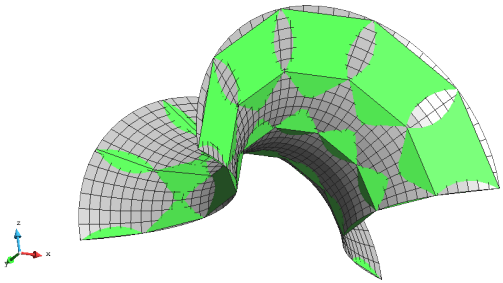


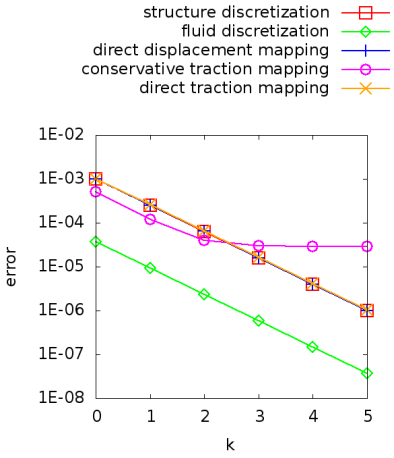
Figure 4.12: The meshes of the catenoid when $k = 0$.

- Both mortar methods give very close results.

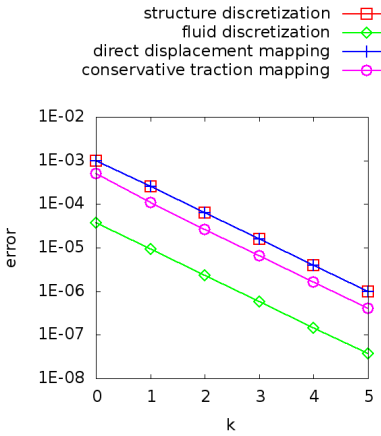
- Both mortar methods have 2nd order convergence on the extruded line and the extruded curve i.e. in the case that the elements of one mesh are fully overlapped by the (projected) elements of the other.

- Due to the curved edges on the catenoid, the direct displacement mapping with both mortar methods shows a deterioration of the convergence rate. The enforcing consistency method helps to improve the convergence rate of the direct displacement mapping, but the convergence rate of the conservative traction mapping is compromised. The reason is that the load on the area out of overlap is additionally added to the structure nodes at the curved edge as a result of force and energy conservation. This is the common problem of all enforcing consistency methods. However, it must be stressed that the improvement of the local quality of the mapped displacements i.e. smoothness of the edges in the deformed configuration is much more significant. Otherwise, it can lead to distortion of the fluid mesh at the curved edges in the deformed configuration which typically results in failure of fluid computation.

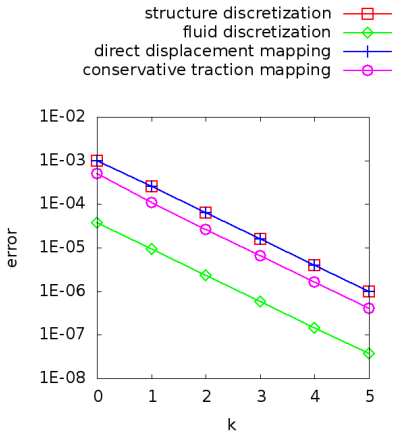
- If traction is mapped in a direct way with the nearest element interpolation method, difference in energy will be caused as shown in Fig. 4.16. But the difference converges approximately with 2nd order.



(a) Nearest element interpolation.

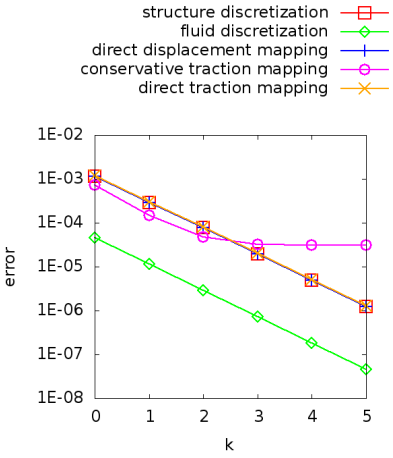


(b) Standard mortar.

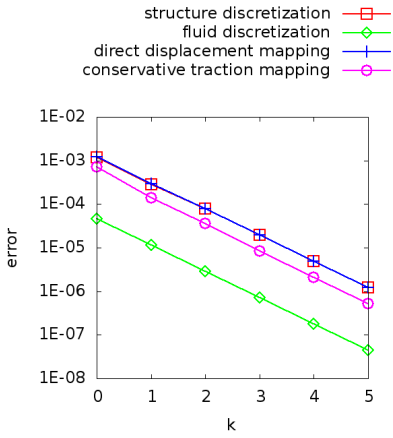


(c) Dual mortar.

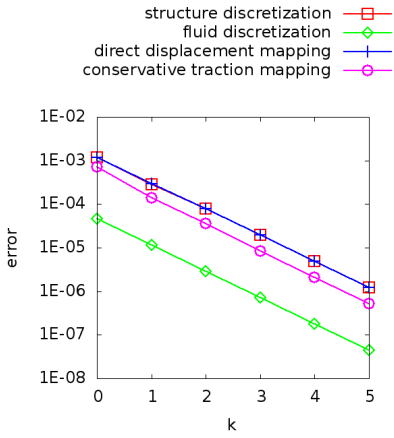
Figure 4.13: Convergence of the discretization and the mapping errors on the extruded line.



(a) Nearest element interpolation.

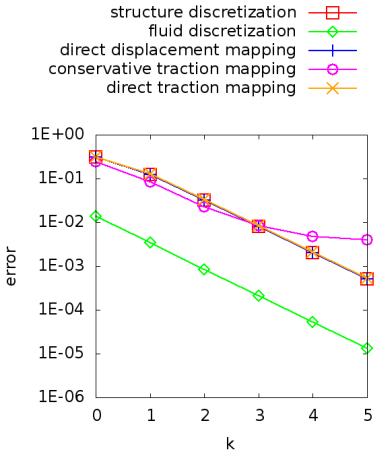


(b) Standard mortar.

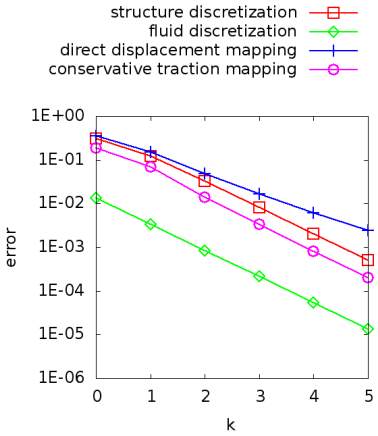


(c) Dual mortar.

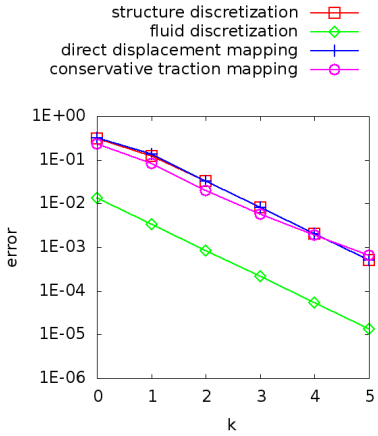
Figure 4.14: Convergence of the discretization and the mapping errors on the extruded curve



(a) Nearest element interpolation.



(b) Standard mortar without enforcing consistency.



(c) Standard mortar with enforcing consistency.

Figure 4.15: Convergence of the discretization and the mapping errors on the catenoid. The dual mortar method gives very close results to the standard one, so it is not additionally presented.

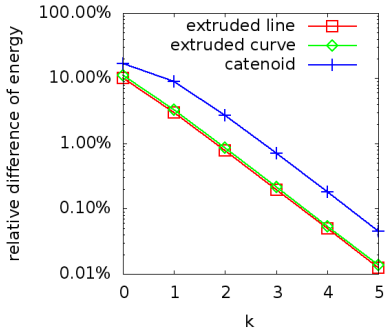


Figure 4.16: Convergence of energy difference from the direct use of nearest element interpolation on different geometries.

4.5 Examples

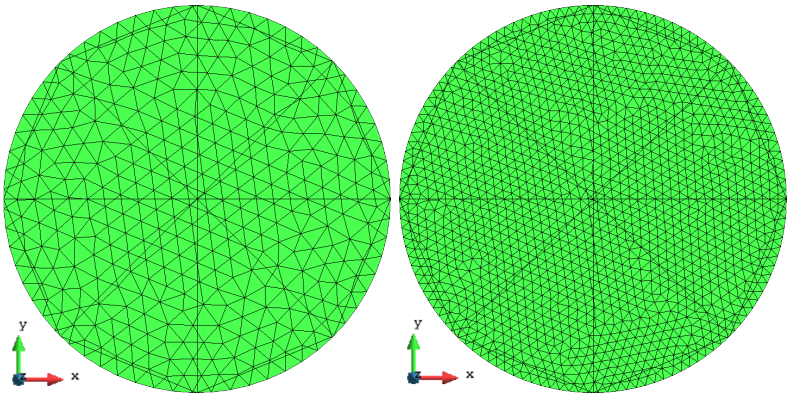
Three examples are presented in this section. The first two examples consist of simple geometries to demonstrate the problems that may happen when using mortar methods or doing conservative mapping. These problems are reproduced in the third example of a wind turbine blade which is a typical situation for the need of robust non-matching grid treatment in practice.

4.5.1 Circular Plate

The circular plate is a typical example of geometries with curved edges. The structure mesh has 8 triangular elements and the fluid mesh has two levels of refinement with 691 and 2856 triangular elements respectively, as shown in Fig. 4.17. The nearest element interpolation is not tested since it does not have the inconsistency problem.

Firstly, a constant displacement field with value 1 is assigned on the structure mesh as shown in Fig. 4.18, which is mapped to both fluid meshes with the standard and the dual mortar methods. The results without enforcing consistency are shown in Fig. 4.19. It can be seen that both methods are inconsistent due to the curved edge. The constant field can be recovered by applying the enforcing consistency algorithm which is guaranteed by definition, so the results are not shown additionally. One remark is that the scaling approach cannot be applied for the finer fluid mesh since the fully non-overlapped fluid elements cause scaling factors equal to infinity.

Another displacement field in the z-direction is assigned to the structure mesh, where 0 is defined on the centered node and 1 on the other nodes. The field and the deformed mesh are shown in Fig. 4.20. Applying mortar methods without enforcing consistency results in highly distorted deformed meshes. The results from using the standard mortar method on both meshes are shown in Fig. 4.21. The results of applying pure scaling on the coarser mesh is shown in Fig. 4.22. Applying it on the finer mesh leads to program error since some scaling factors approach infinity. It can be seen from the results that the deformed mesh is somehow evened at the edge which means the gradient in the displacement field cannot be recovered by scaling. Moreover, the negative range of the dual shape functions make the dual mortar method more sensitive to inconsistency problem. The results from applying embedded inter-/extrapolation on both fluid meshes are shown in Fig. 4.23. It can be seen that the gradient of the field at the edge is recovered smoothly by the dual mortar method but it is slightly overestimated by the standard one. However, the smoothness of the deformed mesh is still acceptable in the overestimation case.



(a) Coarser fluid mesh overlapped by the structure mesh.

(b) Finer fluid mesh overlapped by the structure mesh.

Figure 4.17: Meshes of a circular plate including one structure mesh and two fluid meshes with different levels of refinement. The fluid meshes are shown with the structure mesh overlapping on them to visualize the area out of contact.

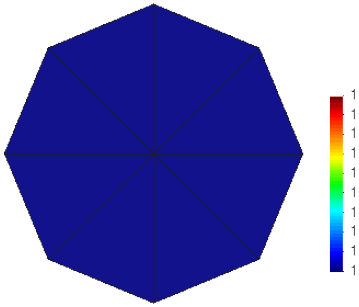
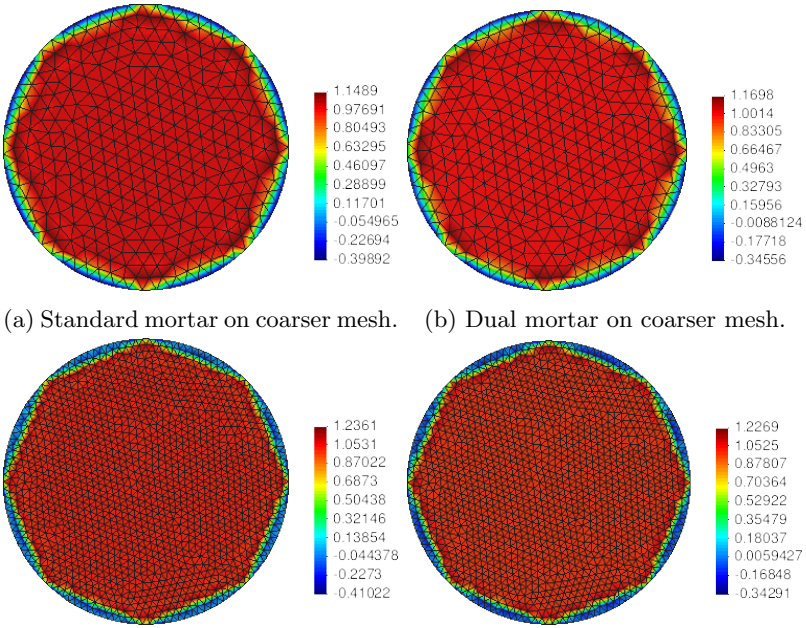


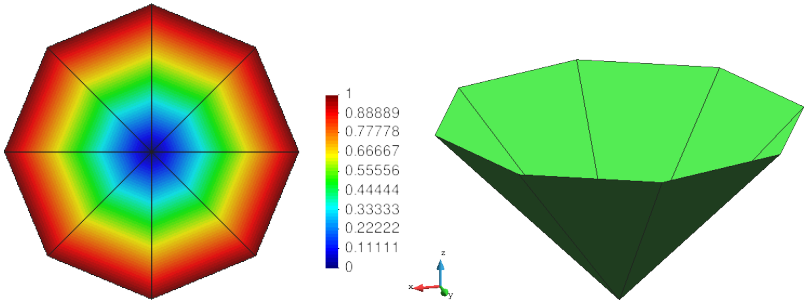
Figure 4.18: Constant field on the structure mesh.



(a) Standard mortar on coarser mesh. (b) Dual mortar on coarser mesh.

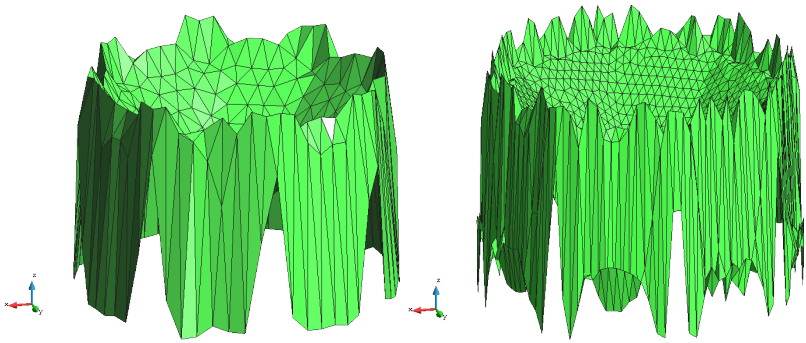
(c) Standard mortar on finer mesh. (d) Dual mortar on finer mesh.

Figure 4.19: Results from mapping the constant field using mortar methods without enforcing consistency.



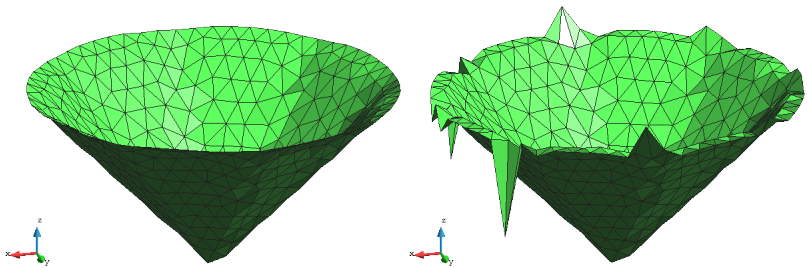
(a) Field. (b) Deformed mesh.

Figure 4.20: A displacement field in z-direction on the structure mesh and the resulting deformed mesh.



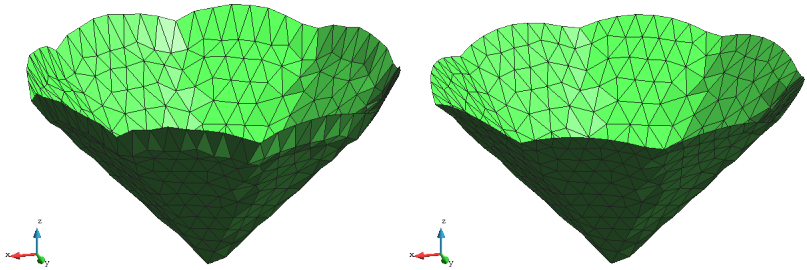
(a) Coarser mesh. (b) Finer mesh.

Figure 4.21: Deformed meshes from the standard mortar method without enforcing consistency. Those from the dual mortar method are similar, hence they are not additionally shown.

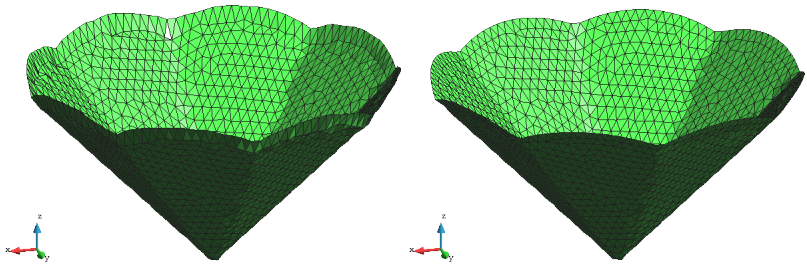


(a) Standard mortar. (b) Dual mortar.

Figure 4.22: Deformed meshes from enforcing consistency with pure scaling on the coarser mesh.



(a) Standard mortar on coarser mesh. (b) Dual mortar on coarser mesh.

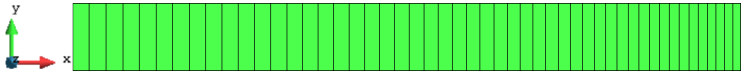


(c) Standard mortar on finer mesh. (d) Dual mortar on finer mesh.

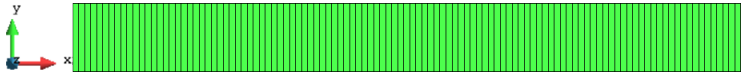
Figure 4.23: Deformed meshes from enforcing consistency with embedded nearest element inter-/extrapolation.

4.5.2 Finer Structure Mesh

As shown in previous sections, the conservative mapping with nearest element interpolation generally gives bad results. Moreover, the conservative mapping with the dual mortar method will give oscillating results when the structure mesh is finer than the fluid mesh. This situation can also happen within some sub-domains of complex coupling surfaces if the structure and the fluid meshes are generated independently from each other. The problem is demonstrated with the geometry of the *extruded line* and the same analytical field as shown in Fig. 4.11a. The fluid mesh has 50 elements with gradient spacing in order to obtain non-continuous dual shape functions, while the structure mesh has 111 equal-sized elements, see Fig. 4.24. The results of conservative mapping with all mapping algorithms are shown in Fig. 4.25 and also drawn along the x-direction in Fig. 4.26. The standard mortar method gives results almost coincided with the analytical solution while the dual one gives results with small oscillations superposed on the correct solution; the nearest element interpolation gives unacceptably large oscillations.

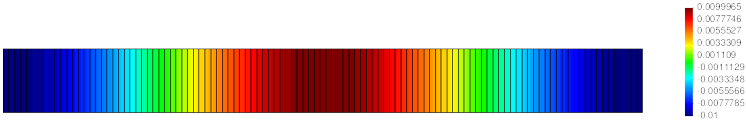


(a) Fluid mesh.



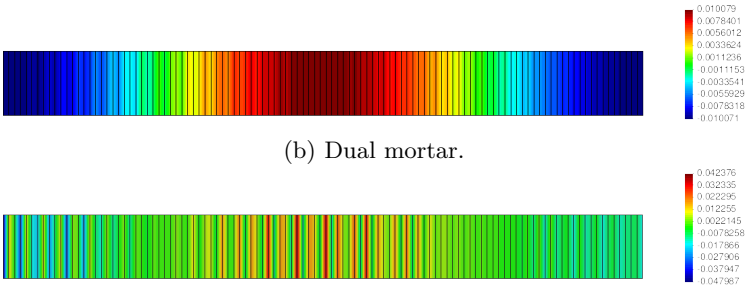
(b) Structure mesh.

Figure 4.24: Meshes of the rectangular surface including a coarser fluid mesh with 50 elements and a finer structure mesh with 111 elements.



(a) Standard mortar.

Figure 4.25: Traction field from conservative mapping of different algorithms.



(b) Dual mortar.

(c) Nearest element interpolation.

Figure 4.25: Traction field from conservative mapping of different algorithms.

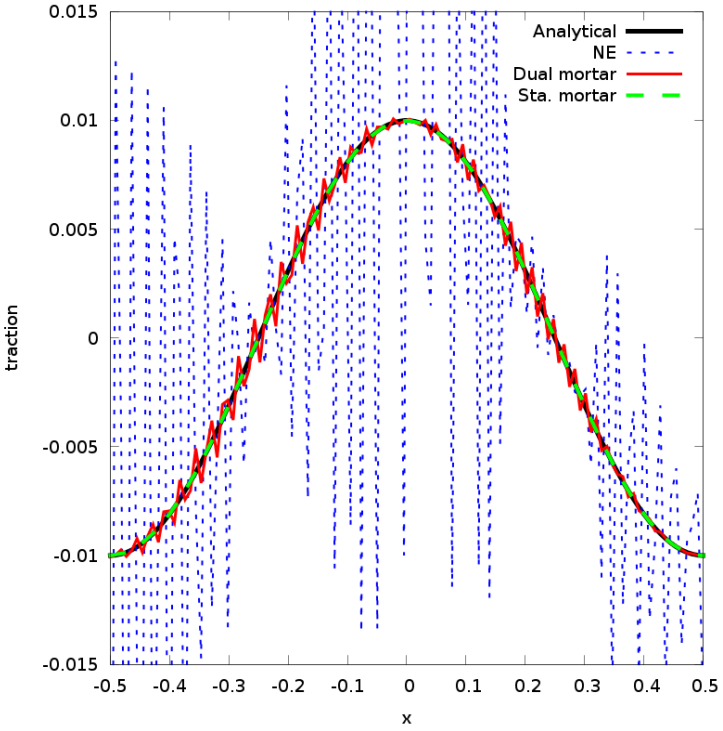
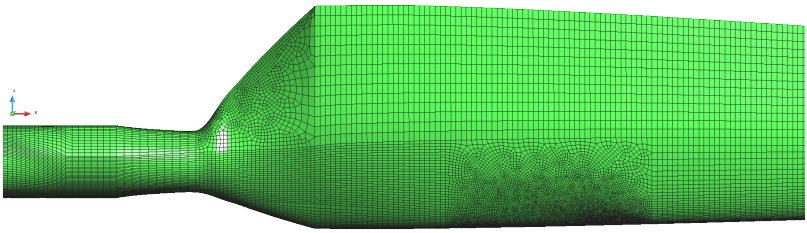


Figure 4.26: Traction along the length direction from conservative mapping with different algorithms. NE stands for nearest element interpolation.

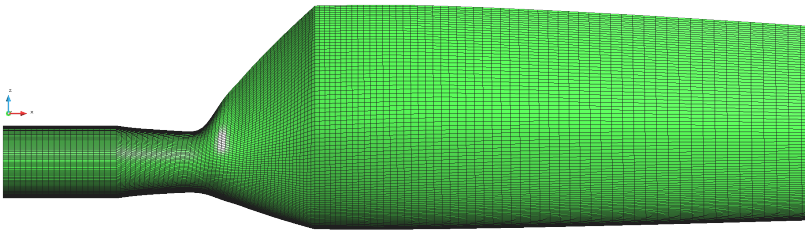
4.5.3 Wind Turbine Blade

In this section, the mapping on the blade surface of the NREL phase VI wind turbine is tested with different mapping algorithms. The displacement and traction fields on the blade surface are extracted from the FSI simulation presented in Section 6.2.1 and are applied in the following mapping test. The surface meshes from the structure and the fluid sides are shown in Fig. 4.27 and Fig. 4.28. It can be seen that the structure mesh is finer in the length direction at some part and the tip surface has curved edges.

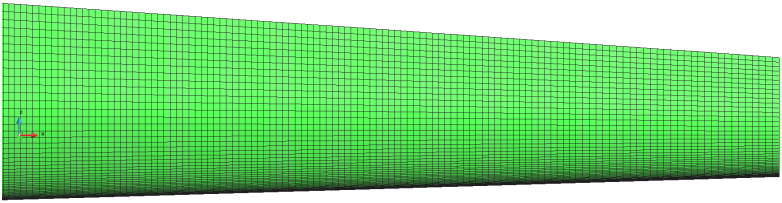
The displacement field on the structure mesh and the mapping results on the fluid mesh are shown in Fig. 4.29. It can be seen that all algorithms give good results. The enforcing consistency algorithm is used for the mortar methods to eliminate the error at the curved edge of the blade tip as shown in Fig. 4.30. Otherwise, the distorted fluid elements at the curved edges can result in failure of the fluid computation. The traction field on the fluid mesh and the mapped results on the structure mesh are shown in Fig. 4.31 (only the field in the inflow direction is drawn to emphasize the turbulent eddies). The conservative mapping with the nearest element interpolation gives highly oscillatory results and the same with the dual mortar method gives small oscillations superposed to the correct solution. Satisfactory results are obtained from the direct mapping with the nearest element interpolation and the conservative mapping with the standard mortar method. But force and energy are only approximately conserved in the former case.



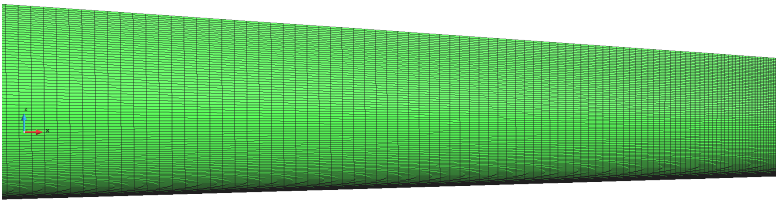
(a) Structure mesh part one.



(b) Fluid mesh part one.

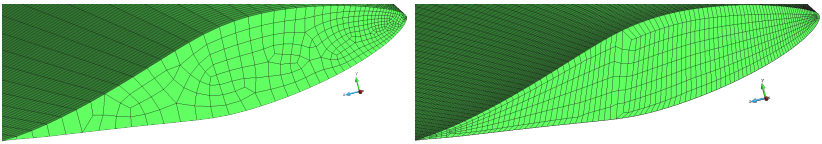


(c) Structure mesh part two.



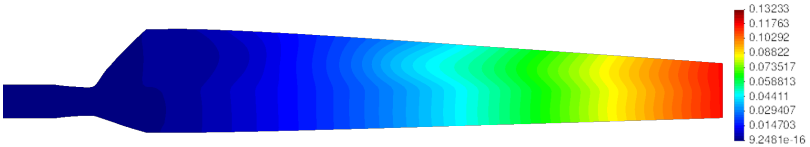
(d) Fluid mesh part two.

Figure 4.27: The surface meshes of the turbine blade. They are shown in two parts due to the high refinement. The structure surface mesh has 105,348 quadrilaterals and 1,776 triangles and the fluid surface mesh has 87,712 quadrilaterals and 4 triangles.

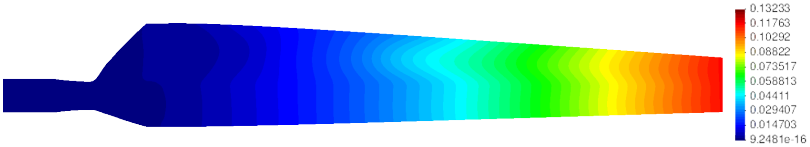


(a) From structure. (b) From fluid.

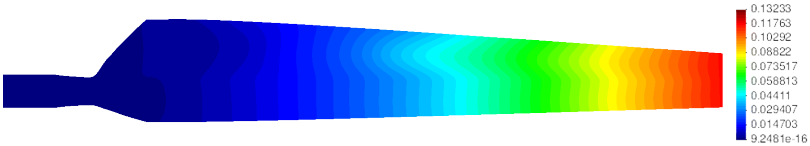
Figure 4.28: The meshes at the tip of the blade.



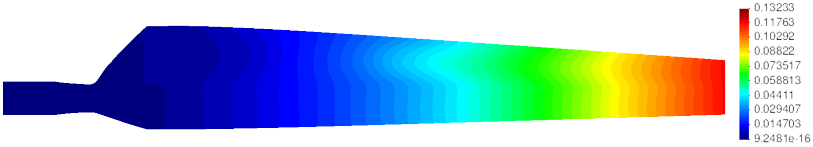
(a) Structure displacement field.



(b) Nearest element interpolation.

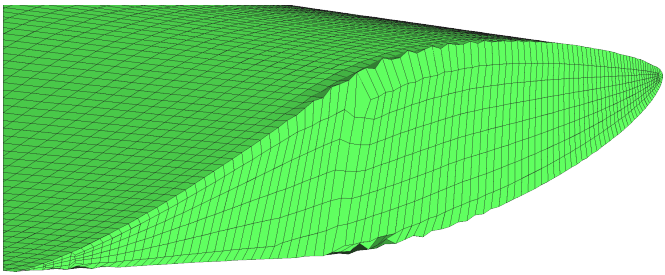


(c) Standard mortar.

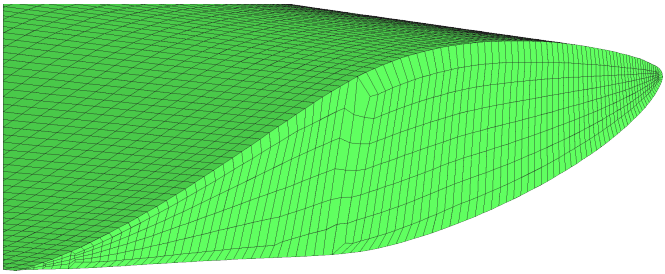


(d) Dual mortar.

Figure 4.29: Displacement field on the structure mesh and mapping results on the fluid mesh from direct mapping with different algorithms. The displacements are shown in magnitude.

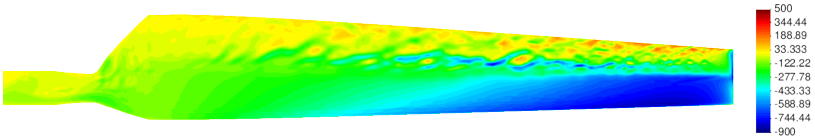


(a) Without enforcing consistency.

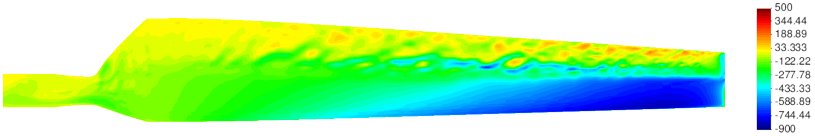


(b) With enforcing consistency.

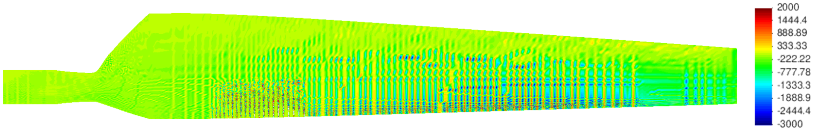
Figure 4.30: Deformed meshes resulting from standard mortar method at the tip. The dual mortar method gives very close results which are not additionally presented.



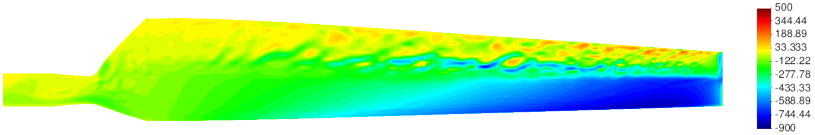
(a) Fluid traction field.



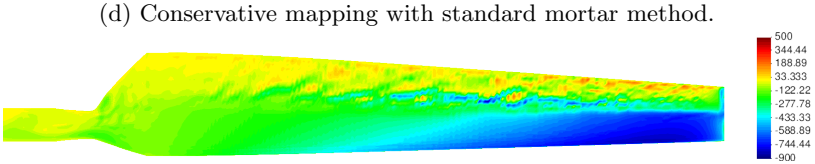
(b) Direct mapping with nearest element interpolation.



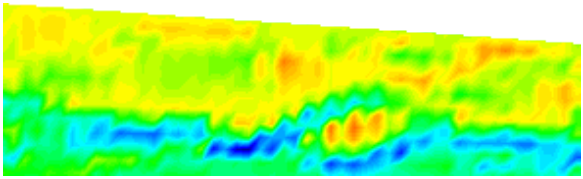
(c) Conservative mapping with nearest element interpolation.



(d) Conservative mapping with standard mortar method.



(e) Conservative mapping with dual mortar method.



(f) Zooming in to highlight the oscillations in (e). The unphysical oscillations are superposed to the turbulent eddies.

Figure 4.31: Traction field on the fluid mesh and the mapped results on the structure mesh. Only the component in the inflow direction is presented.

4.6 Summary

Three mapping algorithms dealing with surface meshes are investigated and compared, namely the nearest element interpolation, the standard mortar method and the dual mortar method. The mortar methods are inconsistent at curved edges where some fluid surface elements are not fully in contact with structure surface elements, due to the discrepancy between the integration domains from the two partitions. This problem is solved by the newly developed enforcing consistency approach where inter-/extrapolation is embedded into the mortar matrices. It is superior than the state of the art solutions using scaling because it can handle fully uncovered fluid elements. Both the analysis and the test examples have shown that the mortar methods are suitable for conservative mapping. But the dual mortar method can give slight oscillations where the structure mesh density is higher, although it is more efficient than the standard version. For nearest element interpolation, conservative traction mapping is less accurate and can give very high oscillations in traction; direct traction mapping is more accurate, but then the interface energy is only approximately conserved.

How to choose mapping algorithms in practice is suggested next. For the case that only the nearest element interpolation method is available, although it is less accurate and may bring large oscillations, but the accuracy of deformation is usually acceptable if a structure is stiff and insensitive to these locally appearing errors. However, the stress distribution close to the wet surface inside the structure is also less accurate and can have oscillations as a consequence. In contract, if structures are sensitive to the local errors such as soft membranes, or if a more accurate stress distribution close to the wet surface is pursued, then the direct traction mapping should be used, though there is a slight difference in the energy. For the case that the mortar algorithms are available, the dual mortar method can be applied in most problems. But the slight oscillations in the traction field should be distinguished from those caused by turbulent eddies. In the worst case, the standard mortar method can be used which performs stably well.

Chapter 5

Mapping with Beam Elements

A beam structure has one spatially dominant direction and therefore can be modeled and analyzed as an elastic curve which is discretized by 1D elements in FEM. 1D models are widely used in the aeroelastic analysis of beam-like structures such as wind turbine blades [62, 124], airplane wings [131, 108] or other composite beams [13], as a simplification of modeling these structures with solid or shell elements [105, 19, 69, 120]. Additional cross section analysis is required for modeling with 1D elements to extract the properties of cross sections [90, 16, 25]. In structural wind engineering, slender buildings and bridges are also modeled as 1D curves at the pre-development stage, to analyze the dynamical response to the wind loads [121, 67].

In FSI simulation of a beam structure, data have to be mapped between 1D beam elements and a fluid surface mesh. Different from the case that both meshes are surface meshes, the relation between the 1D elements and the surface mesh needs to be built. Since the 1D mesh usually has only a small number of elements, the ability to represent the rotation degrees of freedom (DOFs) is important for the mapping accuracy and the smoothness of the deformed surface. Both [2] and [30] apply radial basis function (RBF) model for the problem. The former uses linearized rotations as parameters inside the RBF model while the latter creates a new intermediate mesh so that mapping techniques for surface meshes can be reused. The two methods require a big number of 1D elements and are restricted to small rotations. However, geometrical nonlinearity should be considered when mapping on flexible beam structures that undergo large deformation and rotation. The structural analysis of geometrically nonlinear beam elements can be found in

[8]. The algorithm for the reconstruction of the deformed beam surface is inspired by the co-rotating formulation of nonlinear beam elements [81, 45, 87, 33].

This chapter focuses on deformation mapping which is also called *surface reconstruction*. A co-rotating algorithm which can recover large displacements and rotations is presented and evaluated in a comprehensive way. Since the algorithm is based on the assumption that cross sections remain rigid, it is required to relate a surface node with the cross-section passing through it. This is realized by an efficient algorithm which assumes that all cross sections are parallel. Structured and unstructured surface meshes at the fluid are handled in different ways with the consideration of efficiency. Moreover, the convergence behavior of the co-rotating algorithm is assessed also in comparison with a linear kinematics algorithm, which clearly demonstrates the limits of the latter approach that is frequently used in practice.

To handle linking rigid body motions and coordinates transformations, the idea of *forward kinematics* is used, i.e. a rigid body motion operator is defined to describe rigid body motion and coordinates transformation between different coordinate systems (also called frames). The operator is equivalent to the 4×4 *Denavit-Hartenberg* matrix, which condenses a rotation matrix and a translation vector together. Similar applications can be found in multibody system [118], robotics [31] and skeletal animation [106], where rigid elements connected by hinges are handled. It is different to the mapping problem discussed here while the structure body is elastic. The rigid body motion operator here helps to provide mathematical description of the mapping algorithms.

The details of the formulation and the implementation of the deformation mapping are presented in Section 5.1, including the determination of the cross section passing a surface node, the definition of a rigid body motion operator as well as the interpolation of the rotations and displacements at a given cross section with either the linearized or the co-rotating algorithm. Section 5.2 presents the load mapping briefly. The convergence behaviors of the linearized and the co-rotating algorithms are tested with analytically prescribed deformations, as shown in Section 5.3. Moreover, the smoothness and robustness of the co-rotating algorithm are demonstrated in Section 5.4, with examples of an airplane wing and a wind turbine blade under artificially large deformation. Finally, this chapter is concluded in Section 5.5.

5.1 Deformation Mapping

The deformation mapping is based on the assumption that the cross sections of a beam remains rigid so that the displacements of the fluid surface nodes can be determined from the rigid body motion (RBM) of the corresponding cross sections. It is accomplished in two steps: the cross section that passes through a given surface node is determined in the first step; the rotation and translation of the cross section are interpolated from the DOFs of a corresponding 1D element in the second step.

5.1.1 Determine Cross Section

A simple and efficient algorithm is implemented in this work which assumes that all cross sections are parallel. It is valid since a beam structure is straight or only slightly curved in most cases, i.e. the tangential direction of the beam axis does not change a lot. With the normal of the parallel cross sections $\mathbf{n}_{cs}$, the plane of the cross section passing a surface node Q can be computed as shown in Fig. 5.1. The intersection P between the plane and a beam curve element is defined as the center of the cross section. It is easily achieved in the implementation: the nodes on both meshes are projected to the line defined by $\mathbf{n}_{cs}$, then P is equal to the new coordinate of Q in the direction of $\mathbf{n}_{cs}$. Since the rotation is defined around P, the beam curve should be located as close to the elastic or the shear center as possible. For the case that both centers do not coincide, the beam curve should be placed properly so that the numerical error with respect to the kinematic theory is minimal.

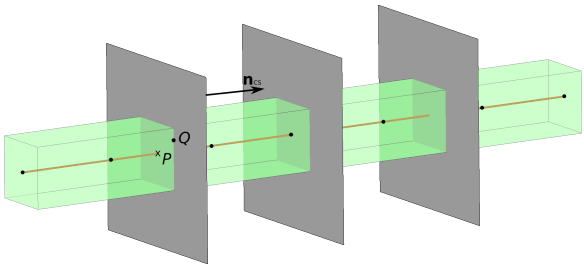


Figure 5.1: The planes of cross sections are assumed parallel. The cross section passing a surface point Q intersects with an 1D element at P, which is defined as the center of the cross section.

When surface nodes locate outside an outermost cross section plane which passes one end of the beam curve as in Fig. 5.2, their motions will be assigned according to the DOFs at the curve end, i.e. these nodes move together as a rigid body.

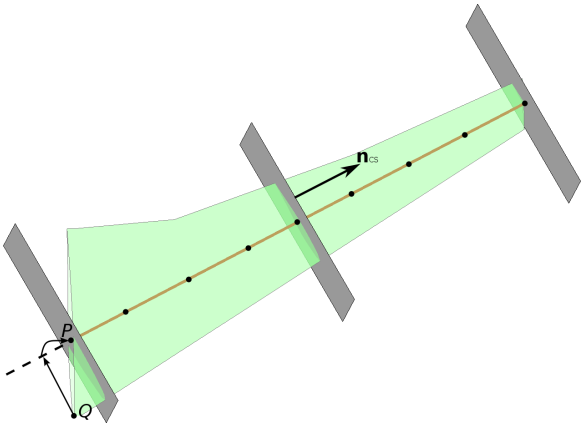
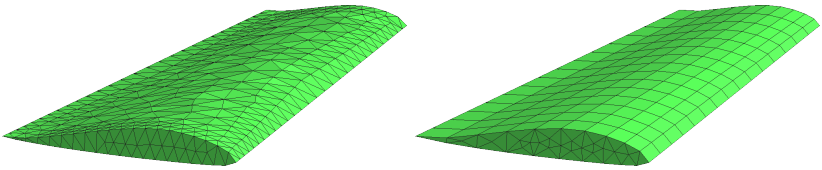


Figure 5.2: Surface nodes can locate outside the leftmost and the rightmost planes.

The efficiency of the algorithm can be further improved if the surface mesh is a structured mesh, see Fig. 5.3b. In this case, the surface nodes are projected as before to the line of $\mathbf{n}_{cs}$, then they are sorted by the new coordinate in the $\mathbf{n}_{cs}$ direction. A structured mesh usually has the same number of nodes n_r for each row along the beam length direction, therefore every n_r nodes are grouped together according to the positions after sorting. The nodes at the tip or the root of the beam surface should be separately grouped if the number of them is different than n_r. The nodes belonging to the same group are regarded as also belonging to the same cross section and are assigned with the same RBM. The preprocessing usually cannot guarantee that the nodes of the same group have the same coordinate in the $\mathbf{n}_{cs}$ direction, so the average of all the coordinates is chosen as the center of the cross section.

The simple and efficient algorithm above relates each surface node to a point on an 1D element. Some remarks regarding its accuracy and usage are given below:

- There is numerical error if the identification of the center is wrong, e.g. in case that the some 1D elements do not locate exactly



(a) unstructured (b) structured except at the root

Figure 5.3: Two meshes of the same surface. The cross section is separately computed for each node on the unstructured mesh, while the nodes on the structured mesh can be divided into rows along the length direction, and the nodes of the same row are regarded as on the same cross section.

at the beam axis, or the elastic center and the shear center do not coincide. As a result, the translation of the cross section is accurate, while the rotation has error, i.e. the rotation radii are wrong though the orientation of the cross section after rotating is still correct. The overall deformed shape is satisfactory but some cross sections are shifted around the correct location.

- There is also numerical error if the beam is curved, since the parallel cross sections are not always orthogonal to the beam axis. This will lead to error in the shape details, e.g. wrong distribution of angle of attack along an airwing. This is why only slightly curved beams are allowed as mentioned before.

- The two error sources above can also lead to numerical error in load which is computed in a energy conjugated way. But similarly, the error can be regarded as a local effect which is bounded.

- The algorithm can also handle connected beams. An example of the latter case is illustrated in Fig. 5.4, where two beams are connected by a rigid joint. This serves as a basic case for a topology of multiple beams. To solve the problem, partition b is modeled as infinitely rigid, which does not participate in the mapping. The mapping is performed in two groups: between a and $A \cup B$ as well as between c and C. Surface nodes of partition B will follow the rigid body motion defined on the top end of partition a.

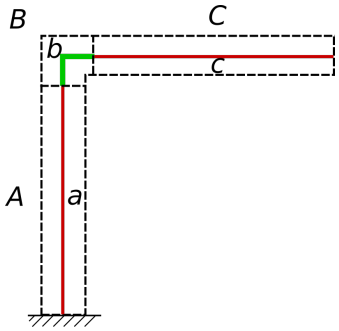


Figure 5.4: Mapping between beams connected by a rigid joint. The three partitions of the beam elements are denoted by small letters while those of the surface elements are denoted by capital letters.

5.1.2 Rigid Body Motion and Coordinates Transformation

The RBM of a cross section is naturally defined in a coordinate system whose origin is the center, since a rotation is attached to a center. But the motion has to be finally transformed to the global coordinate system where the surface nodes are defined. In the following, an operator is defined which helps to describe RBM and coordinates transformation.

Definition of rigid body motion operator

In this work, the RBM operator $\Phi : R^3 \rightarrow R^3$ based on a certain coordinate system is defined as:

$$\Phi(\mathbf{x}) = \mathbf{R}\mathbf{x} + \mathbf{t}, \tag{5.1}$$

where $\mathbf{R}$ is a rotation matrix (3×3), $\mathbf{t}$ is a translation vector and $\mathbf{x}$ are the coordinates of a point.

The inverse of $\Phi(\mathbf{x})$ is also an RBM operator:

$$\Phi^{-1}(\mathbf{x}) = \mathbf{R}^{-1}(\mathbf{x} - \mathbf{t}) = \mathbf{R}^T\mathbf{x} - \mathbf{R}^T\mathbf{t}. \tag{5.2}$$

$\mathbf{R}^{-1} = \mathbf{R}^T$ since $\mathbf{R}$ is an orthogonal matrix. Assume two RBMs are respectively defined as $\Phi_1(\mathbf{x}) = \mathbf{R}_1\mathbf{x} + \mathbf{t}_1$ and $\Phi_2(\mathbf{x}) = \mathbf{R}_2\mathbf{x} + \mathbf{t}_2$, the total motion of doing Φ_1 and Φ_2 in a sequence results in a composite RBM operator:

$$\Phi(\mathbf{x}) = \Phi_2 \circ \Phi_1(\mathbf{x}) = \mathbf{R}_2(\mathbf{R}_1\mathbf{x} + \mathbf{t}_1) + \mathbf{t}_2 = \mathbf{R}_2\mathbf{R}_1\mathbf{x} + (\mathbf{R}_2\mathbf{t}_1 + \mathbf{t}_2). \tag{5.3}$$

Note that only RBMs defined on the same coordinate system can be combined.

The RBM operator can also be used for the transformation of coordinates between different coordinate systems. Assume a coordinate system B is defined on another coordinate system A, and $\mathbf{e}_x^A$, $\mathbf{e}_y^A$ and $\mathbf{e}_z^A$ are the orthonormal base vectors of A which are equal to $(1,0,0)^T$, $(0,1,0)^T$ and $(0,0,1)^T$ respectively, while $\mathbf{e}_x^B$, $\mathbf{e}_y^B$ and $\mathbf{e}_z^B$ are the orthonormal base vectors of B defined with respect to $\mathbf{e}_x^A$, $\mathbf{e}_y^A$ and $\mathbf{e}_z^A$. The rotation from $\begin{bmatrix}\mathbf{e}_x^A & \mathbf{e}_y^A & \mathbf{e}_z^A\end{bmatrix}$ (a matrix constructed from the three vectors) to $\begin{bmatrix}\mathbf{e}_x^B & \mathbf{e}_y^B & \mathbf{e}_z^B\end{bmatrix}$ defined on A can be obtained as

$$\mathbf{R}_{A\to B}\begin{bmatrix}\mathbf{e}_x^A & \mathbf{e}_y^A & \mathbf{e}_z^A\end{bmatrix} = \begin{bmatrix}\mathbf{e}_x^B & \mathbf{e}_y^B & \mathbf{e}_z^B\end{bmatrix} \Longrightarrow \mathbf{R}_{A\to B} = \begin{bmatrix}\mathbf{e}_x^B & \mathbf{e}_y^B & \mathbf{e}_z^B\end{bmatrix}. \tag{5.4}$$

Moreover, assume that $\mathbf{t}_{A\to B}$ defined on A denotes the translation from the origin of A to that of B, if a point has coordinates $\mathbf{x}_A$ in A and and coordinates $\mathbf{x}_B$ in B, the transformation between them can be derived as

$$\begin{bmatrix}\mathbf{e}_x^A & \mathbf{e}_y^A & \mathbf{e}_z^A\end{bmatrix}\mathbf{x}_A = \begin{bmatrix}\mathbf{e}_x^B & \mathbf{e}_y^B & \mathbf{e}_z^B\end{bmatrix}\mathbf{x}_B + \begin{bmatrix}\mathbf{e}_x^A & \mathbf{e}_y^A & \mathbf{e}_z^A\end{bmatrix}\mathbf{t}_{A\to B} \tag{5.5}$$

$$= \mathbf{R}_{A\to B}\mathbf{x}_B + \mathbf{t}_{A\to B}, \tag{5.6}$$

which can be rewritten by an RBM operator as

$$\mathbf{x}_A = \Phi_{A\to B}(\mathbf{x}_B) = \mathbf{R}_{A\to B}\mathbf{x}_B + \mathbf{t}_{A\to B}. \tag{5.7}$$

The inverse transformation can be derived as

$$\mathbf{x}_B = \Phi_{A\to B}^{-1}(\mathbf{x}_A). \tag{5.8}$$

What is to be emphasized is that $\Phi_{A\to B}$ as well as its components $\mathbf{R}_{A\to B}$ and $\mathbf{t}_{A\to B}$ are defined on A. The notations will be kept as a style for coordinates transformation in the following. Another remark is that the transformation here can contain translation which is different from the term in the FEM context where it contains only rotation.

Transformation of rigid body motion

Fig. 5.5 shows three coordinate systems and the transformations between them. They are:

- The global coordinate system (GCS) with axes $X-Y-Z$ and origin O.

- The element coordinate system (ECS) with axes $x - y - z$ and the same origin O as the GCS. Each 1D element has its own ECS. The x axis is parallel to the element and the other two axes can be defined arbitrarily following the orthogonality. (It is chosen here that the y axis locates in the $X - Y$ plane and the z axis is obtained through the right hand rule.) The rotation from the GCS to the ECS is defined by $\mathbf{R}_{G\to E}$.

- The section coordinate system (SCS). Each cross section has its own SCS. Its axes are parallel to those of the ECS and its origin lies at the center of the cross section P. The translation between the ECS and the SCS is defined by $\mathbf{t}_{E\to P}$, which is the displacement from O to P.

Given Φ_P as the RBM of a cross section defined on the SCS, it can be transformed to the GCS as

$$\Phi_G = \mathbf{R}_{G\to E} \circ \mathbf{t}_{E\to P} \circ \Phi_P \circ \mathbf{t}_{E\to P}^{-1} \circ \mathbf{R}_{G\to E}^{-1}, \tag{5.9}$$

where $\mathbf{R}$ or $\mathbf{t}$ is used instead of Φ in order to represent pure rotation or translation. (5.9) should be interpreted from right to left in a way that a point is transformed through two steps from the GCS to the SCS, then an RBM is applied on it, and finally its new position is transformed back to the GCS through the inverse operations of the former two transformation steps. With Φ_G the displacement vector of a surface node $\mathbf{x}$ (also defined on the GCS) on the cross section can be computed as

$$\mathbf{u} = \Phi_G(\mathbf{x}) - \mathbf{x}, \tag{5.10}$$

which is the equation of deformation mapping.

Except Φ_P, the other terms in the expression of Φ_G in (5.9) can be computed from geometrical relations. The rotation and translation in Φ_P can be interpolated from the DOFs of a linear or nonlinear 1D element, which will be introduced in the following.

5.1.3 Interpolation of Rigid Body Motion

Rotation can be described by an angle θ around a unit vector $\mathbf{n}$, or a rotation vector $\vec{\theta} = \theta\mathbf{n} = (\theta_x, \theta_y, \theta_z)^T$. The relation between the rotation vector and its equivalent rotation matrix is [81]

$$\mathbf{R} = \cos\theta\mathbf{I} + \sin\theta\hat{\mathbf{n}} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T, \tag{5.11}$$

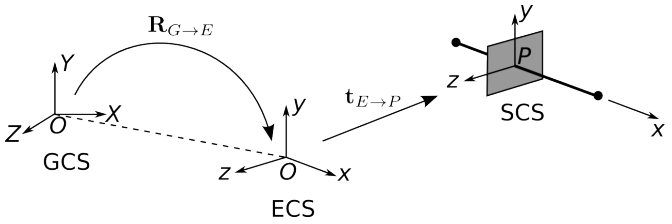


Figure 5.5: The global coordinate system (GCS), the element coordinate system (ECS), the section coordinate system (SCS) and the transformation between them.

where $\mathbf{I}$ is the identity matrix and $\hat{\mathbf{n}}$ is a 3×3 matrix as a compact form of the cross product between $\mathbf{n}$ and an arbitrary vector $\mathbf{x}$:

$$\hat{\mathbf{n}}\mathbf{x} = \mathbf{n}\times\mathbf{x}.$$

For a small rotation (5.11) can be linearized as

$$\mathbf{R} \approx \mathbf{I} + \theta\hat{\mathbf{n}}. \tag{5.12}$$

In this case, the entries in the rotation vector can be regarded as three separate rotations around the axes of the coordinate system.

The 1D beam element in this work has six DOFs on each node including a displacement vector and a rotation vector. Before interpolation, the displacement vector $(v_{Xi}, v_{Yi}, v_{Zi})^T$ and the rotation vector $(\theta_{Xi}, \theta_{Yi}, \theta_{Zi})^T$ defined on the GCS are transformed to the ECS which results in $(v_{xi}, v_{yi}, v_{zi})^T$ and $(\theta_{xi}, \theta_{yi}, \theta_{zi})^T$, where $i = 1$ or 2 being the node index (see Fig. 5.6). The displacement vector can be transformed as

$$(v_{xi}, v_{yi}, v_{zi})^T = \mathbf{R}_{E\to G}(v_{Xi}, v_{Yi}, v_{Zi})^T. \tag{5.13}$$

For small rotations, the rotation vector can be transformed in the same way:

$$(\theta_{xi}, \theta_{yi}, \theta_{zi})^T = \mathbf{R}_{E\to G}(\theta_{Xi}, \theta_{Yi}, \theta_{Zi})^T. \tag{5.14}$$

For large rotations, the equivalent rotation matrix of $(\theta_{Xi},\ \theta_{Yi},\ \theta_{Zi})^T$ denoted by $\mathbf{R}_{Gi}$ is computed using (5.11) and transformed to the ECS as

$$\mathbf{R}_{Ei} = \mathbf{R}_{G\to E}^{-1}\mathbf{R}_{Gi}\mathbf{R}_{G\to E}, \tag{5.15}$$

and then $\mathbf{R}_{Ei}$ is converted to $(\theta_{xi},\ \theta_{yi},\ \theta_{zi})^T$ by solving (5.11).

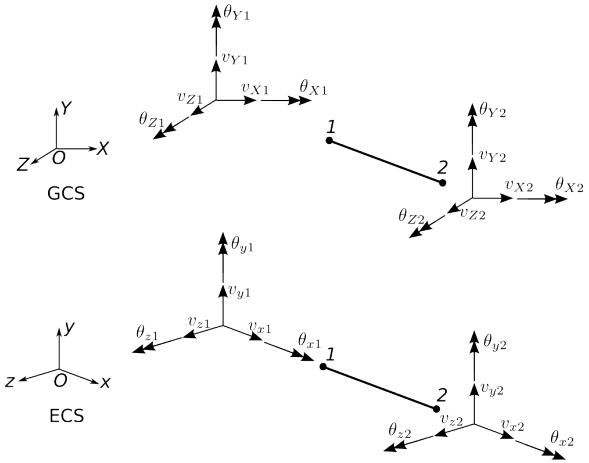


Figure 5.6: DOFs of a beam element corresponding to different coordinate systems.

Linearized Algorithm

The linearized algorithm in the following uses shape functions of a two-node Bernoulli beam element where the kinematic relationship is linearized. The details of the element formulation can be found in [141]. Assume P has parametric coordinate ξ on the 1D element as

$$\xi = \frac{2(x - x_1)}{l} - 1, \tag{5.16}$$

where x and x_1 are respectively the x-coordinate of P and node 1 on the ECS, and l is the length of the element which can be regarded as unchanged for a linear element ($l = x_2 - x_1$ where x_2 is the x-coordinate of node 2 on the ECS).

Define $(v_x, v_y, v_z)^T$ and $(\theta_x,\ \theta_y,\ \theta_z)^T$ as the displacement vector and the rotation vector on P respectively. Then v_x and θ_x can be interpolated with linear shape functions:

$$v_x = \begin{pmatrix} N_1^l & N_2^l \end{pmatrix} \begin{pmatrix} v_{x1} & v_{x2} \end{pmatrix}^T, \tag{5.17a}$$

$$\theta_x = \begin{pmatrix} N_1^l & N_2^l \end{pmatrix} \begin{pmatrix} \theta_{x1} & \theta_{x2} \end{pmatrix}^T, \tag{5.17b}$$

where

$$N_1^l = \frac{1}{2}(1 - \xi), \quad N_2^l = \frac{1}{2}(1 + \xi). \tag{5.18}$$

While v_y, θ_z, v_z and θ_y can be interpolated with Hermitian cubic shape functions:

$$v_y = \begin{pmatrix} N_1^v & N_1^\theta & N_2^v & N_2^\theta \end{pmatrix} \begin{pmatrix} v_{y1} & \theta_{z1} & v_{y2} & \theta_{z2} \end{pmatrix}^T, \tag{5.19a}$$

$$\theta_z = v_y' = \begin{pmatrix} N_1^{v\prime} & N_1^{\theta\prime} & N_2^{v\prime} & N_2^{\theta\prime} \end{pmatrix} \begin{pmatrix} v_{y1} & \theta_{z1} & v_{y2} & \theta_{z2} \end{pmatrix}^T, \tag{5.19b}$$

$$v_z = \begin{pmatrix} N_1^v & N_1^\theta & N_2^v & N_2^\theta \end{pmatrix} \begin{pmatrix} v_{z1} & -\theta_{y1} & v_{z2} & -\theta_{y2} \end{pmatrix}^T, \tag{5.19c}$$

$$\theta_y = -v_z' = \begin{pmatrix} N_1^{v\prime} & N_1^{\theta\prime} & N_2^{v\prime} & N_2^{\theta\prime} \end{pmatrix} \begin{pmatrix} -v_{z1} & \theta_{y1} & -v_{z2} & \theta_{y2} \end{pmatrix}^T, \tag{5.19d}$$

where

$$N_1^v = \frac{1}{4}(1-\xi)^2(2+\xi), \qquad N_1^\theta = \frac{1}{8}l(1-\xi)^2(1+\xi),$$
$$N_2^v = \frac{1}{4}(1+\xi)^2(2-\xi), \qquad N_2^\theta = -\frac{1}{8}l(1+\xi)^2(1-\xi), \tag{5.20a}$$

$$N_1^{v\prime} = -\frac{3}{2l}(1-\xi)(1+\xi), \qquad N_1^{\theta\prime} = -\frac{1}{4}(1-\xi)(1+3\xi),$$
$$N_2^{v\prime} = \frac{3}{2l}(1+\xi)(1-\xi), \qquad N_2^{\theta\prime} = -\frac{1}{4}(1+\xi)(1-3\xi). \tag{5.20b}$$

After interpolation, Φ_P in (5.9) can be constructed from $(v_x,\, v_y,\, v_z)^T$ and $(\theta_x,\, \theta_y,\, \theta_z)^T$.

Co-rotating Algorithm

Since a large rotation cannot be decomposed into three individual rotations, the linearized algorithm cannot be used on beam elements with large rotations.

The solution in this work is inspired by the idea of co-rotating beam elements, where the motion of an 1D element is decomposed into an RBM and a linear deformation with six natural modes including elongation, twist, symmetric and anti-symmetric bendings as shown in Fig. 5.7.

The newly developed co-rotating algorithm for mapping contains three steps as shown in Fig. 5.8:

1. Displace the end nodes according to $(v_{xi}, v_{yi}, v_{zi})^T$. This process also includes the elongation of the element.

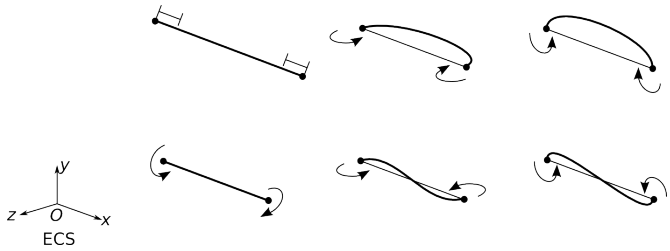


Figure 5.7: Natural deformation modes of a beam element. From top to bottom and left to right they are sequentially elongation, twist, symmetric bending in $x-z$ plane, anti-symmetric bending in $x-z$ plane, symmetric bending in $x-y$ plane and anti-symmetric bending in $x-y$ plane. (This figure is an adapted presentation of the one in [81].)

2. Rigid body rotation around the axis (length direction) of the element.

3. Twist and bend the element. The RBM of the element is already carried out in the first two steps. The third step includes only the twist mode and the symmetric and anti-symmetric bending modes where rotations can be linearized.

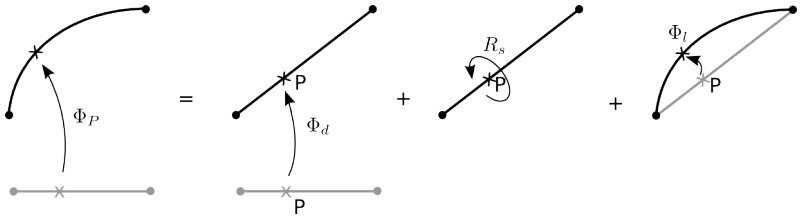


Figure 5.8: The rigid body motion of a cross section can be decomposed into three steps. The cross represents the cross section.

The SCS also moves with the RBM of the element in the first step in order to perform rotations in the next two steps. The motion should be transformed back to the original SCS after the third step. Define Φ_d, $\mathbf{R}_s$ and Φ_l as the RBM operators for the three steps as shown in Fig. 5.8, then Φ_P in (5.9) can be computed as:

$$\begin{aligned}\Phi_P &= \Phi_d \circ \Phi_l \circ \mathbf{R}_s \circ \Phi_d^{-1} \circ \Phi_d \\ &= \Phi_d \circ \Phi_l \circ \mathbf{R}_s \end{aligned} \tag{5.21}$$

It shows that the total motion can also be interpreted as the combination of the three motions in a different order without any transformation between coordinate systems. How the individual operators are computed is presented in the following.

The translation vector t_d in Φ_d is computed by interpolating $(v_{xi}, v_{yi}, v_{zi})^T$ on P using the linear shape functions in (5.18). The rotation in Φ_d is computed as [81]

$$\mathbf{R}_d = (\mathbf{I} - 2\mathbf{n}\mathbf{n}^T)[-\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z], \tag{5.22}$$

with

$$\mathbf{n} = (\mathbf{e}_x + \mathbf{e}_x^d)/\left|\mathbf{e}_x + \mathbf{e}_x^d\right|, \tag{5.23}$$

where $\mathbf{e}_x$, $\mathbf{e}_y$ and $\mathbf{e}_z$ are the base vectors of the ECS which are equal to $(1,0,0)^T$, $(0,1,0)^T$ and $(0,0,1)^T$ respectively, and $\mathbf{e}_x^d$ is the normalized vector in the new element length direction. So far Φ_d is obtained.

The RBMs on both end nodes can be constructed from $(v_{xi}, v_{yi}, v_{zi})^T$ and $(\theta_{xi}, \theta_{yi}, \theta_{zi})^T$ which are denoted by Φ_{Pi}. Since the RBM of the first step on both end nodes Φ_{di} can be computed with $(v_{xi}, v_{yi}, v_{zi})^T$ and $\mathbf{R}_d$, the remaining motion can be computed by $\Phi_{li} \circ \mathbf{R}_s = \Phi_{di}^{-1} \circ \Phi_{Pi}$. Φ_{li} contains no translation since both end nodes are already displaced in the first step, so $\Phi_{li} = \mathbf{R}_{li}$ and therefore $\Phi_{li} \circ \mathbf{R}_s = \mathbf{R}_{li}\mathbf{R}_s$. $\mathbf{R}_{li}\mathbf{R}_s$ contains small rotations from bending and twist and may also contain a big axial rigid rotation. The axial rotation needs to be extracted from the combined rotation $\mathbf{R}_{li}\mathbf{R}_s$. After computing $(\theta_{xi}^s, \theta_{yi}^s, \theta_{zi}^s)^T$ as the equivalent rotation vector of $\mathbf{R}_{li}\mathbf{R}_s$, the angle of the axial rotation can be approximated as

$$\theta_s = \frac{1}{2}(\theta_{x1}^s + \theta_{x2}^s). \tag{5.24}$$

The approximation is valid since θ_{xi}^s is the dominant entry in the rotation vector. Besides, an approximation of θ_s is enough because the purpose is to deduct the large axial rotation so that the remaining rotations can be linearized. So far, $\mathbf{R}_s$ is also obtained which can be constructed from θ_s and $\mathbf{e}_x$ using (5.11) with $\mathbf{n} = \mathbf{e}_x$.

After deducting the axial rotation the remaining rotation on the end nodes can be computed as

$$\mathbf{R}_{li} = \Phi_{di}^{-1} \circ \Phi_{Pi} \circ \mathbf{R}_s^{-1}, \tag{5.25}$$

which contains twist and bending. The equivalent rotation vectors on the end nodes $(\theta_{xi}^l, \theta_{yi}^l, \theta_{zi}^l)^T$ can be computed from $\mathbf{R}_{li}$ by solving (5.11), and the displacement vectors on them have $(v_{xi}^l, v_{yi}^l, v_{zi}^l)^T =$

$(0,0,0)^T$. Both vectors are used to interpolate the displacements and rotations on P using the linearized algorithm introduced in Section 5.1.3. Then Φ_l in (5.21) can be constructed from the results of interpolation. So far, Φ_d, $\mathbf{R}_s$ and Φ_l in (5.21) are all obtained.

5.2 Load Mapping

The forces on the surface nodes are used to compute the forces and moments on the curve nodes in the load mapping. The force vector $\mathbf{f}_Q$ on a surface node Q is transferred to P first, as illustrated in Fig. 5.9. This results in a force vector $\mathbf{f}_P$ and a moment vector $\mathbf{m}_P$ on P with

$$\mathbf{f}_P = \mathbf{f}_Q, \tag{5.26a}$$

$$\mathbf{m}_P = \overrightarrow{PQ} \times \mathbf{f}_Q, \tag{5.26b}$$

where $\overrightarrow{PQ}$ is the difference vector from P to Q in the deformed configuration.

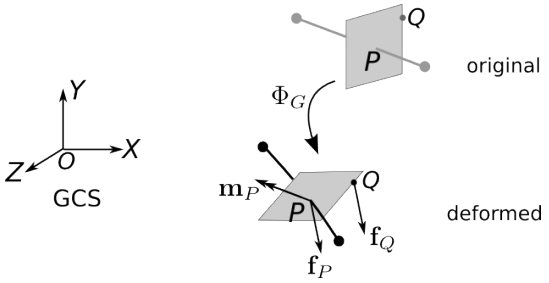


Figure 5.9: Transfer a nodal force to the center of the cross section.

The next task is to transfer $\mathbf{f}_P$ and $\mathbf{m}_P$ to the end nodes of the corresponding 1D element. It is based on the *conservative mapping* idea that is presented in Chapter 4. It also corresponds to the computing of *consistent nodal forces* and *moments* in beam FEM ([141, 78]), i.e. the forces and moments are distributed to the end nodes with the same shape functions for the conjugated displacements and rotations. For linear 1D elements, $\mathbf{f}_P$ and $\mathbf{m}_P$ are firstly transformed to the ECS, resulting in $(f_x^P, f_y^P, f_z^P)^T$ and $(m_x^P, m_y^P, m_z^P)^T$. With (5.17) and (5.19)

conservative mapping gives

$$f_x^P v_x = \begin{pmatrix} f_x^P N_1^l & f_x^P N_2^l \end{pmatrix} \begin{pmatrix} v_{x1} & v_{x2} \end{pmatrix}^T, \tag{5.27a}$$

$$m_x^P \theta_x = \begin{pmatrix} m_x^P N_1^l & m_x^P N_2^l \end{pmatrix} \begin{pmatrix} \theta_{x1} & \theta_{x2} \end{pmatrix}^T, \tag{5.27b}$$

$$f_y^P v_y = \begin{pmatrix} f_y^P N_1^v & f_y^P N_1^\theta & f_y^P N_2^v & f_y^P N_2^\theta \end{pmatrix} \begin{pmatrix} v_{y1} & \theta_{z1} & v_{y2} & \theta_{z2} \end{pmatrix}^T, \tag{5.27c}$$

$$m_z^P \theta_z = \begin{pmatrix} m_z^P N_1^{v\prime} & m_z^P N_1^{\theta\prime} & m_z^P N_2^{v\prime} & m_z^P N_2^{\theta\prime} \end{pmatrix} \begin{pmatrix} v_{y1} & \theta_{z1} & v_{y2} & \theta_{z2} \end{pmatrix}^T, \tag{5.27d}$$

$$f_z^P v_z = \begin{pmatrix} f_z^P N_1^v & f_z^P N_1^\theta & f_z^P N_2^v & f_z^P N_2^\theta \end{pmatrix} \begin{pmatrix} v_{z1} & -\theta_{y1} & v_{z2} & -\theta_{y2} \end{pmatrix}^T, \tag{5.27e}$$

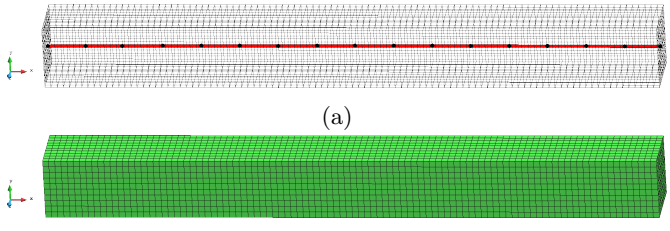
$$m_y^P \theta_y = \begin{pmatrix} m_y^P N_1^{v\prime} & m_y^P N_1^{\theta\prime} & m_y^P N_2^{v\prime} & m_y^P N_2^{\theta\prime} \end{pmatrix} \begin{pmatrix} -v_{z1} & \theta_{y1} & -v_{z2} & \theta_{y2} \end{pmatrix}^T. \tag{5.27f}$$

For nonlinear 1D elements, the increments of DOFs $(\mathrm{d}\, v_{xi}, \mathrm{d}\, v_{yi}, \mathrm{d}\, v_{zi})^T$ and $(\mathrm{d}\, \theta_{xi}, \mathrm{d}\, \theta_{yi}, \mathrm{d}\, \theta_{zi})^T$ are defined as aligned with the element in the co-rotating configuration, which corresponds to the second step in Fig. 5.8. Since the increments are small, the same shape functions used for the DOFs in (5.17) and (5.19) can be used for them. To apply conservative mapping, $\mathbf{f}_P$ and $\mathbf{m}_P$ are firstly transformed to the co-rotating configuration, and since now they are aligned with the conjugated increments of DOFs, they can be also distributed with (5.27). The process above is repeated for each surface node, and the forces and moments are accumulated to obtain the final load on the 1D elements.

5.3 Convergence Tests

A beam with constant square cross section is used in the convergence tests of deformation mapping. It has length $l = 10$ in the x direction and constant square cross section in the $y - z$ plane with the size $a = 1 \times 1$ (x, y and z represent the axes of the GCS from here on). The surface of the beam is meshed equally with 100 elements in x direction and 10 elements in both y and z directions. The beam curve locates at the axis of the beam. It is discretized equally into 2^k elements where $k \in \{1, 2, 3, 4, 5, 6\}$. The meshes are shown in Fig. 5.10. The DOFs of the 1D elements are generated from analytical functions and are mapped

to the beam surface mesh with both the linearized and the co-rotating algorithms. The convergence behaviors of both algorithms are tested.



(a)

(b)

Figure 5.10: The curve and surface meshes of a beam with square cross section. The surface mesh is displayed with two different styles.

5.3.1 Bending

The beam curve is clamped on the left end and bent into an arc in the $x-y$ plane without changing its length. The analytical function of the arc can be obtained as

$$y = -r\left(1 - \sqrt{1 - \left(\frac{x}{r}\right)^2}\right) \quad \text{with } 0 < x < r\sin(\alpha), \tag{5.28}$$

where α is the rotation angle (slope) at the right end and $r = \frac{l}{\alpha}$ is the radius of the arc. The displacements and rotations of an arbitrary point x on the beam axis can be derived:

$$\begin{aligned} \theta_z &= -\frac{x}{l}\alpha, \\ u_x &= r\sin(-\theta_z) - x, \\ u_y &= -(r - r\cos(-\theta_z)) \quad \text{with } 0 < x < l. \end{aligned} \tag{5.29}$$

Three test cases with $\alpha = 20°$, $40°$ and $60°$ are performed for which the arcs are plotted in Fig. 5.11. The DOFs on the beam elements are computed according to (5.29) and are mapped to the beam surface nodes with both the linearized and co-rotating algorithms. The reference displacements on the surface nodes are computed based on the RBMs defined by (5.29). The error of the displacements in the L2 norm

is defined as

$$e = \sqrt{\frac{\sum_{i=1}^{n_s \times 3} (U_i - U_i')^2}{n_s \times 3}}, \tag{5.30}$$

where U contains the displacement vectors on all surface nodes resulting from mapping and U' contains the reference values. The error with respect to different refinement levels of the beam curve is shown in Fig. 5.12. The co-rotating algorithm gives smaller error and converges with an order of 2.08. Theoretically interpolations with cubic functions give fourth order accuracy in displacement and third order accuracy in rotation. The reduction of the convergence order to 2.08 is due to the fact that the derivatives are approximated by the rotation angles instead of the tangent functions of them. The deformed surface meshes with $k = 4$ are shown in Fig. 5.13 and Fig. 5.14.

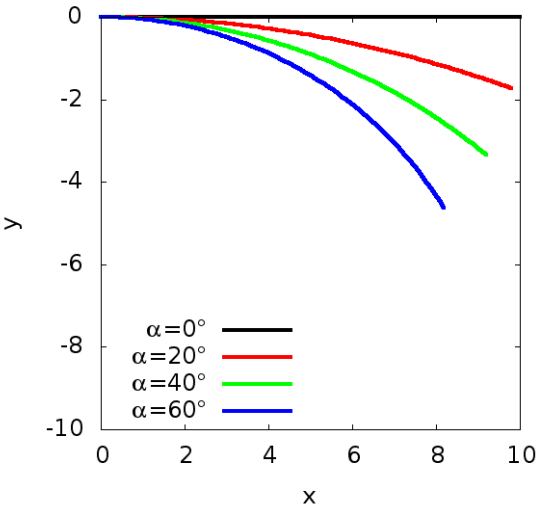


Figure 5.11: Plot of the function in (5.28) with $l = 10$ and different values of α.

5.3.2 Bending and Twist

The beam is twisted first and then bent with the same analytical function as in Section 5.3.1. The twist angle is linearly distributed along

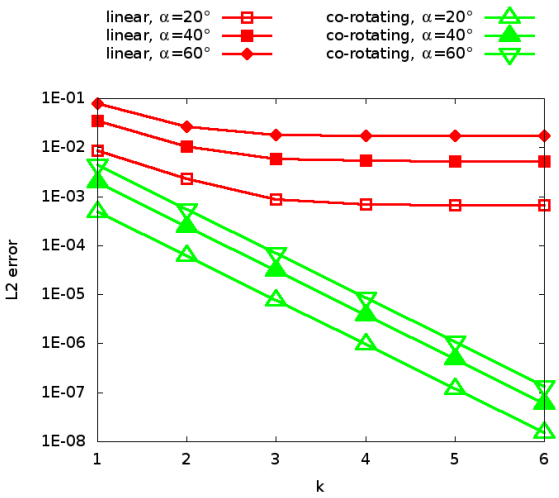


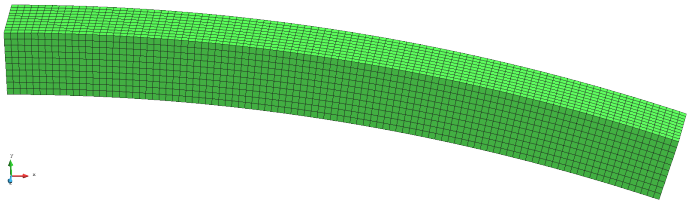
Figure 5.12: Convergence of deformation mapping error of both linearized and co-rotating algorithms under different α.

the length as

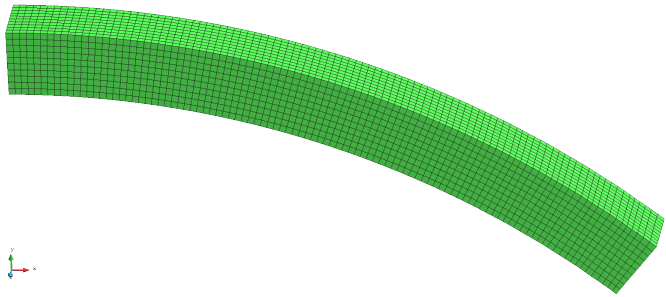
$$\theta_x = \beta \frac{x}{l} \quad \text{with } 0 < x < l, \tag{5.31}$$

where β is the twist angle at the right end. Three tests with $\alpha = \beta = 20°$, $40°$ and $60°$ are performed. The reference results are computed from the analytical functions as in the bending test. The error of the displacements of the surface nodes in the L2 norm with respect to different refinement levels of the beam curve is shown in Fig. 5.15. The deformed surface meshes with $k = 4$ are shown in Fig. 5.16 and Fig. 5.17.

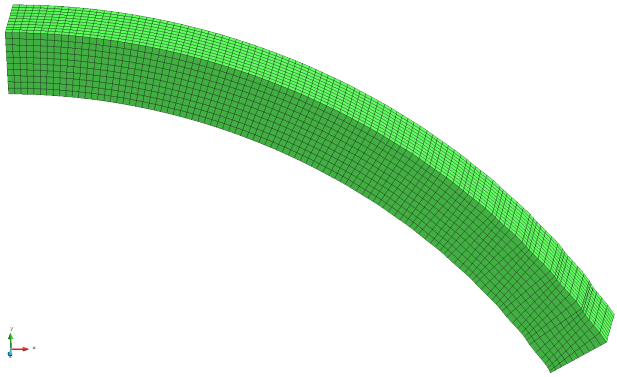
It can be seen that the co-rotating algorithm is again more accurate, but the convergence order decreases during refinement e.g. from 1.93 to 1.39 for $\alpha = \beta = 60°$. Additional error comes from the step where 3D rotation is decomposed into three individual rotations. However, the co-rotating algorithm already fulfills the requirements of a convergence order higher than 1 and a smooth surface without distorted elements.



(a) $\alpha = 20°$

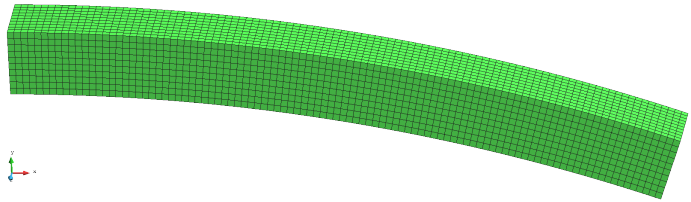


(b) $\alpha = 40°$

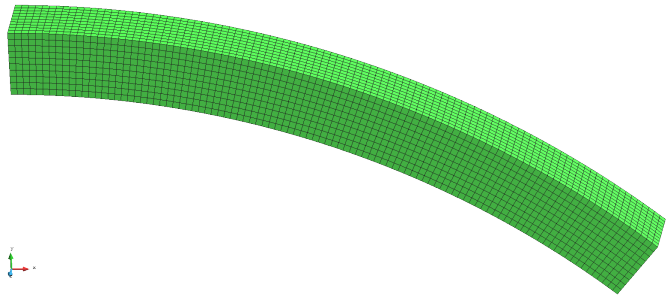


(c) $\alpha = 60°$

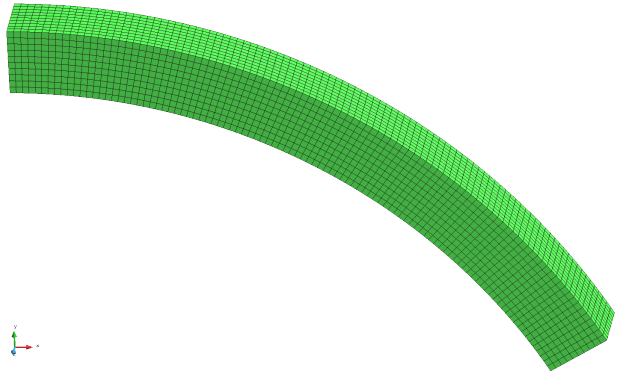
Figure 5.13: Results of the linearized algorithm with 16 beam curve elements under different α.



(a) $\alpha = 20°$



(b) $\alpha = 40°$



(c) $\alpha = 60°$

Figure 5.14: Results of the co-rotating algorithm with 16 beam curve elements under different α.

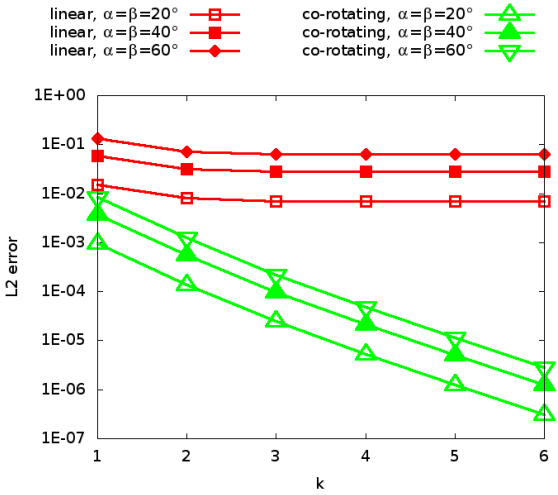
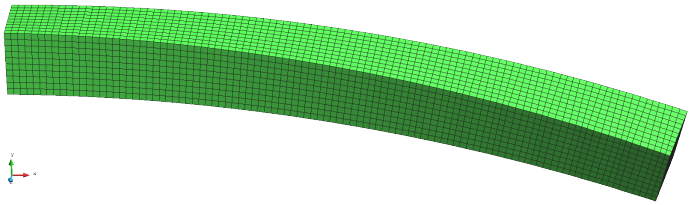
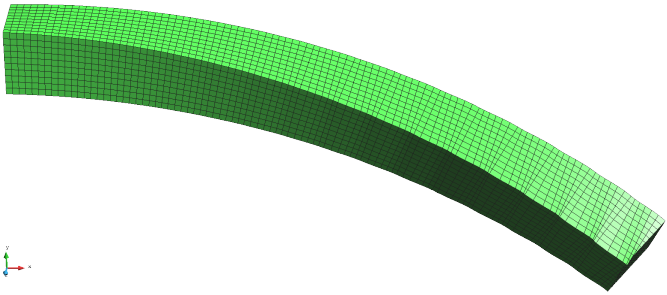


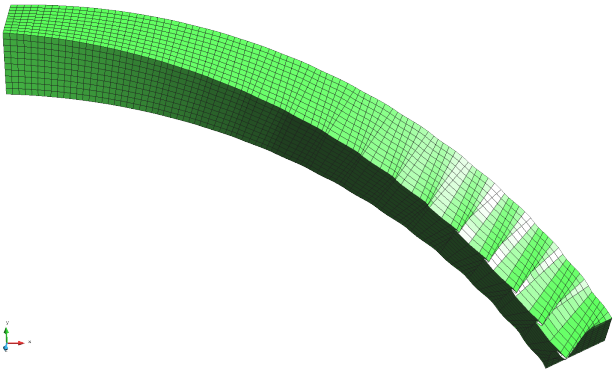
Figure 5.15: Convergence of deformation mapping error of both linearized and co-rotating algorithms under different α und β.



(a) $\alpha = \beta = 20°$

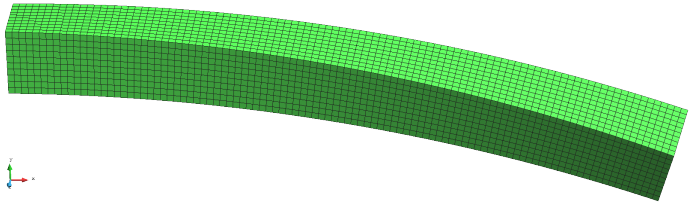


(b) $\alpha = \beta = 40°$

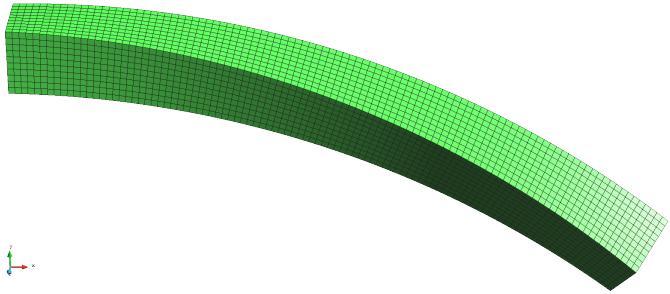


(c) $\alpha = \beta = 60°$

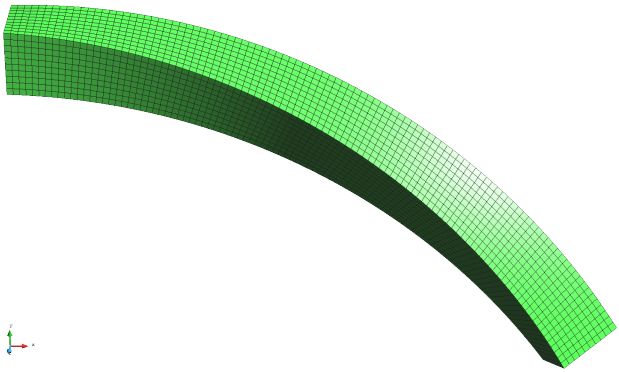
Figure 5.16: Results of the linearized algorithm with 16 beam curve elements under different α and β.



(a) $\alpha = \beta = 20^\circ$



(b) $\alpha = \beta = 40^\circ$



(c) $\alpha = \beta = 60^\circ$

Figure 5.17: Results of the co-rotating algorithm with 16 beam curve elements under different α and β.

5.4 Practical Examples

The co-rotating algorithm for mapping large displacements and rotations is further tested on two more complex geometries. The first is a wing of an airplane which can be found in [20]. The meshes of the beam curve and the surface are shown in Fig. 5.18. The beam curve is a straight line passing the 0.4 chord at the root and 0.5 chord at the tip. It is equally meshed by 8 elements. It is applied with the same analytically described twist and flapwise bending as in section 5.3.2 with $\alpha = \beta = 60°$. The result is shown in Fig. 5.19. One remark is that the nodes are connected by straight lines instead of curves in the deformed 1D mesh due to the limit of the visualization.

The second geometry is the blade of the NREL phase VI wind turbine [61]. The beam curve is a straight line passing the axis of the cylindrical root section which is equally meshed by 8 elements. Two surface meshes are tested: an unstructured mesh with triangles and a structured mesh with quadrilaterals allover except some triangles at the tip. For the surface reconstruction of the unstructured mesh a rigid body motion operator has to be computed for each fluid node. The efficient treatment proposed in Section 5.1.1 is used here for the structured mesh, so each 198 fluid nodes along the beam length direction are regarded as belonging to the same cross section, except the 793 nodes at the blade tip. It results in only 208 rigid body motion operators for the individual cross sections. The treatment is quite practical since resolving the boundary layer of turbulent flows usually leads to structured boundary meshes. The meshes are shown in Fig. 5.20. The beam is applied with the same analytically described twist and flapwise bending as in section 5.3.2 with $\beta = 60°$ and $\alpha = 180°$, which means it is bent into a half circle. Such big deformation do not happen in reality but it is used to test the performance of the co-rotating algorithm. The results are shown in Fig. 5.21 and Fig. 5.22.

The tests on both geometries result in the smooth deformed surfaces without distorted or collapsed elements, and thus verify the robustness and smoothness of the co-rotating algorithm for large displacements and rotations.

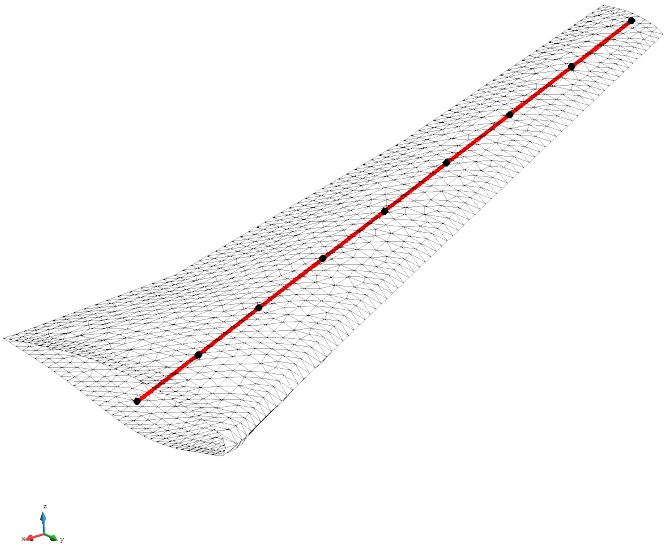


Figure 5.18: The curve and surface meshes of the airplane wing.

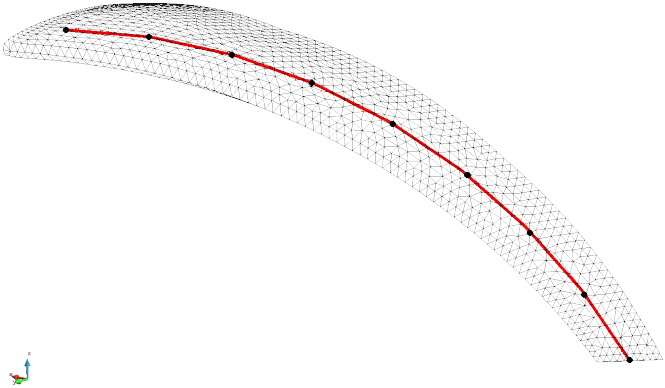
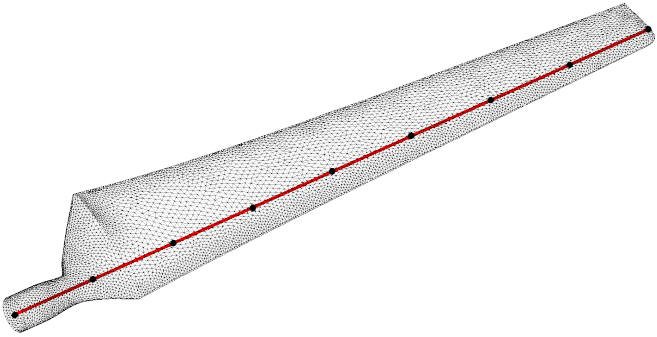
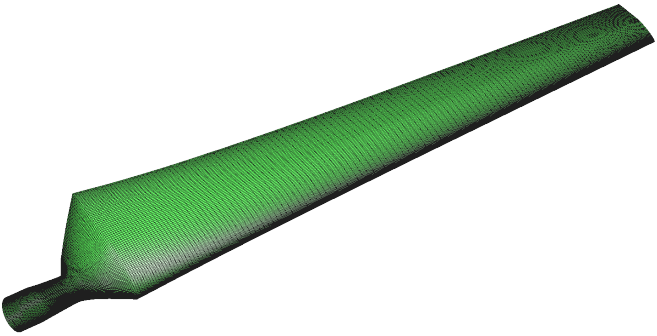


Figure 5.19: The curve and surface meshes of the airplane wing after deformation.

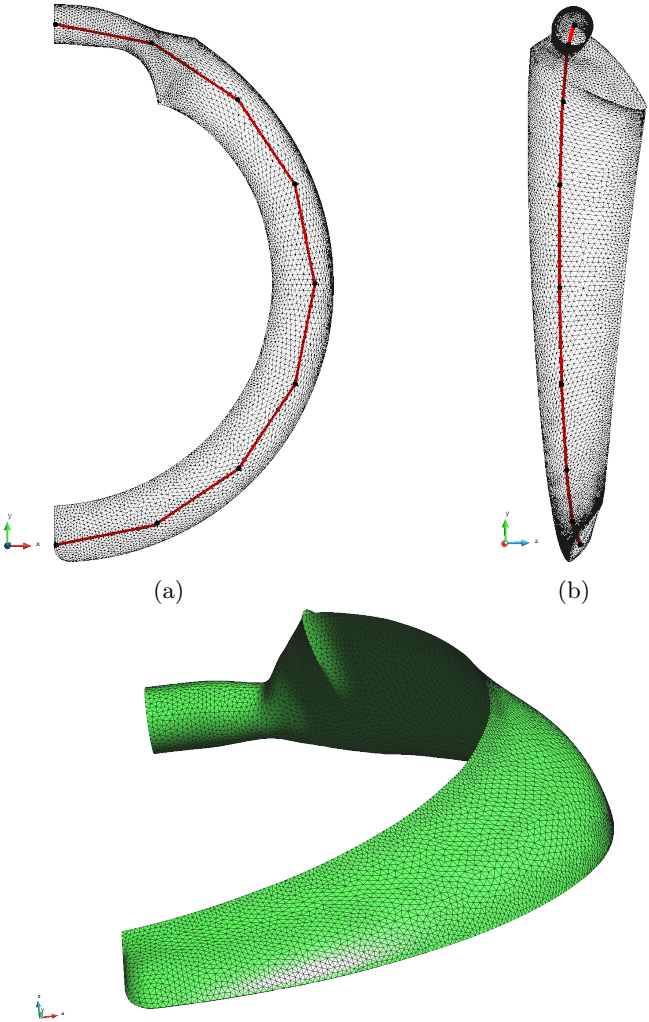


(a) The curve and the unstructured surface meshes. The unstructured mesh has 20876 elements and 10458 nodes.



(b) The structured mesh which has 41680 elements and 41779 nodes.

Figure 5.20: The curve and the surface meshes of the blade of the NREL phase VI wind turbine.



(a) (b)

(c)

Figure 5.21: The curve and the unstructured surface meshes of the blade after deformation with three different views. The unstructured surface mesh is displayed with two different styles.

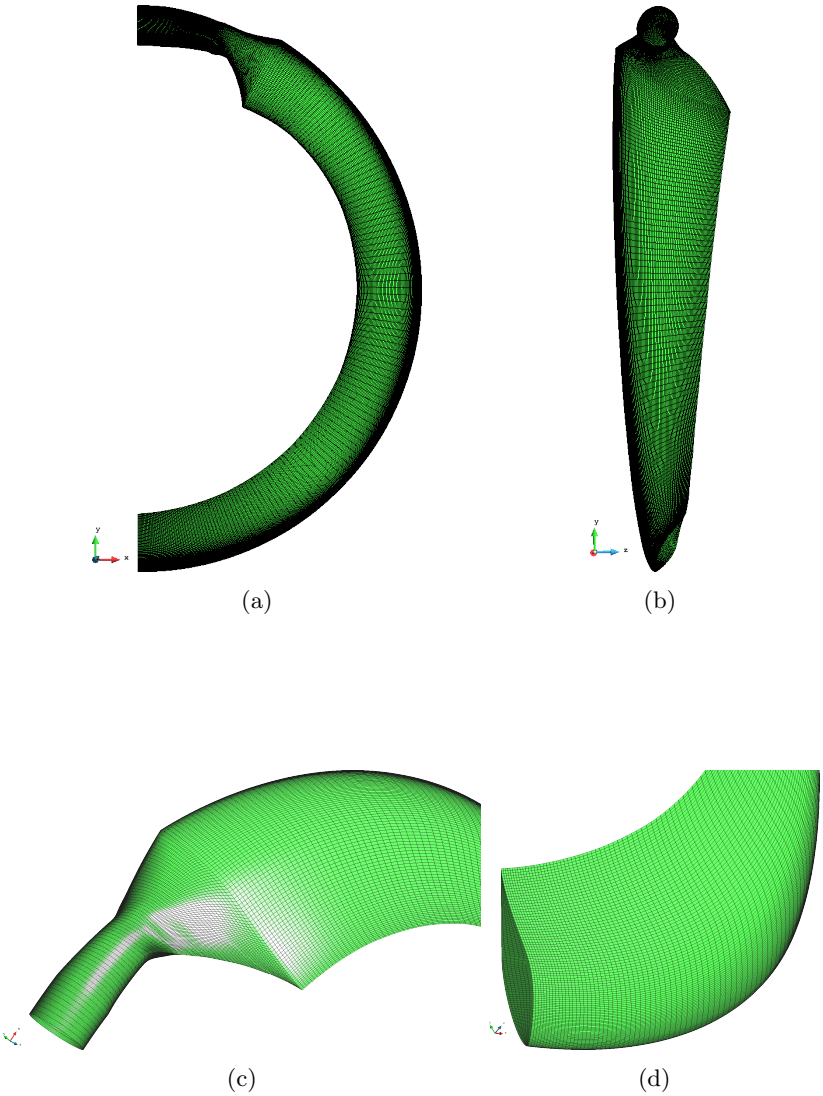


Figure 5.22: The curve and the structured surface meshes of the blade after deformation with four different views. The last two view zoom in at the root and tip due to the high refinement.

5.5 Summary

This chapter focuses on the surface reconstruction of a 2D fluid surface mesh from an 1D beam mesh in the context of FSI. The mapping is achieved by two steps: the cross section of a surface node is determined in the first step and the rotation and translation of the cross section is interpolated in the second step. The co-rotating algorithm is able to interpolate large displacements and rotations. The convergence tests show its accuracy and satisfactory convergence behavior under pure bending as well as under combined twist and bending, which cannot be achieved by the linearized algorithm. The tests on an airplane wing and a wind turbine blade demonstrate the smoothness and robustness of the co-rotating algorithm with only a small number of 1D elements and under artificially large deformation.

The mapping is based on the beam kinematics with simplifications. The elastic and the shear centers are assumed coincided, and the cross sections are assumed rigid and orthogonal to the beam axis during deformation. This corresponds to the assumptions for nonlinear slender beams as in the so-called theory of elastica. When the assumptions are not valid, e.g. in case of Timoshenko beam or warping, the algorithms can be extended by adding more parameters related to warping or shear deformation. However, the presented algorithm has a wide range of applications and can give approximate solution when these factors are not severe.

A comparison between mapping with beam elements and mapping with surface meshes introduced in Chapter 4 is given below. For linear beam elements, the nodal rotations can be operated as vectors so it also results in a mapping matrix. But DOFs on the same node are coupled meaning that the mapping matrix does not work in a componentwise way. For nonlinear beam elements, the rotation angles appear inside trigonometric functions, so the mapping operator is nonlinear. The consistency criterion is easy to realize since pure translation can be exactly mapped. With regard to the load mapping, the conservative mapping presented in Section 4.2 is the same process as computing the consistent nodal forces and moments presented in Section 5.2.

Chapter 6

Co-Simulation Examples

The co-simulation environment EMPIRE has been validated with FSI benchmark examples such as a flow past a cylinder attached with an elastic flag [127] and a driven cavity with flexible bottom [134]. The works of validating EMPIRE can be found in [136] and [50]. Besides, the co-simulation environment has been also applied in many industrial FSI problems especially with respect to civil engineering. Moreover, applying EMPIRE in co-simulations with more than two partitions has been presented in [120], where fluid-structure-control coupled problems are studied.

In this chapter, three co-simulation examples are selected to be presented, including:

- FSI simulation of a flat membrane in a wind tunnel. The membrane is modeled as a zero-thickness object in the fluid domain, and the interface contains the upper and lower surfaces of the membrane. One difficulty comes from solving the mesh motion with sharp edges at the interface. The simulation results can be compared with the measurements from the wind tunnel experiments.

- FSI of wind turbine blades. The blades are modeled by a large number of structural shell elements in another work. However, they are modeled by only a small number of beam elements in this work and the mapping algorithms for beam elements presented in Chapter 5 can be used. Since the fluid model is exactly the same, the results due to different structural models are compared.

- Shape optimization of a prototypical 2D hydrofoil. The optimization problem is simple in the sense that there are only one single design variable and one single objective, and the fluid and structure models are small. But it is a representative case of optimization on a multiphysics problem.

In the following, the examples will be presented in Section 6.1, 6.2 and 6.3 sequentially.

6.1 Flat Membrane in a Wind Tunnel

To study the basic aeroelastic behavior of membrane structures under wind load, experiments on a simple flat membrane are carried out by a research group in Harbin institute of technology in China. The experiments have measured the structural and the fluid properties individually as well as some properties during interaction. Due to the simplicity in the geometry and the physical properties of both the membrane and the wind, the problem can serve as a benchmark for FSI simulation on membrane structures. The establishment, conditions and measurements of the experiments can be found in [27].

To guarantee the quality of the FSI simulation, models of the single partitions are validated first before the coupled problem is simulated. The in-house structural solver Carat++ [51] and the open-source fluid solver OpenFOAM [137, 74] are used for solving the individual domains. They are coupled within EMPIRE to solve the FSI problem.

6.1.1 Experiments

The experiments are briefly introduced here with a focus on the parameters which relate to the numerical models and the post-processing.

Geometry

The installation of the flat membrane and the measuring instruments inside the wind tunnel is shown in Fig. 6.1, which is also sketched in Fig. 6.2. The membrane is located in parallel to the ground with a height of $H = 400\,\text{mm}$. It is rectangular and has a length of $L = 600\,\text{mm}$ and a width of $B = 1200\,\text{mm}$. The longer edges of the membrane surface are fixed by two steel cylindrical tubes with a diameter of 12 mm. The thickness of the membrane is 0.4 mm. The test section of the wind tunnel has a height of 3 m and a width of 4 m.

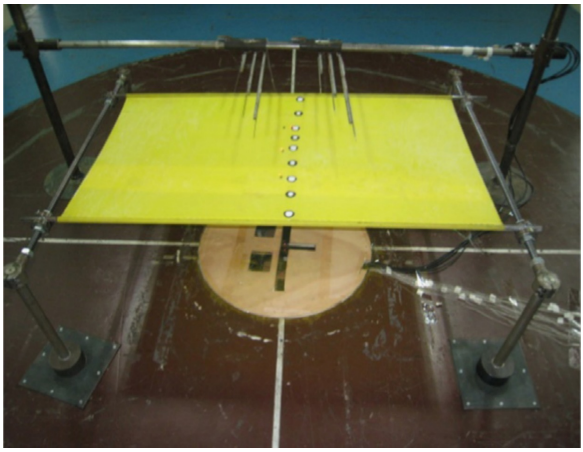


Figure 6.1: Experiment setup of the flat membrane in a wind tunnel (photographed from [27]).

Wind properties

The wind tunnel is able to generate a smooth flow with almost constant profile of velocity and turbulence intensity. The flow velocity at the inlet is 10 m/s with a turbulence intensity of 0.5%. The air flow has a kinematic viscosity of $1.81 \cdot 10^{-5}\,\mathrm{m}^2/\mathrm{s}$ and a density of $1.2250\,\mathrm{kg/m}^3$ which are measured at 15 °C.

Membrane properties

The material of the membrane is latex with a density of 1033.45 $\mathrm{kg/m}^3$, a Young's modulus of 1.638×10^{-3} GPa and a Poisson's ratio of 0.4. A pretension of 30 N/m is added on the membrane by the two cylindrical steel tubes fixing the longer edges.

Measuring

Three laser displacement meters are installed on the wind tunnel floor to measure the vibration of the membrane. They are located under the $L/4$, $L/2$ and $3L/4$ positions of the center line of the membrane along the inflow direction. During the experiment, the displacements are sampled with a frequency of 500 Hz for a total time of 100 s. The hot wires over the membrane are used to measure the wind velocities at different

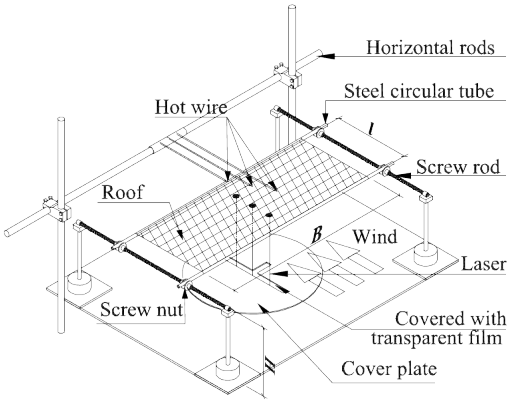


Figure 6.2: Sketch of the experiment setup of the flat membrane (photographed from [27]). The displacements under wind load are measured at the center line with three equidistant laser displacement meters.

positions. When comparing the simulation results with the measurements, only the displacements are considered which can represent the aeroelastic behavior of the membrane.

To validate the single fluid model, the flat membrane is replaced by a glass plate with the same geometry except that it has a different thickness of 18 mm, as shown in Fig. 6.3. Note that the cylindrical tubes fixing the longer edges are not needed. The experiment is conducted with the same flow condition as that with the membrane. The wind pressures on the upper and lower surfaces of the glass plate are measured by pressure transducers, which are distributed equidistantly at the center lines along the inflow direction of the upper and lower surfaces, as shown in Fig. 6.4. The pressures are sampled with a frequency of 625 Hz for a total time of 20 s.

The eigenfrequency of the membrane under 30 N/m pretension is measured as 7.0 Hz. Note that it is the single measurement provided to validate the numerical model of the membrane.

6.1.2 Structure Model and Validation

The structure is modeled by 40×40 equal-sized nonlinear membrane elements as shown in Fig. 6.5. The geometry, material properties and

Figure 6.3: Experiment setup of the glass plate in the wind tunnel (photographed from [27]).

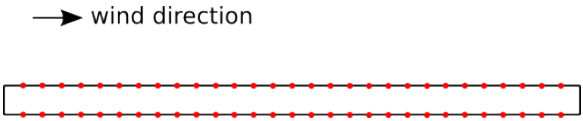


Figure 6.4: There are 29 equidistant pressure measurement locations at the center line of the upper and lower surfaces of the glass plate, respectively.

pretension are set according to the description in Section 6.1.1.

An eigenfrequency simulation is perfromed first on the membrane. The simulated 1st eigenfrequency is 7.10 Hz with a mode shown in Fig. 6.6. The error is 1.4% compared with the measured value 7.0 Hz.

For the transient FSI simulation, the generalized-α method is used for time integration. The time step size is 0.001 s and the total time is 20 s.

6.1.3 Fluid Model and Validation

Due to different geometrical details between the rigid plate and the flat membrane with the fixing cylinders, two different meshes of the fluid domain are created for both cases. The first mesh is used in the CFD simulation of the glass plate in the wind tunnel, as shown in Fig. 6.7.

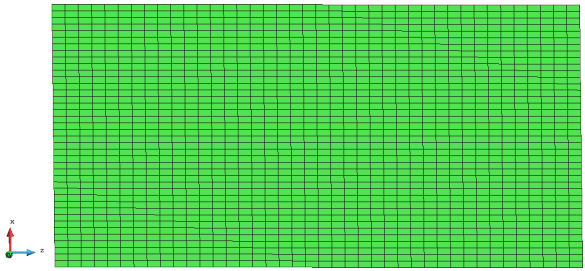


Figure 6.5: The flat membrane is modeled by 40 × 40 membrane elements.

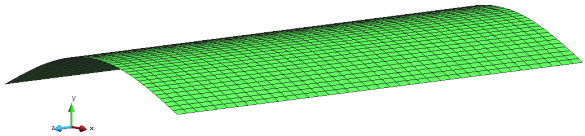
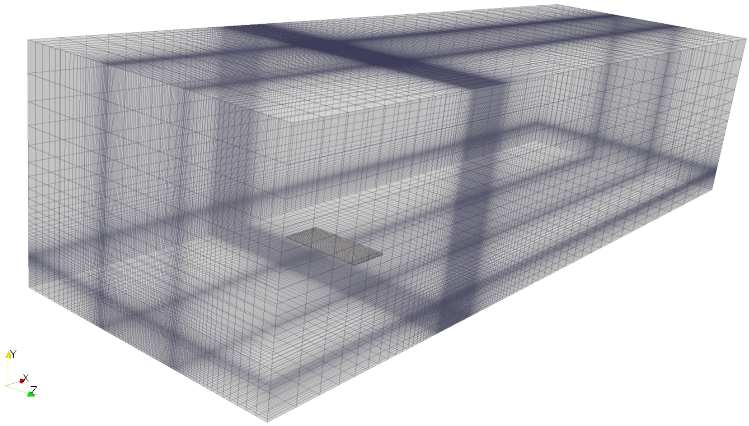


Figure 6.6: Simulated eigenvector regarding the first eigenfrequency of the flat membrane.

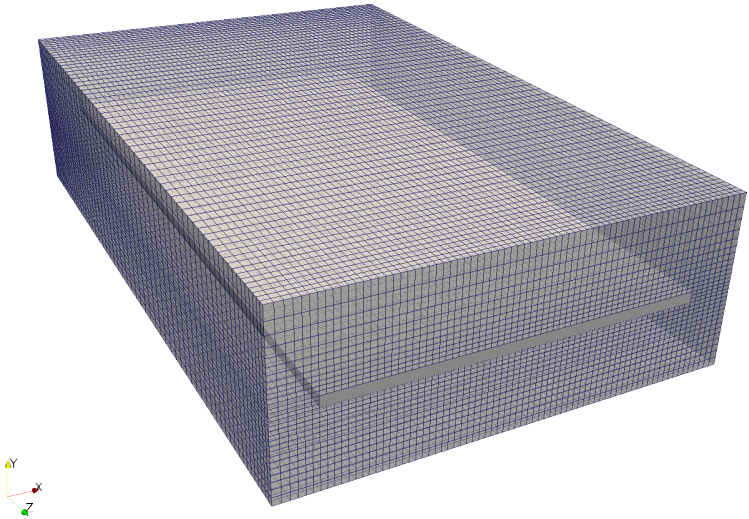
The second mesh is used in the FSI simulation of the flat membrane in the wind tunnel, as shown in Fig. 6.8. Both meshes have very close grid densities and they are different mainly around the individual obstacles. Moreover, the boundary conditions, the solution algorithms and time step size are the same in both simulations. As a result, the validation of the former model can justify the latter one for the FSI simulation.

Glass plate

The fluid mesh in Fig. 6.7 is structured and has ≈ 1.0 million cells. The height of the cells next to the plate surfaces is 6 mm which gives a $y+$ value approximately between 4 and 120 during the simulation. The flow velocity and turbulence intensity at the inlet follow the experiment description in Section 6.1.1. The k-ω-SST method is chosen for turbulence modeling. The values of k and ω at the inlet are computed out of the turbulence intensity. Central difference is used to interpolate the diffusion term and the Vanleer's TVD scheme is used to interpolate the convection term. The PIMPLE algorithm is chosen for solving the incompressible Navier-Stokes equations. The time step size is 0.001 s and the total time is 20 s. The maximal CFL number during the simulation

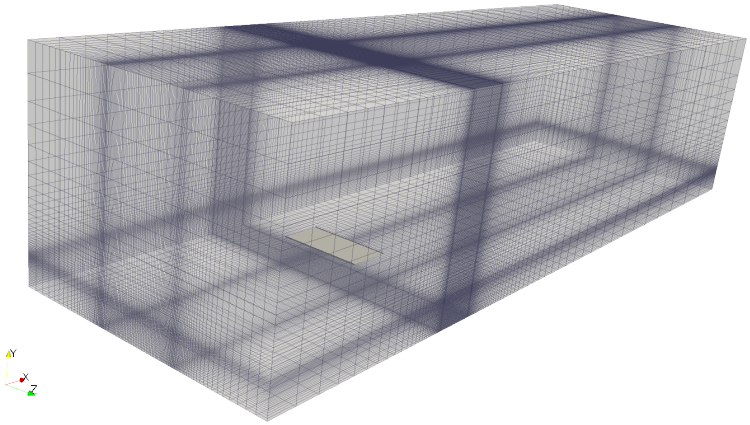


(a) Fluid mesh

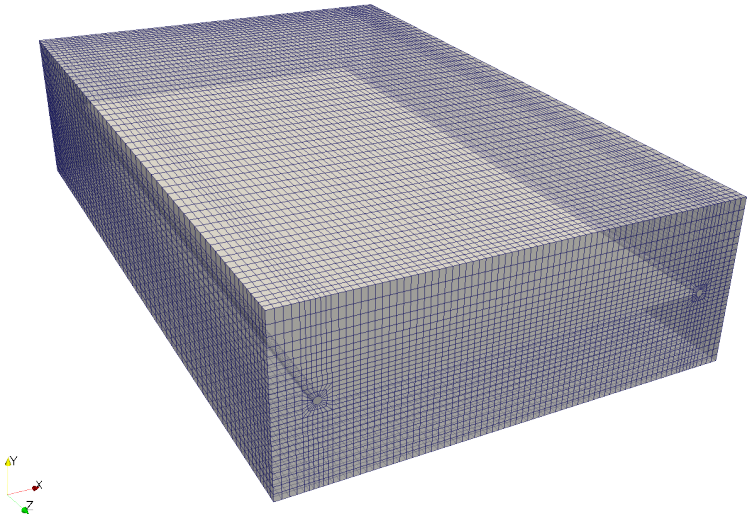


(b) Zoomed in around the plate.

Figure 6.7: Fluid mesh of the wind tunnel with the glass plate.



(a) Fluid mesh.



(b) Zoomed in around the membrane.

Figure 6.8: Fluid mesh of the wind tunnel with the flat membrane.

is around 33 while the average is 0.13.

The pressure and velocity fields from the CFD simulation at 20 s are shown in Fig. 6.9. The statistical results of the simulated pressures at the measuring locations in the last 10 s are compared with those from the measurements, as shown in Fig. 6.10. The meaning of the box-and-whisker plot is illustrated in Fig. 6.11. From the results it can be seen that:

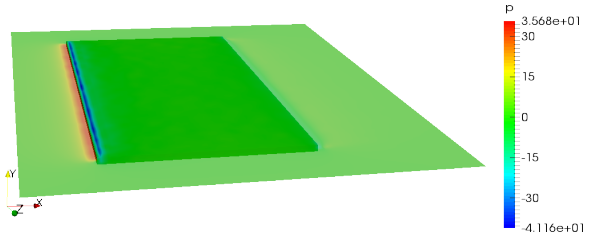
- In the measurements, the minimums and maximums are far beyond the regular range of deviation, so it is a sign of problems of the measurements at single times. Therefore, the extremes in the measurements cannot be trusted, but the standard deviations are still meaningful.
- The standard deviations from the simulation are smaller than those from the measurement. The difference can come from the time-averaged behavior of the turbulence model, or the objects not modeled in the simulation such as the wooden blocks that fixing the glass plate (see Fig. 6.3).
- A very good agreement regarding the mean values is obtained, especially from the 20% position to the right end. However, the mean values around the leading edges are not accurately simulated, where the wind starts to hit the obstacle.

It is found in practice that the results can hardly be improved by increasing the grid density or choosing another turbulence model. However, the results are accepted since the deviations and especially the mean values on a major part of the plate are satisfactory. So the fluid model is considered as validated in the average sense.

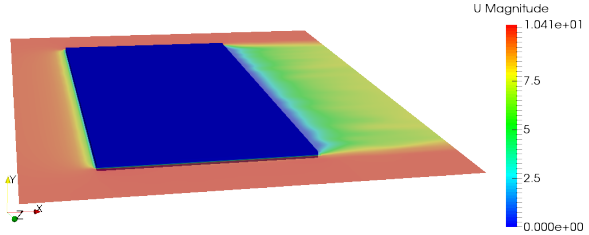
Flat membrane

The fluid mesh in Fig. 6.8 is structured and has ≈ 1.1 million cells. The height of the cells next to the membrane surfaces is 6 mm which gives a $y+$ value between 2.5 and 115 during the FSI simulation. The settings of the fluid part for the FSI simulation are the same as those for the CFD simulation with the glass plate.

The meshes on the membrane upper and lower surfaces are matching. They are overlapped i.e. the thickness is modeled as zero. One of the meshes is shown in Fig. 6.12. The grid density is higher around the free edges of the membrane to deal with the problem of the Laplace's mesh motion solver. The details are given below.



(a) Pressure field.

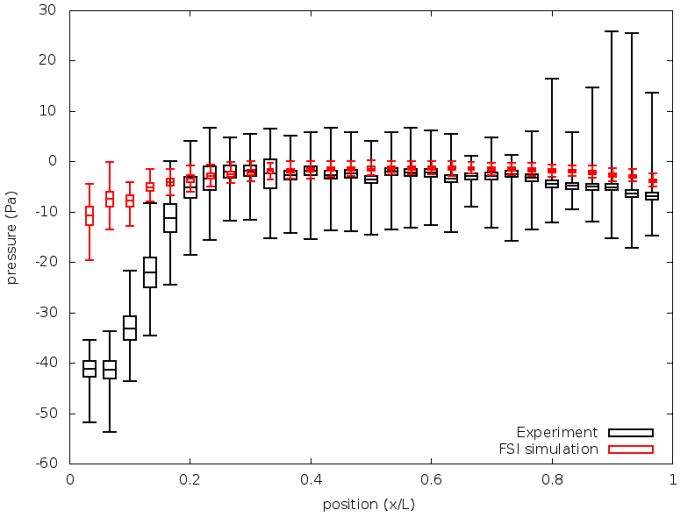


(b) Velocity magnitude field.

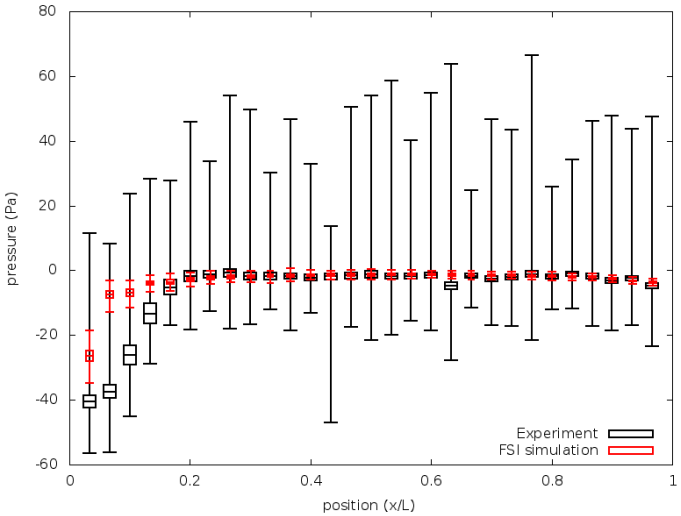
Figure 6.9: Flow fields around the plate at 20s of the CFD simulation.

Let's first consider a 2D heat transfer example in Fig. 6.13a, where an internal zero-thickness object has temperature of 1 °C and the domain boundaries have temperature of 0 °C. The temperature field solved by the Laplace's equation solver has oval-like contour lines, and the temperature around the ends of the object is smaller than that on the object. If the temperature field becomes the mesh displacement field in the vertical direction (scaled with a certain factor), a deformed mesh can be obtained as shown in Fig. 6.13b. It can be seen that the cells around the ends of the object are distorted. The reason is that the displacements of the cells around the end points are smaller than the ones on these points. The problem is more severe if the cells have a larger length ratio. In practice it is found that even more advanced diffusivity algorithms cannot solve the problem.

Coming back to our case, the two free edges (the shorter edges) of the membrane in 3D correspond to the two end points of the object in the 2D example, i.e. the 2D example represents a cross section of the wind tunnel orthogonal to the inflow direction. And it is found that the FSI simulation will crash due to distorted cells around the free edges if the cells have a relatively big length ratio. As a workaround, the length ratio is reduced for these cells, which at least guarantees that



(a) Upper surface.



(b) Lower surface.

Figure 6.10: Comparison between the simulated and measured pressures on the surfaces of the glass plate.

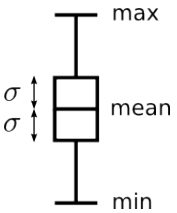


Figure 6.11: Meaning of the box-and-whisker for statistical analysis.

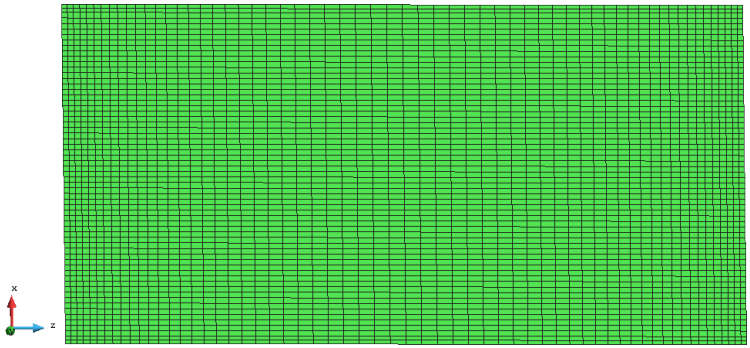
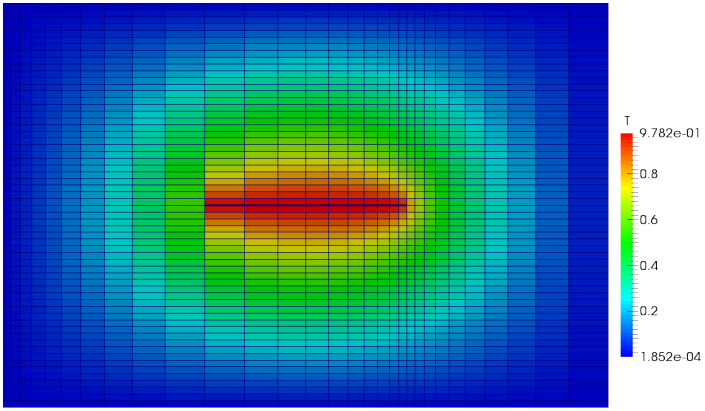


Figure 6.12: Fluid mesh at the membrane surface. There are 63 elements in the direction of the shorter edges and 60 elements in the direction of the longer edges.

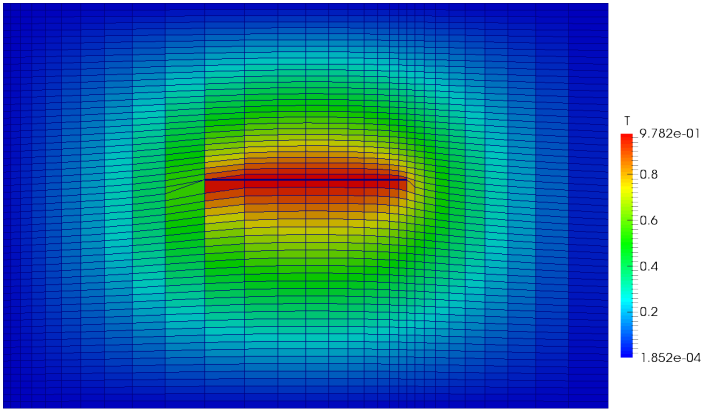
distorted cells will not happen in this simulation. Improvement of the Laplace's mesh motion solver is not further studied in this work, but one idea is shared here that the cells around the sharp edges should take approximately the same displacements as those on the edges.

6.1.4 FSI Simulation and Results

Displacements and forces are exchanged at the upper and lower surfaces of the membrane. In the co-simulation, each client defines two meshes which coincide with each other. But the surface meshes form the structure and the fluid are non-matching as shown in Fig. 6.5 and Fig. 6.12, so the standard mortar mapping algorithm is used. Since the wet-surface is a simple rectangular plane, the algorithm can be used in a conservative way without enforcing consistency. The sum of the fluid forces on the upper and lower membrane surfaces is assigned to



(a) Temperature field solved by Laplacian solver.



(b) Apply the temperature as the mesh displacement in the vertical direction.

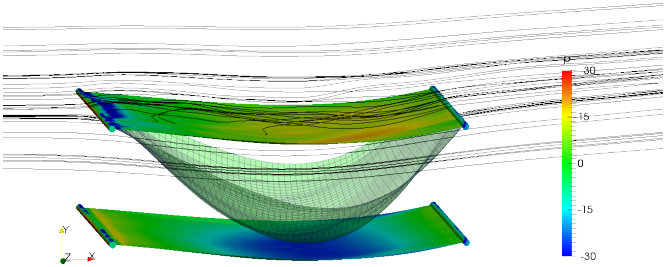
Figure 6.13: An illustration of the problem when solving the mesh displacement by the Laplace's equation. The temperature field in (a) is applied as the mesh displacement field in (b).

the structure partition as external forces. The displacements of the upper and lower surface meshes from the structure are the same so there are also no gaps between the upper and lower membrane surfaces in the fluid domain. Iterative coupling is used in combination with the Aitken's relaxation method. Linear extrapolation is used for prediction. The iterative loop at each time step is converged after two iterations with an absolute residual below $1 \cdot 10^{-7}$ m. So the interaction is quite stable.

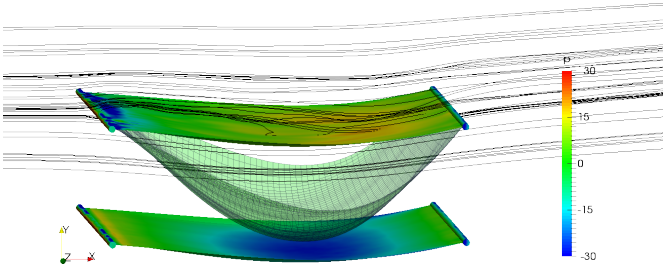
The deformed membrane and the flow around it at different times are shown in Fig. 6.14. The simulated displacements on the membrane at the measuring locations are compared with those from the measurements, as presented in Fig. 6.15. The statistical results of them in the last 10 seconds are given in Fig. 6.16. Note that the displacement is defined on the location of a laser meter instead of on a material point of the membrane. From the figures it can be seen that:

- The deformation of the membrane is a 3D phenomenon, i.e. the internal region has a larger deformation than the edges and the wind over the membrane can flow in the z-direction.
- The simulated displacements have a smaller amplitude of oscillation than the measured ones. And the high-frequency modes in the displacements as a sign of turbulence are not reproduced in the simulation.
- The ranges of the simulated displacements locate within those of the measured values, but the standard deviations of the simulated values are considerably smaller.

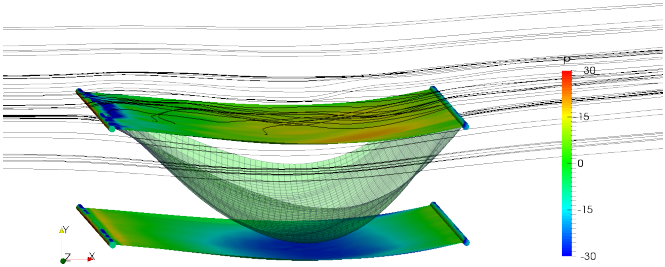
Similar to the CFD simulation results, the results of the FSI simulation are satisfactory only in the average sense. And the dynamical behavior of the interaction between the wind and the membrane is not accurately simulated. To get better results in the future, more studies should be put in turbulence modeling. What's more, as a potential benchmark example, the pressure measuring over the glass plate should be improved. More data should be measured as well e.g. higher order eigenfrequencies and displacements at more locations.



(a) 19.5 s.

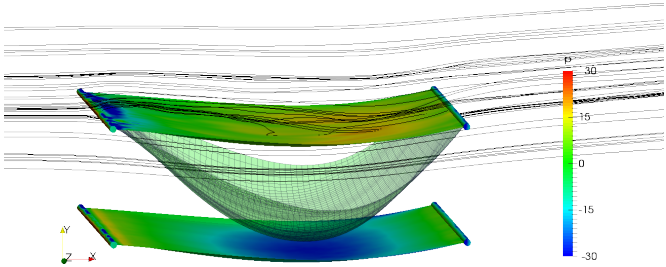


(b) 19.6 s.

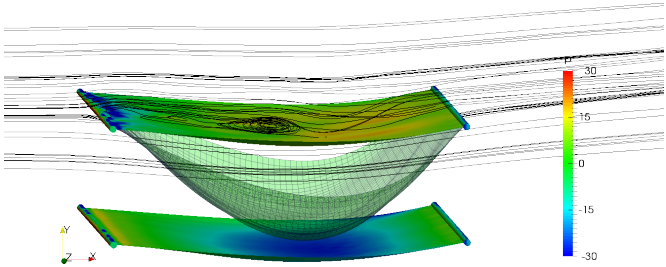


(c) 19.7 s.

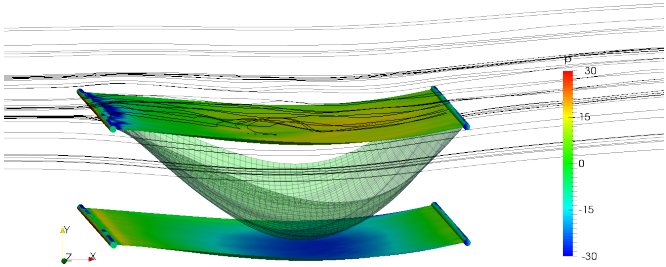
Figure 6.14: Results of the FSI simulation at different times. The surface above shows the pressure field on the upper surface of the membrane while the surface below shows the pressure field on the lower surface. The grid surface shows the deformation with a scaling factor of 5.



(d) 19.8 s.

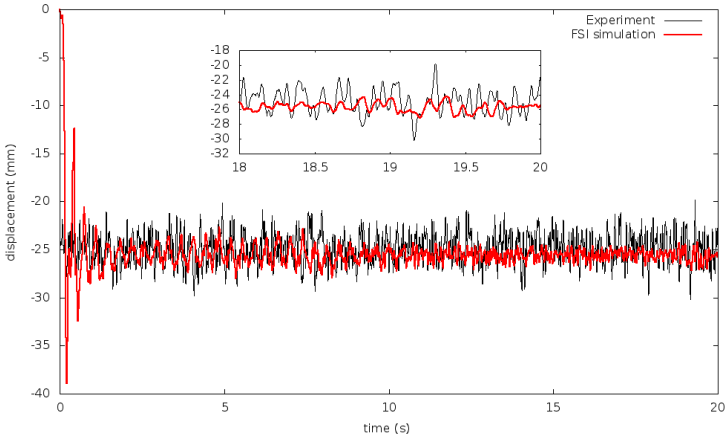


(e) 19.9 s.

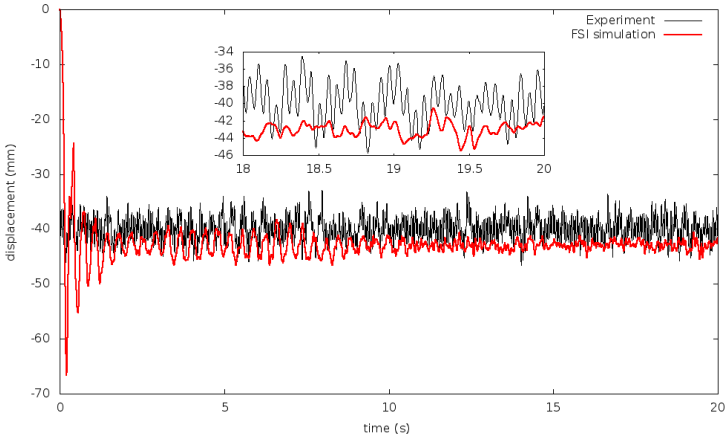


(f) 20.0 s.

Figure 6.14: Results of the FSI simulation at different times. The surface above shows the pressure field on the upper surface of the membrane while the surface below shows the pressure field on the lower surface. The grid surface shows the deformation with a scaling factor of 5.

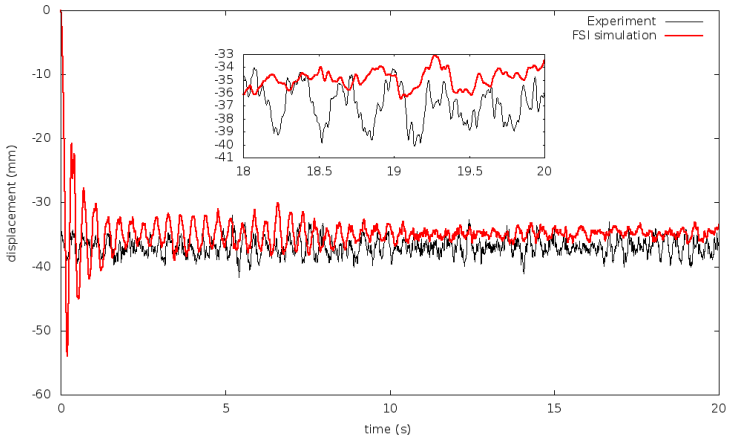


(a) 1/4 length.



(b) 1/2 length.

Figure 6.15: Evolution of the displacements at three measured locations from the FSI simulation and the experiment.



(c) 3/4 length.

Figure 6.15: Evolution of the displacements at three measured locations from the FSI simulation and the experiment.

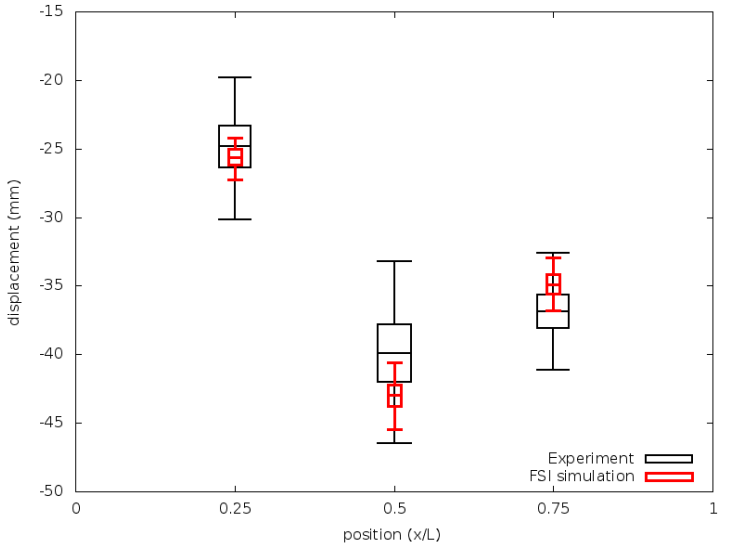


Figure 6.16: Statistical results of the simulated and measured displacements at different measured locations.

6.2 NREL Phase VI Wind Turbine

The NREL phase VI wind turbine has two blades with a diameter of 10.058 m and its hub height is 12.192 m. Experiments are performed on it by the National Renewable Energy Laboratory (NREL) in the USA to study the aerodynamic behavior of wind turbines. The wind turbine is well documented in [61] including the geometries, structural and flow properties and the experiment results. Therefore, the problem is chosen for CFD and FSI simulations in many works such as [69], [86] and [98].

Among the existing works, [120] uses EMPIRE to perform a co-simulation with up to four solvers, including a fluid solver, a structure solver, a generator solver and a controller of the pitching angle. Different from the other works where the rotational velocity of the wind turbine is fixed, the co-simulation in [120] enables dynamical rotational velocity. Besides, the so-called "emergency brake maneuver" is simulated where the controller can brake the rotation by changing the pitching angle of the blades.

In this work, an FSI simulation of the wind turbine is performed where the blades are modeled by structural beam elements. The mapping algorithms for beam elements developed in Chapter 5 can be used. The validated fluid model in [120] is reused in the FSI simulation here, but the structural shell model is replaced by a beam model.

6.2.1 Previous Simulation with Shell Elements

The FSI simulation of the NREL phase VI wind turbine in [120] is briefly introduced here. The commercial code Abaqus is used as the structure solver and OpenFOAM is used as the fluid solver. The fluid model here will be reused in the new FSI simulation where the structure is modeled by beam elements.

Structure Model

Each blade is modeled by $\approx$ 60 thousands nonlinear shell elements in Abaqus. The composite structures and the installed instruments have also been taken into consideration. The geometries of one blade is shown in Fig. 6.17. The CSM model is validated against a few structural properties given in the documentation. The results will be comparatively presented later together with those from the beam model.

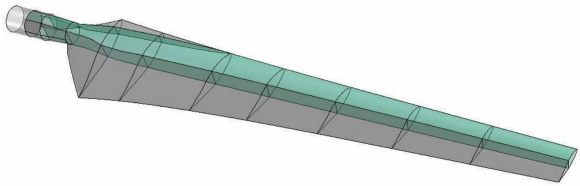


Figure 6.17: Shell model of one blade (photographed from [120]).

Fluid Model

The fluid domain is shown in Fig. 6.18a. The rotation of the blades is realized by a region rotating with the blades. A side view of the rotating region is given in Fig. 6.18b. The interface between this region and the region outside is called *arbitrary mesh interface* (AMI), where the data are mapped via the standard mortar method [44]. The mesh has totally ≈ 9.9 million cells, and especially there are ≈ 2.6 million cells around each blade surface as shown in Fig. 6.19. It results in an average $y+$ of ≈ 7 and a maximum $y+$ of ≈ 12 on the blade surface.

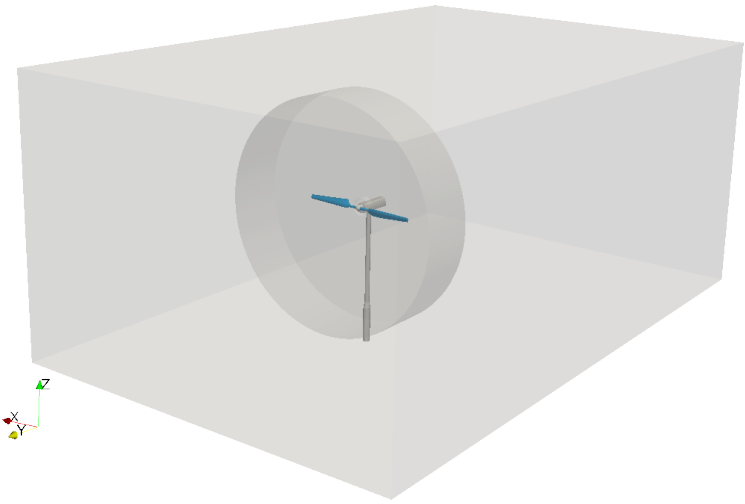
The air has a density of $1.23\,\mathrm{kg/m^3}$ and a kinematic viscosity of $1.46 \cdot 10^{-5}\,\mathrm{m^2/s}$. The inlet turbulence intensity is 0.5%. The test case concerned in this work has an inlet wind velocity of $7\,\mathrm{m/s}$. The rotational velocity is $72\,\mathrm{r/min}$ and the pitching angle of each blade is $3°$.

Regarding the numerical algorithms, the BDF2 method is used for time integration and second order accurate spatial interpolation schemes are used for the diffusion and the convection terms in the momentum equation. The k-ω-SST method is used for turbulence modeling.

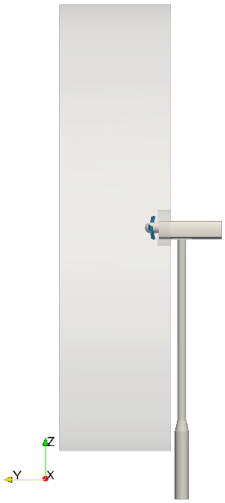
The simulated pressure coefficients at different locations on the blade radius are checked against the measurements. The results show very good agreement which are not further presented here. For details the reader is referred to [120].

FSI Simulation and Results

The displacements and forces are exchanged at the blade surfaces. And each solver sends two meshes to the server which are the individual blades. The standard mortar mapping algorithm is used on the blade meshes with enforcing consistency, which is already presented in Section 4.5.3. The iterative coupling is not affordable in this case since solving

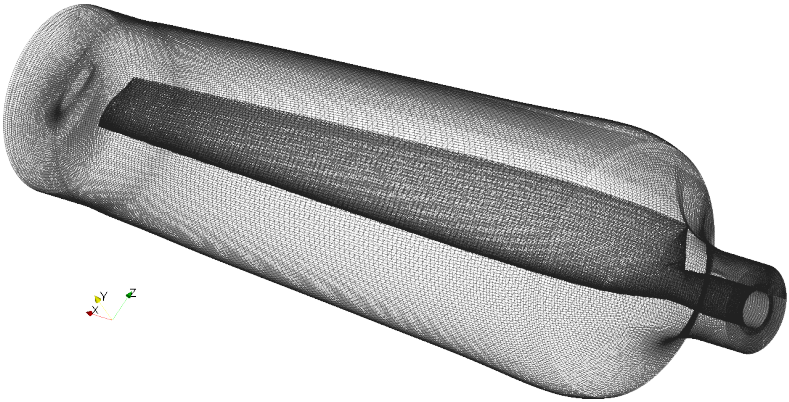


(a) Fluid domain.

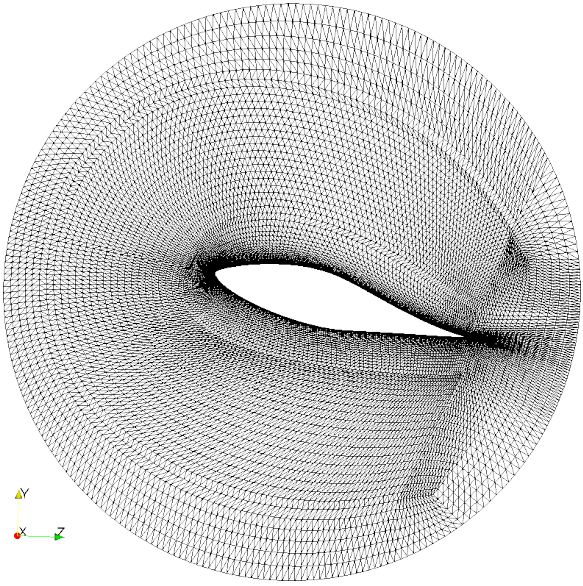


(b) The rotating region.

Figure 6.18: Fluid domain of the wind turbine simulation.



(a) Mesh.



(b) Cross section at the radius of 3m.

Figure 6.19: Part of the fluid mesh around the wind turbine blade.

the fluid domain takes a lot of time. As a result, loose coupling is used in combination with linear extrapolation.

The blade tip displacement and the rotor shaft torque over time are shown in Fig. 6.20. These results will be compared with those from the FSI simulation with beam elements.

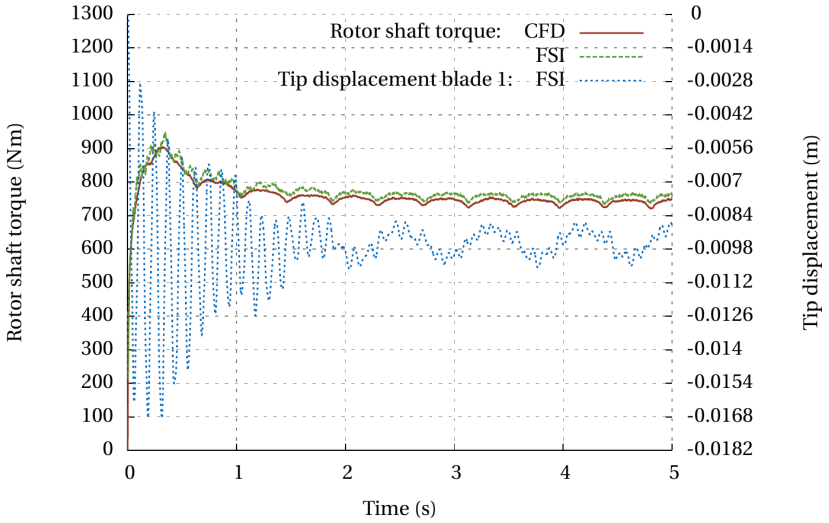


Figure 6.20: Simulation results including the rotor shaft torque and the tip displacement in y direction (photographed from [120]). The torque from a CFD simulation is given as a reference where the blades are rigid.

6.2.2 New Simulation with Beam Elements

Different from the work that is presented before, the turbine blades are modeled by beam elements in this work. Carat++ is chosen as the structure solver. The fluid model built inside OpenFOAM in the previous work is reused without changes. Another simple client is written which is in charge of transforming the loads from the original coordinate system to the rotational coordinate system. The three clients are coupled in EMPIRE to accomplish the FSI simulation of the wind turbine.

Structure model and validation

The beam model is built based on the beam properties of the blades at 26 different cross-sections given in [61]. The properties include mass

per unit length $\rho A\,(\mathrm{kg/m})$, axial stiffness $EA\,(\mathrm{N})$, edgewise stiffness $EI_{yy}\,(\mathrm{Nm^2})$, flapwise stiffness $EI_{zz}\,(\mathrm{Nm^2})$ and distance from center of rotation $x\,(\mathrm{m})$, where E is the Young's modulus, A is the area of the cross-section, I_{zz} and I_{yy} are respectively the area moment of inertia in the flapwise and edgewise directions. The beam properties are not measured, but estimated via some cross-section analysis tools e.g. [90, 16] instead. However, it is mentioned in the original document of these properties [61] that "a satisfactory match between blade frequencies predicted with the structural estimates and the measured frequencies has not been achieved".

Each blade is modeled by 10 beam elements as shown in Fig. 6.21. The distributions of the stiffness and mass properties over the blade length are linearly interpolated within the 26 discrete positions and then integrated on the beam elements to obtain the element properties. However, as the problem that has been mentioned before, following the documented beam properties results in big differences between simulated and measured values in center of gravity, total mass and eigenfrequencies. The simulated first eigenfrequency is 50% higher than the measured one which means the numerical model is too stiff. Therefore, the element properties are scaled in order to obtain a satisfactory agreement.

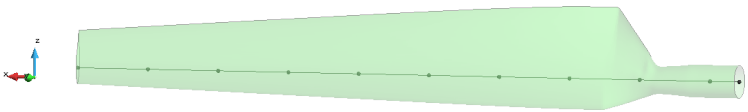


Figure 6.21: Beam model of the blade with 10 elements.

The final properties of the beam elements of one blade is given in Table 6.1. The Young's modulus is set as $E = 2.5 \cdot 10^7\,\mathrm{Pa}$, the density as $\rho = 1\,\mathrm{kg/m^3}$ and the Poisson's ratio as 0.2. One remark is that the single values of E, ρ, A, I_{zz}, I_{yy} here do not have physical meanings since the values of E and ρ are artificially set. But the combinations of them namely ρA, EA, EI_{yy} and EI_{zz} do have physical meanings. The beam elements are rotated around the beam axis by 3° according to the given pitching angle.

The eigenfrequencies and the tip displacement under gravity load are simulated with the beam elements. The simulated results as well as the total mass and center of gravity in the numerical model are compared with those form the shell model in [61] and those from the

Elem.	x_1 (m)	x_2 (m)	A (m^2)	I_{zz} (m^4)	I_{yy} (m^4)
1	0.32	0.529	18.1421	0.199407	0.152649
2	0.529	1.029	18.1421	0.0804055	0.0674528
3	1.029	1.529	16.2441	0.0311405	0.11323
4	1.529	2.029	16.0413	0.0233819	0.117525
5	2.029	2.529	14.3230	0.0178074	0.0998286
6	2.529	3.029	11.8717	0.0120966	0.0681247
7	3.029	3.529	10.7029	0.00871279	0.0497703
8	3.529	4.029	9.54008	0.00583339	0.0352038
9	4.029	4.529	8.43026	0.00368089	0.0238397
10	4.529	5.029	7.36111	0.00214558	0.0152379

Table 6.1: Setup of the beam elements.

measurements, as shown in Table 6.2. The results in the table are clarified in the following:

- In fact, the beam model is set based on the documented cross-section properties and tuned based on the first five properties of the shell model. That is the reason why these properties between them are very close. There are no more available information from the shell model so the beam model can approach the shell model only based on these limited properties.

- It is found that to obtain satisfactory first and second flapwise eigenfrequencies at the same time is not possible by just scaling the documented values, so it is decided to make the first eigenfrequency closer. The bigger differences in the second flapwise eigenfrequency and the tip z-th displacement are due to different mass and stiffness distributions as well as different background theories of the two structure models. Since the distributions of stiffness and mass in the shell model are not given, the results cannot be improved.

- Although the total mass, the center of gravity and the first eigenfrequencies are very close, these properties are only related to the most basic dynamical behaviors. The differences in other properties show that both models are similar in the overall sense but different in the local sense.

The eigen modes are shown in Fig. 6.22 and the deformation due to the gravity load is shown in Fig. 6.23.

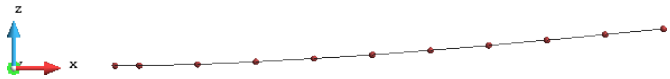
For transient analysis in the FSI simulation, the generalized α method is used for time integration with a time step size of 0.001 s and a total time of 5 s.

Property	Measured value	Shell [120]	Beam	Unit	Diff. %
Total mass	60.2	60.12	60.12	kg	0
Center of gravity	2.266	2.287	2.285	m	0.09
First flapwise eigenfrequency	7.313	7.901	7.909	Hz	0.10
Second flapwise eigenfrequency	30.062	30.981	32.734	Hz	5.66
First edgewise eigenfrequency	9.062	12.740	12.761	Hz	0.16
Tip z-th displacement under gravity		2.333	2.625	mm	12.5

Table 6.2: Measured and simulated structural properties of the blade. The differences are defined between the results of the shell model and the beam one.

(a) First flapwise eigenmode.

(b) Second flapwise eigenmode.



(c) First edgewise eigenmode.

Figure 6.22: Eigenmode shapes of the beam model.

Figure 6.23: Deformation of the blades under gravity load. The deformation is scaled up with a factor of 500.

Computation of forces

During the FSI simulation, the beam elements are solved in a coordinate system rotating with the blades as shown in Fig. 6.24, so the blades can be modeled as cantilever beams that are clamped at the roots. Linear beam elements are used since the deformation of the blades under the wind load is small. To model the rigid body rotation of the blades, an additional client *beamForces* is implemented which computes the total forces on the beam elements in the rotating coordinate system. The coupling scenario of the FSI problem is shown in Fig. 6.25. In beamForces, the fluid forces $\mathbf{f}$ are added with the gravity forces $\mathbf{f}_g$, and then the results are transformed to the rotating coordinate system with the matrix $\mathbf{R}$. The total forces $\mathbf{f}_t$ is equal to the sum of the fluid forces, the gravity forces and the centrifugal forces $\mathbf{f}_c$ which are constant. Note that the deformations $\mathbf{s}$ are defined on the rotating coordinate system, but they do not have to be transformed because the adapter in OpenFOAM needs also deformations defined on this coordinate system instead of the global one. In fact, beamForces is only a tool for the structure solver implemented externally instead of a physical partition.

FSI simulation and results

For FSI simulation, the linearized mapping algorithm for beam elements is used due to small bending deformation. Mapping happens between connections 2a and 2b as well as between connections 4 and 1. Loose coupling is chosen in combination with linear extrapolation in time, so the connections are wrapped by a single time step loop. The L2 norm of the interface residual regarding the displacements stays between $1 \cdot 10^{-7}$ m and $1 \cdot 10^{-6}$ m during the simulation, i.e. the solution process is stable.

The blade tip displacement and the rotor shaft torque over time are shown in Fig. 6.26. A time series of the simulation is presented in

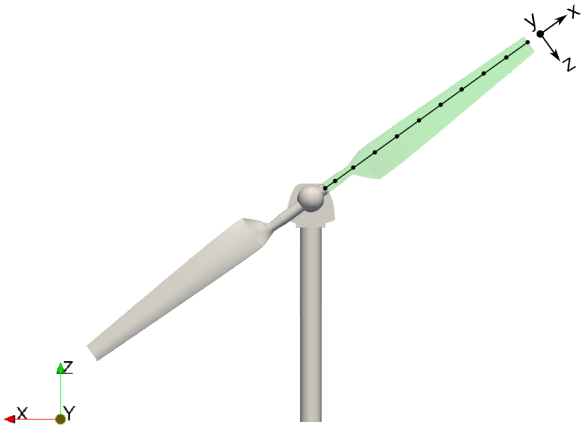


Figure 6.24: The rotating coordinate system defined on one blade.

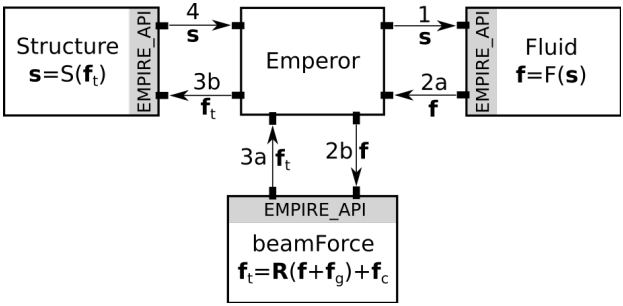


Figure 6.25: Coupling scenario of the FSI simulation of the wind turbine.

Fig. 6.27, where the vortices in the flow fields are visualized by the isosurfaces of the Q-criterion [40].

Compared with the results from the shell model in Fig. 6.20, the following can be concluded:

- The torques are very close but the average tip displacement from the beam model is about 30% larger. Both torques are also close to the one resulting form the CFD simulation, this means the torque is almost not affected by the aeroelasticity of the blades because they are stiff enough.
- The differences in the tip displacements are caused by the differences in the structural properties of both models. This cannot

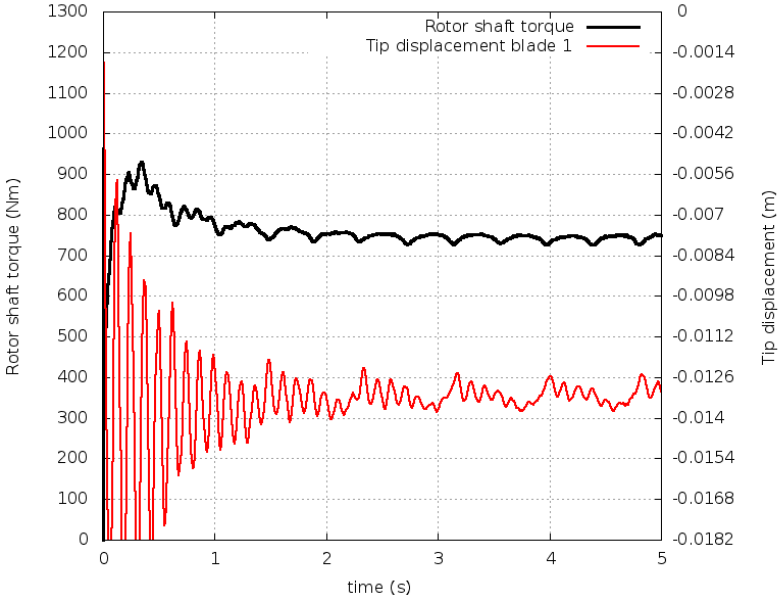
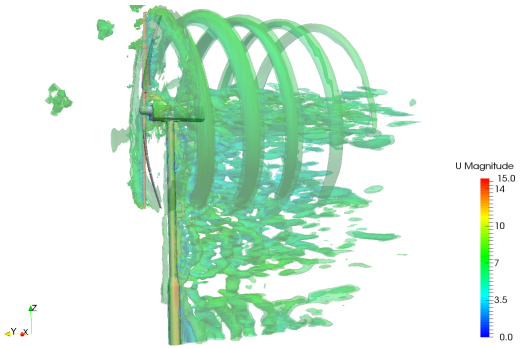


Figure 6.26: Results of the FSI simulation with the beam model.

be improved due to the inaccurate data given in the experiment document and the limited information from the shell model.

- Compared with the shell model with $\approx$ 60 thousands nonlinear shell elements, there are only 10 elements in the beam model. Using beam elements saves not only the modeling but also the computation effort. Given more accurate cross-section analysis tools, the beam model is a good choice for complex beam structures.

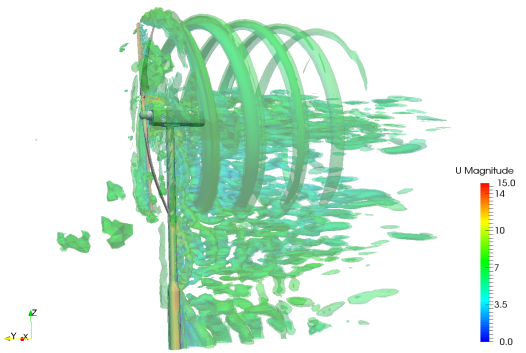
(a) 0.3 s.



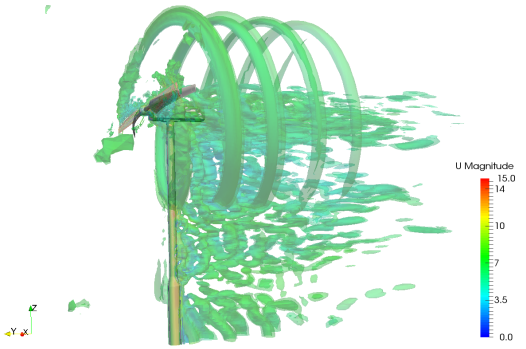
(b) 2.7 s.

(c) 2.95 s.

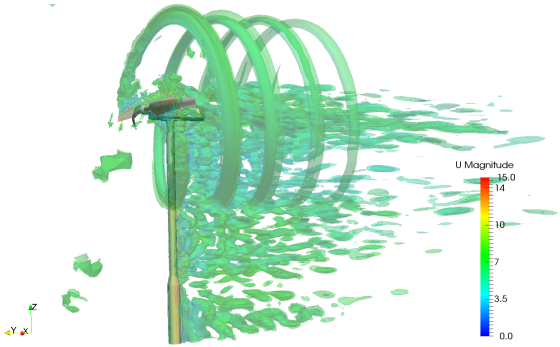
Figure 6.27: Q-criterion isosurface colored by velocity magnitude on the flow field from the FSI simulation. The bending deformation is scaled with a factor of 70.



(d) 3.15 s.



(e) 3.35 s.



(f) 5 s.

Figure 6.27: Q-criterion isosurface colored by velocity magnitude on the flow field from the FSI simulation. The bending deformation is scaled with a factor of 70.

6.3 Shape Optimization of a Prototypical Hydrofoil

A shape optimization example is presented here which aims at demonstrating the application of EMPIRE in coupled problems with more than two partitions and more complex coupling scenarios. The example is shown in Fig. 6.28 where a prototypical hydrofoil attached to a string is put inside a channel flow. The shape of the hydrofoil is described as a NURBS surface with control points. A single design variable controls the shape of the hydrofoil by displacing the control points. The hydrofoil moves vertically due to the lift force to a stable position and the flow becomes steady. The goal of the optimization is to minimize the drag force on the hydrofoil. Numerical models are not validated analytically or by experiments. This example is designed to present the establishment and mechanism of an optimization framework rather than to find the optimal of a certain practical problem.

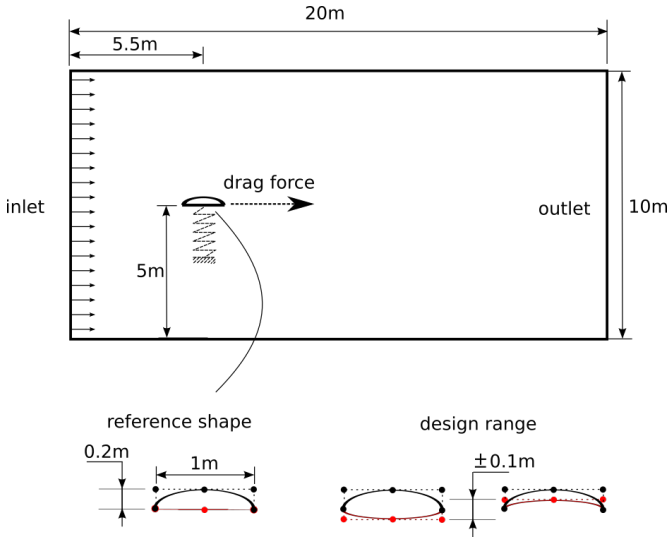


Figure 6.28: Sketch of the shape optimization problem of a prototypical hydrofoil. The control points at the bottom are colored by red which can move according to the design variable.

Three clients are coupled in the co-simulation: OpenFOAM is chosen as the fluid solver, the mechanical system is written in a new client *mechanicalClient* and the optimizer is written in MATLAB which applies the optimization toolbox there.

6.3.1 Fluid Model

The fluid mesh with the reference shape is shown in Fig. 6.29. It is actually a 3D mesh with a constant width of 1 m in the z direction. The width is discretized by only one cell, and the front and back boundaries are set to "empty" in OpenFOAM so that the 3D model is internally solved in 2D.

The flow properties are close to water i.e. the kinematic viscosity is $1 \cdot 10^{-6}\,\mathrm{m}^2/\mathrm{s}$ and the density is $1 \cdot 10^{3}\,\mathrm{kg}/\mathrm{m}^3$. The upper and lower boundaries are set to slip walls and the pressure is set to 0 at the outlet. In order to obtain a steady state in the flow field, the velocity at the inlet is set to a small value as 0.5 m/s. The first-order scheme BDF1 is chosen for time integration. Regarding spatial interpolation, the central difference algorithm is used for the diffusion term while the upwind algorithm is used for the convection term. The PIMPLE algorithm is used for solving the Navier-Stokes equations. The time step is set to 0.1 s which results in a maximum CFL number around 4.5. The total time is 100 s.

A fluid simulation is performed first on the fixed hydrofoil. The evolution of the lift and drag forces is shown in Fig. 6.30. It can be seen that the flow arrives at a steady state before 100 s. The pressure and velocity fields around the hydrofoil at 100 s are shown in Fig. 6.31.

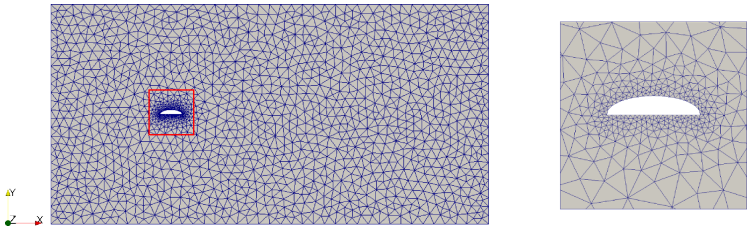


Figure 6.29: The fluid mesh and the zoomed-in region around the hydrofoil.

6.3.2 Structure Model

The structure of the hydrofoil is simply modeled as a mechanical system with one degree of freedom which is the displacement in the vertical direction. The stiffness is set as $K = 1000\,\mathrm{N}/\mathrm{m}$. Since the hydrofoil will move to a stable position, the displacement at the stable position can be computed by $u = f/K$, where f is the lift force from the steady

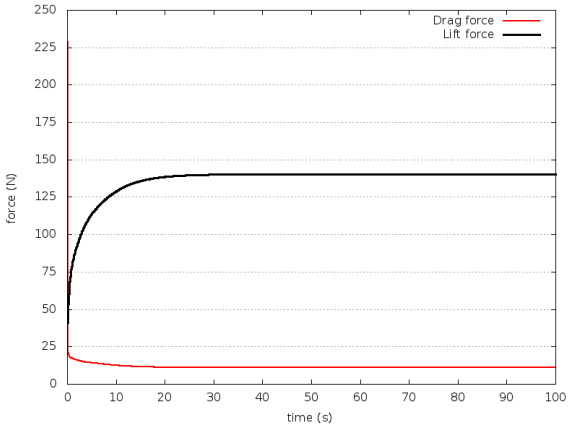


Figure 6.30: Evolution of the lift and drag forces during the CFD simulation.

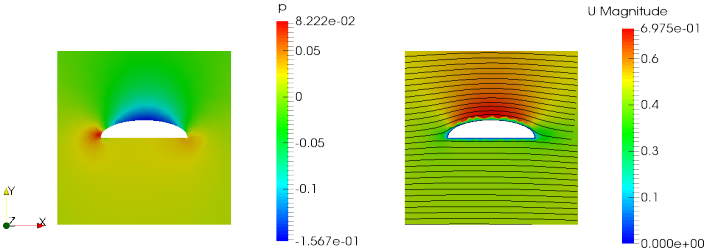


Figure 6.31: Flow fields at 100 s in the CFD simulation.

state flow. The mass is set to $M = 2000\,\text{kg}$. The damping is set as $C = 50\,\text{Ns/m}$ to stabilize the interaction between the flow and the hydrofoil. The values of K, M and C are tuned with several try-runs and the chosen values can lead to a steady state of FSI. The mechanical system can be described by an ordinary differential equation (ODE) as $M\ddot{u} + C\dot{u} + Ku = f$ where the dots over u denote the time derivatives. The ODE is implemented inside the client *mechanicalClient*. The Newmark-β algorithm is used for time integration. The time step and the total time are the same as in the fluid model.

For FSI simulation, mechanicalClient holds a mesh of the wet-surface which is matching to the one from the fluid solver. The input of this client is the forces on the mesh and the output is the displacements on

it. Internally, f is computed by summing up the nodal forces in the vertical direction, and then u is solved and assigned to all mesh nodes.

6.3.3 FSI Simulation

An FSI simulation is performed first with the reference shape. The nearest neighbor mapping algorithm is used since the interface meshes from both clients are matching. Iterative coupling is used in combination with linear extrapolation. The simulation is run for 100 s to guarantee that a steady state is reached.

The evolution of the lift and the elastic forces is shown in Fig. 6.32. The forces become almost equal and constant after 50 seconds, which is a sign of steady state. A series of flow fields at different times are shown in Fig. 6.33.

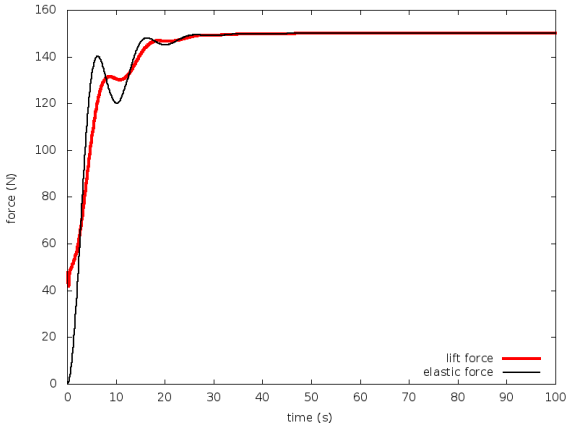
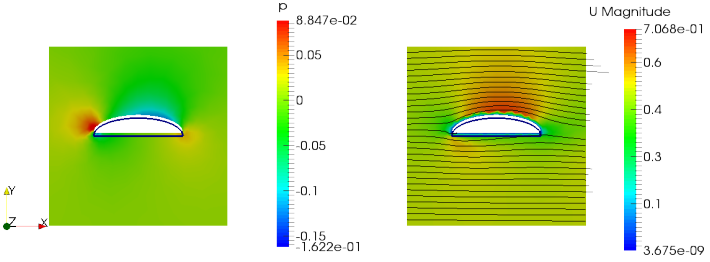


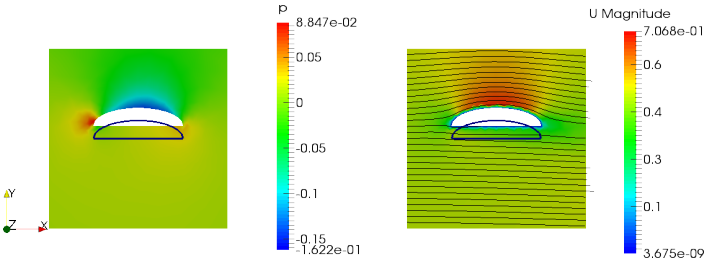
Figure 6.32: Evolution of the lift force and the elastic force of the spring during the FSI simulation.

6.3.4 Optimization

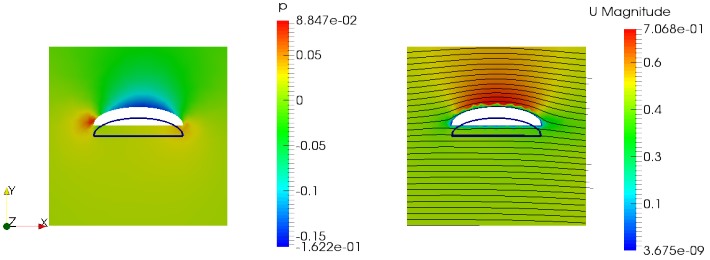
The coupling scenario of the co-simulation is shown in Fig. 6.34, where the connections are defined both statically and dynamically. The coupling is realized by putting the transient FSI problem inside the optimization loop (loop 1). The FSI simulation is set up in the same way as shown in Section 6.3.3, except that the shape of the hydrofoil in the fluid is changed.



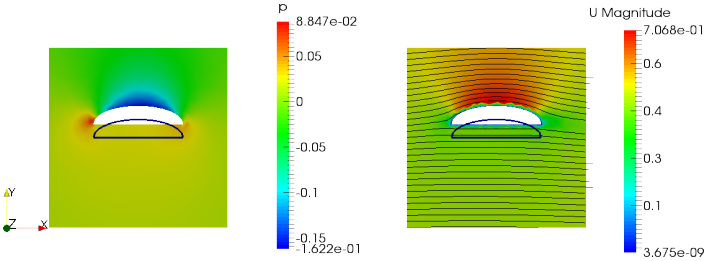
(a) 1 s.



(b) 5 s.



(c) 10 s.



(d) 50 s.

Figure 6.33: Flow field during the FSI simulation at different times.

Optimizer

The optimizer is written in MATLAB. It owns a mesh of the hydrofoil modeled by a NURBS surface. It calls a MATLAB routine fmincon to minimize a MATLAB function which triggers one FSI simulation each time it is called. After fmincon finds the minimum, the optimizer will inform Emperor the convergence which will be then broadcast to the other clients, then the co-simulation is finished.

Shape parametrization. The shape of the hydrofoil in 2D is modeled by one NURBS curve with 9 control points, the first and the last of which are coincided. Their positions are shown in Fig. 6.28. The parameters of the NURBS curve are the same as those used for describing a circle with $2nd$ order polynomials. The knot vector is $(0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1)$ and the weights are $(\frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2})$. The shape in 3D can be modeled by a NURBS surface with 18 control points by extruding the NURBS curve in the third direction with linear polynomials. The parameters of the NURBS surface are sent from the the optimizer to Emperor as an isogeometric mesh (represented by the class IGAMesh in Emperor).

Design variable. The shape is changed by moving the control points at the bottom in the vertical direction. These control points are moved with the same displacement, while the other control points are fixed in space. The displacement denoted by q is the single design variable of the optimization. It is equal to 0 at the reference shape. The range of q is $[-0.1, 0.1]$. The extreme shapes at the upper and lower bounds are shown in Fig. 6.28. The displacements of all the control points are denoted by $\mathbf{u}_{sh}$, which can be derived from q (the displacement on a control point is either equal to q or 0). Note that these displacements should be differentiated from the displacements of the hydrofoil during FSI simulation, which are denoted by $\mathbf{u}$.

Mesh morphing. Mesh morphing is the method of changing the shape of a mesh by changing certain parameters. The mesh morphing of this problem is realized by the isogeometric mortar mapper which maps $\mathbf{u}_{sh}$ from the isogeometric mesh to the finite volume surface mesh, which is used in connection 1. Note that the total displacements $\mathbf{u}_t = \mathbf{u}_{sh} + \mathbf{u}$ determine the actual shape and location of the hydrofoil in the fluid domain. For the mechanical system mesh morphing is not needed because it contains only a single mass point.

Objective. The objective is the drag force p at the steady state, i.e. the goal is to minimize p which is dependent on q. p is of type Signal which is internally a single float number. It is computed out of the fluid

forces **f**.

Minimization. The routine fmincon in MATLAB is used, which is able to find the minimal of a scalar objective regarding multiple design variables with non-linear equality and inequality constraints. For the problem here, the initial value, upper and lower bounds of q are passed into fmincon. Besides, a function $p = P(q)$ has to be passed into fmincon as well. fmincon will run the function iteratively by feeding a new value of q into it and obtaining a new value of p from it. After each iteration, fmincon will calculate the update of the design variable Δq based on certain algorithms. For this problem, fmincon is configured as follows: the interior point algorithm [23] is used which employs sequential quadratic programming (SQP) and trust regions; the derivatives are approximated by finite difference; the termination tolerance of the objective is set to $2 \cdot 10^{-4}$ N; the initial value of q is 0.1, the upper and lower bounds of q are 0.1 and -0.1. The function $p = P(q)$ is also written in MATLAB. It first computes $\mathbf{u}_{sh}$ out of q and then sends $\mathbf{u}_{sh}$ to Emperor to trigger one FSI simulation. After the FSI simulation is finished, it receives p from OpenFOAM and returns it to fmincon.

Optimization and results

Before optimization, FSI simulations with prescribed shape updates are precomputed to obtain a first impression of the relation between the objective and the design variable, as shown in Fig. 6.36. It can be foreseen that the drag force has a minimum inside the interval $[-0.06, -0.02]$.

The same configuration of the FSI simulation in Section 6.3.3 is used in the optimization. So the steady state of the FSI problem is solved in a transient simulation with nested time step and iterative coupling loops. The overall coupling scenario is set up by wrapping the nested loops for the FSI coupling inside an optimization loop.

The optimization is performed and the optimal is found after 12 optimization iterations. The outputs of fmincon are shown in Table 6.3. The "FSI-count" is the number of FSI simulations performed, which is the same as the number of optimization iterations. The "opt-count" is step number of the optimization algorithm in fmincon. And each step contains two FSI simulations since the finite difference method is used to compute the "optimality" which is actually the derivative. The drag forces dependent on the shape updates are drawn in Fig. 6.37, where the finite difference evaluation of the derivative can be clearly seen from the curve of the optimization process. Different shapes during the optimization process are drawn in the Figure. The optimal design

is found to be at -0.04908, in which case the steady state flow fields at 100 s are shown in Fig. 6.35.

Opt.-count	FSI-count	Design	Objective	Step length	Optimality
1	1	0.1	17.62		
	2	0.099	17.568	0.2	5.212e+01
2	3	-0.1	10.86		
	4	-0.099	10.828	0.07671	8.434e+01
3	5	-0.02329	10.565		
	6	-0.02429	10.534	0.03773	2.053e+01
4	7	-0.06102	9.9879		
	8	-0.06202	9.998	0.00920	1.013e+01
5	9	-0.05182	9.9316		
	10	-0.05282	9.9345	0.00374	2.925e+00
6	11	-0.04808	9.9283		
	12	-0.04908	9.9280	0.00029	2.516e-01

Table 6.3: Statistics of the optimization process.

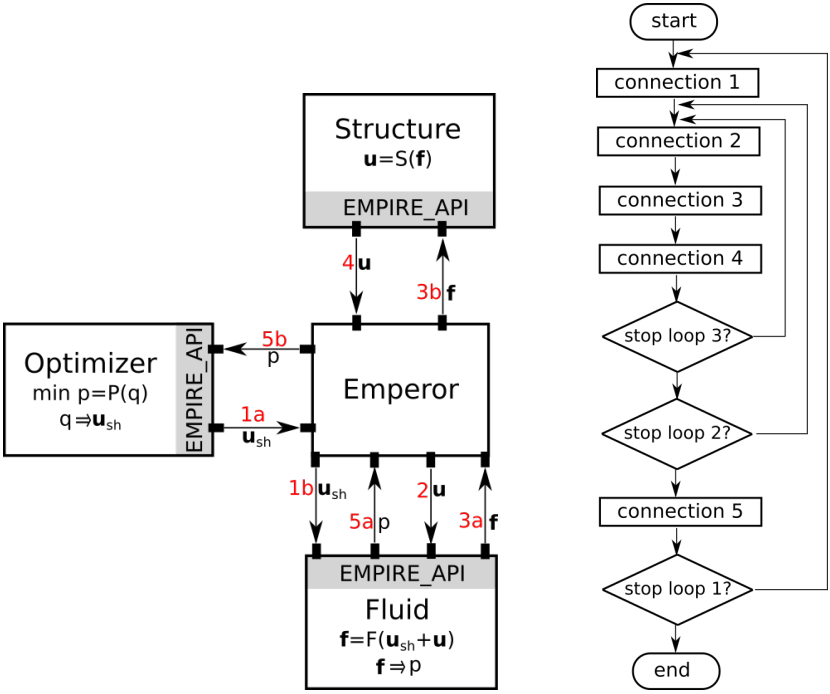


Figure 6.34: Coupling scenario of the optimization problem. (a) Statical diagram. (b) Dynamical diagram.

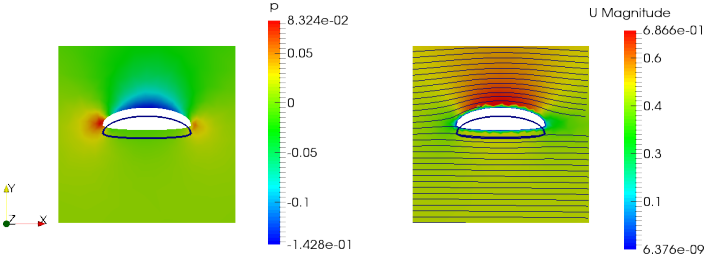


Figure 6.35: Flow fields at the last time step of the FSI simulation with the optimized shape.

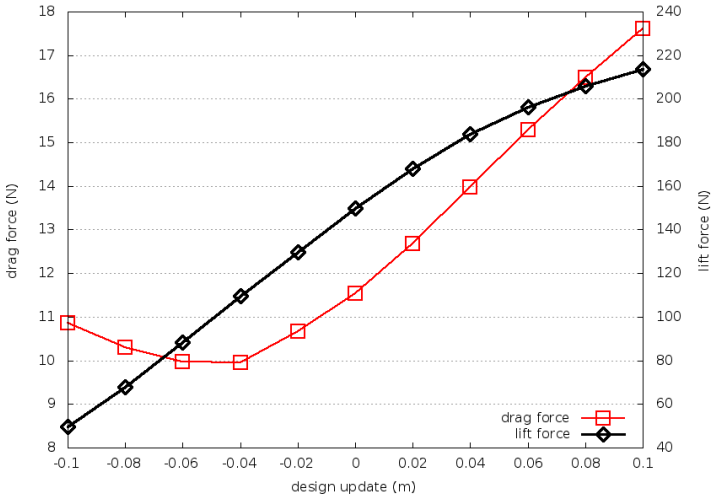


Figure 6.36: Lift and drag forces at the stead state of the FSI simulations with prescribed design updates.

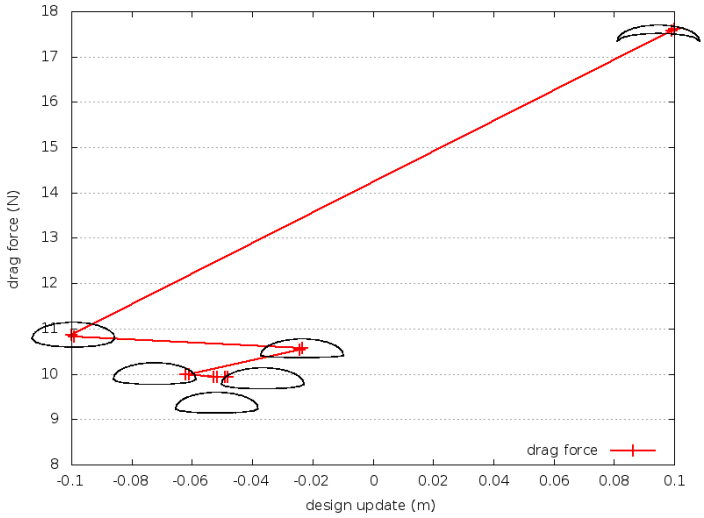


Figure 6.37: Design and objective during the optimization process. Update of the shapes is also drawn.

6.4 Summary

During the development of EMPIRE, it has been successfully validated by benchmark cases as well as applied in practical multiphysics co-simulations including FSI, FSI plus control and generator, and optimization of FSI. This chapter presents three examples of them.

The first example is the FSI simulation of a flat membrane in a wind tunnel. It is modeled as a zero-thickness object in the fluid domain, and the upper and lower surfaces are two separate interfaces defined for coupling. Compared with the given experimental measurements on the displacements, there is a good agreement in the mean values and the standard deviations.

The second example is the FSI simulation of the NREL phase VI wind turbine. Compared with another work, each blade is modeled by only 10 linear beam elements instead of 60 thousand nonlinear shell elements. The mapping algorithm for linear beam elements presented in Chapter 5 is applied. The beam elements are modeled in a coordinate system rotating with the blade, and a third client helps to transform the forces to this coordinate system. As a result, three programs are coupled in the co-simulation. The simulation results from the shell model and those from the beam model are compared. They have very close torque but the beam model has considerably larger displacements. The main reason is the lack of known structural properties to make the beam model approach to the shell model.

The third example is the shape optimization of a prototypical 2D hydrofoil. It locates inside a channel flow and is connected to a spring, which results in steady state FSI. This example aims at demonstrating the use of EMPIRE in a co-simulation with more partitions and a more complex coupling scenario. The shape of the hydrofoil is described by NURBS parameters and is morphed by one design variable. The objective is the drag force on the hydrofoil. The optimizer is implemented in MATLAB so that existing optimization routines can be reused.

Chapter 7

Conclusion and Outlook

This thesis starts with the computational models for fluid, structure and FSI problems. A general multiphysics problem can contain two or more than two partitions. The partitioned strategy of solving multiphysics problems results into co-simulation, where the governing equations of the individual partitions are rewritten into black box solvers, and the coupling conditions specify the relations between the inputs and outputs of different solvers.

A software environment named EMPIRE is newly developed to support co-simulations for general multiphysics problems. The structure is based on client-server model, where the black box solvers exchange data with each other through an additional server program. The server is also in charge of performing data operations on interface data. A library with API is provided for the clients where the types of interface data are defined and the send/receive functions of them are also provided. The biggest advantage of EMPIRE compared with other existing co-simulation environments lies in the flexibility, i.e. the user is allowed to set up an arbitrary co-simulation scenario which contains loops and sequences of connections.

Another main focus of this work is the mapping technique, which applies coupling conditions on non-matching meshes at the fluid-structure interface. Three mapping algorithms for surface meshes are investigated namely nearest element interpolation, standard mortar method and dual mortar method. It is found that the mortar methods have inconsistency problem at curved edges, which is solved by the newly developed enforcing consistency approach. When doing conservative traction mapping, the dual mortar method can give slight oscillations and nearest element

interpolation can give large oscillations. In case that only nearest element interpolation is available, conservative mapping can still be used if the structure is relatively stiff and if accurate stress distribution close to the wet surface is not pursued. Otherwise, direct mapping can be used. In case that the mortar methods are available, the dual mortar method is generally a good choice. The standard mortar method can be used if slight oscillations are not accepted.

Mapping with beam elements require a different technique than mapping with surface meshes, where nodal displacements on the surface mesh are computed according to the DOFs of the 1D beam elements. The mapping algorithms are based on the assumptions that the cross sections remain rigid and the beam elements locate at the beam axis. The interpolation of the rigid body motion of a cross section according to the DOFs of a beam element is realized either with the linearized or the co-rotating algorithm. Different tests have shown that the linearized algorithm works well for beams with small deformation and the co-rotating algorithm can handle large displacements and rotations.

Three co-simulation examples are presented including FSI simulation of a flat membrane in a wind tunnel, FSI simulation of a wind turbine whose blades are modeled by beam elements, and shape optimization of a prototypical hydrofoil. Other examples especially with respect to FSI with control can be found in the work of [120]. These examples have not only validated the coupling functionalities of EMPIRE, but have also demonstrated its capability in supporting flexible co-simulation scenarios.

Due to the time limit of this PhD work, EMPIRE has only been applied for a limited number of multiphysics problems. As the case for all software, EMPIRE can only be further developed with more practical applications. Nevertheless, since the requirements of flexibility and extendability are well realized by the data structures, it has big potential in solving new types of multiphysics problems. As an open-source project with application of modern software techniques, it is hoped that EMPIRE can be used by more research groups in the future as a platform of algorithm development and multiphysics co-simulation.

Appendix A

Proof of Inconsistency of Mortar Methods

A.1 Standard Mortar Method

Let $\mathbf{1}_\mathrm{f}$ and $\mathbf{1}_\mathrm{s}$ be the constant discrete fields equal to one on the fluid and the structure mesh respectively. If the direct displacement mapping is consistent, $\mathbf{U}_\mathrm{f} = \mathbf{1}_\mathrm{f}$ and $\mathbf{U}_\mathrm{s} = \mathbf{1}_\mathrm{s}$ should satisfy (4.20). Since $\mathbf{N}_\mathrm{f}^\mathrm{T}\mathbf{1}_\mathrm{f} = 1_\mathrm{f}(\mathbf{x})$ and $\mathbf{N}_\mathrm{s}^\mathrm{T}\mathbf{1}_\mathrm{s} = 1_\mathrm{s}(\mathbf{x})$, we have in (4.20)

$$\mathrm{LHS} = \int_{\Gamma_\mathrm{f}} \mathbf{N}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{f}\,\mathrm{d}\Gamma_\mathrm{f} = \int_{\Gamma_\mathrm{f}} \mathbf{N}_\mathrm{f}(\mathbf{x})1_\mathrm{f}(\mathbf{x})\,\mathrm{d}\Gamma_\mathrm{f}, \tag{A.1a}$$

$$\mathrm{RHS} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{s}\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{f}(\mathbf{x})1_\mathrm{s}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}. \tag{A.1b}$$

If the conservative traction mapping is consistent, $\mathbf{P}_\mathrm{s} = \mathbf{1}_\mathrm{s}$ and $\mathbf{P}_\mathrm{f} = \mathbf{1}_\mathrm{f}$ should satisfy (4.24). In (4.24) we have

$$\mathrm{LHS} = \mathbf{M}_{\mathrm{ss}}\mathbf{1}_\mathrm{s} = \int_{\Gamma_\mathrm{s}} \mathbf{N}_\mathrm{s}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{s}\,\mathrm{d}\Gamma_\mathrm{s} = \int_{\Gamma_\mathrm{s}} \mathbf{N}_\mathrm{s}(\mathbf{x})1_\mathrm{s}(\mathbf{x})\,\mathrm{d}\Gamma_\mathrm{s}, \tag{A.2a}$$

$$\begin{aligned}\mathrm{RHS} = \mathbf{M}_{\mathrm{fs}}^\mathrm{T}\mathbf{1}_\mathrm{f} &= \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{s}(\mathbf{x})\mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{f}\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}} \\ &= \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{s}(\mathbf{x})1_\mathrm{f}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}.\end{aligned} \tag{A.2b}$$

In (A.1) and (A.2), generally LHS $\neq$ RHS except when the integration domains on both sides are the same, so both methods are inconsistent. However, the discrepancy between both integration domains is limited

since they are discretizations of the same continuous geometry. Therefore, the direct and the conservative mapping with the standard mortar algorithm are not exactly but approximately consistent.

A.2 Dual Mortar Method

If the direct displacement mapping is consistent, $\mathbf{U}_\mathrm{f} = \mathbf{1}_\mathrm{f}$ and $\mathbf{U}_\mathrm{s} = \mathbf{1}_\mathrm{s}$ should satisfy (4.29). In (4.29) we have

$$\mathrm{LHS} = \int_{\Gamma_\mathrm{f}} \hat{\mathbf{N}}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{f}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{f}\,\mathrm{d}\Gamma_\mathrm{f} = \int_{\Gamma_\mathrm{f}} \hat{\mathbf{N}}_\mathrm{f}(\mathbf{x})1_\mathrm{f}(\mathbf{x})\,\mathrm{d}\Gamma_\mathrm{f}, \tag{A.3a}$$

$$\mathrm{RHS} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \hat{\mathbf{N}}_\mathrm{f}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{s}\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \hat{\mathbf{N}}_\mathrm{f}(\mathbf{x})1_\mathrm{s}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}. \tag{A.3b}$$

If the conservative traction mapping is consistent, $\mathbf{P}_\mathrm{s} = \mathbf{1}_\mathrm{s}$ and $\mathbf{P}_\mathrm{f} = \mathbf{1}_\mathrm{f}$ should satisfy (4.33). In (4.33) we have

$$\mathrm{LHS} = \mathbf{M}_\mathrm{ss}\mathbf{1}_\mathrm{s} = \int_{\Gamma_\mathrm{s}} \mathbf{N}_\mathrm{s}(\mathbf{x})\mathbf{N}_\mathrm{s}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{s}\,\mathrm{d}\Gamma_\mathrm{s} = \int_{\Gamma_\mathrm{s}} \mathbf{N}_\mathrm{s}(\mathbf{x})1_\mathrm{s}(\mathbf{x})\,\mathrm{d}\Gamma_\mathrm{s}, \tag{A.4a}$$

$$\begin{aligned}\mathrm{RHS} &= \hat{\mathbf{M}}_\mathrm{fs}^\mathrm{T}\mathbf{D}_\mathrm{ff}^{-1}\mathbf{M}_\mathrm{ff}\mathbf{1}_\mathrm{f} = \hat{\mathbf{M}}_\mathrm{fs}^\mathrm{T}\mathbf{1}_\mathrm{f} = \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{s}(\mathbf{x})\hat{\mathbf{N}}_\mathrm{f}^\mathrm{T}(\mathbf{x})\mathbf{1}_\mathrm{f}\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}} \\ &= \int_{\Gamma_{\mathrm{s}\to\mathrm{f}}} \mathbf{N}_\mathrm{s}(\mathbf{x})1_\mathrm{f}(\mathbf{x})\,\mathrm{d}\Gamma_{\mathrm{s}\to\mathrm{f}}.\end{aligned} \tag{A.4b}$$

In (A.3) and (A.4), generally LHS $\neq$ RHS except when the integration domains on both sides are the same. So the same conclusion can be derived as in A.1 for the dual mortar method.

Bibliography

[1] N. Aghajari and M. Schäfer. Efficient shape optimization for fluid–structure interaction problems. *Journal of Fluids and Structures*, 57:298–313, 2015.

[2] R. Ahrem, A. Beckert, and H. Wendland. Recovering rotations in aeroelasticity. *Journal of fluids and structures*, 23(6):874–884, 2007.

[3] A. Apostolatos, R. Wüchner, and K.-U. Bletzinger. Non-matching grid transfer schemes for partitioned fluid-structure interaction simulations using isogeometric analysis. In *5th GACM Colloquium on Computational Mechanics, Hamburg*, 2013.

[4] M. Arnold, P. W. Cheng, F. Biskup, et al. Simulation of fluid-structure-interaction on tidal current turbines based on coupled multibody and cfd methods. In *The Twenty-third International Offshore and Polar Engineering Conference*. International Society of Offshore and Polar Engineers, 2013.

[5] M. N. D. Barcelos Júnior. *Aeroelastic optimization methodology for viscous and turbulent flows*. PhD thesis, University of Colorado, 2007.

[6] K. Bathe, C. Nitikitpaiboon, and X. Wang. A mixed displacement-based finite element formulation for acoustic fluid-structure interaction. *Computers & Structures*, 56(2):225–237, 1995.

[7] K.-J. Bathe. *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[8] K.-J. Bathe and S. Bolourchi. Large displacement analysis of three-dimensional beam structures. *International Journal for Numerical Methods in Engineering*, 14(7):961–986, 1979.

[9] M. Bathe and R. Kamm. A fluid-structure interaction finite element analysis of pulsatile blood flow through a compliant stenotic artery. *Journal of Biomechanical Engineering*, 121(4):361–369, 1999.

[10] Y. Bazilevs, V. Calo, Y. Zhang, and T. J. Hughes. Isogeometric fluid–structure interaction analysis with applications to arterial blood flow. *Computational Mechanics*, 38(4-5):310–322, 2006.

[11] Y. Bazilevs, M.-C. Hsu, J. Kiendl, R. Wüchner, and K.-U. Bletzinger. 3D simulation of wind turbine rotors at full scale. Part II: Fluid–structure interaction modeling with composite blades. *International Journal for Numerical Methods in Fluids*, 65(1-3):236–253, 2011.

[12] A. Beckert and H. Wendland. Multivariate interpolation for fluid-structure-interaction problems using radial basis functions. *Aerospace Science and Technology*, 5(2):125–134, 2001.

[13] A. V. Belver, A. Foces Mediavilla, A. Lorenzana Iban, and R. Rossi. Fluid-structure coupling analysis and simulation of a slender composite beam. *Science and Engineering of Composite Materials*, 17(1):47–77, 2010.

[14] T. Belytschko, W. K. Liu, B. Moran, and K. Elkhodary. *Nonlinear finite elements for continua and structures*. John Wiley & Sons, 2013.

[15] C. Bernardi, Y. Maday, and A. T. Patera. A new nonconforming approach to domain decomposition: the mortar element method. *Nonlinear Partial Differential Equations and Their Applications*, 1994.

[16] J. P. A. A. Blasques, R. Bitsche, V. Fedorov, and M. A. Eder. Applications of the beam cross section analysis software (becas). *Proceedings of the 26th Nordic Seminar on Computational Mechanics*, pages 46–49, 2013.

[17] F. J. Blom. A monolithical fluid-structure interaction algorithm applied to the piston problem. *Computer Methods in Applied Mechanics and Engineering*, 167(3):369–391, 1998.

[18] F. M. Bos. *Numerical simulations of flapping foil and wing aerodynamics: Mesh deformation using radial basis functions*. PhD thesis, Delft University of Technology, 2010.

[19] C. Bottasso, F. Campagnolo, A. Croce, and C. Tibaldi. Optimization-based study of bend–twist coupled rotor blades for passive and integrated passive/active load alleviation. *Wind Energy*, 16(8):1149–1166, 2013.

[20] K.-H. Brakhage, W. Dahmen, and P. Lamby. *A unified approach to the modeling of airplane wings and numerical grid generation using B-spline representations*. Springer, 2010.

[21] S. C. Brenner and R. Scott. *The mathematical theory of finite element methods*, volume 15. Springer, 2008.

[22] B. Bruegge and A. H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns and Java-(Required)*. Prentice Hall, 2004.

[23] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999.

[24] L. Cavagna, P. Masarati, and G. Quaranta. Coupled multibody/computational fluid dynamics simulation of maneuvering flexible aircraft. *Journal of Aircraft*, 48(1):92–106, 2011.

[25] H. Chen, W. Yu, and M. Capellaro. A critical assessment of computer tools for calculating composite wind turbine blade properties. *Wind Energy*, 13(6):497–516, 2010.

[26] X. Chen, G. Zha, and M. Yang. Numerical simulation of 3-d wing flutter with fully coupled fluid–structural interaction. *Computers & fluids*, 36(5):856–867, 2007.

[27] Z. Chen, Y. Wu, and X. Sun. Research on the added mass of open-type one-way tensioned membrane structure in uniform flow. *Journal of Wind Engineering and Industrial Aerodynamics*, 137:69–77, 2015.

[28] J. Chung and G. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-α method. *Journal of applied mechanics*, 60(2):371–375, 1993.

[29] T. Cichosz and M. Bischoff. Consistent treatment of boundaries with mortar contact formulations using dual lagrange multipliers. *Computer Methods in Applied Mechanics and Engineering*, 200(9):1317–1332, 2011.

[30] M. Cordero-Gracia, M. Gómez, and E. Valero. A radial basis function algorithm for simplified fluid-structure data transfer. *International Journal for Numerical Methods in Engineering*, 99(12):888–905, 2014.

[31] J. J. Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[32] A. de Boer, A. H. van Zuijlen, and H. Bijl. Comparison of conservative and consistent approaches for the coupling of non-matching meshes. *Computer Methods in Applied Mechanics and Engineering*, 197(49):4284–4297, 2008.

[33] A. De Rosis, G. Falcucci, S. Ubertini, and F. Ubertini. A coupled lattice boltzmann-finite element approach for two-dimensional fluid–structure interaction. *Computers & Fluids*, 86:558–568, 2013.

[34] J. Degroote. *Development of algorithms for the partitioned simulation of strongly coupled fluid-structure interaction problems.* PhD thesis, Ghent University, 2010.

[35] J. Degroote, P. Bruggeman, R. Haelterman, and J. Vierendeels. Stability of a coupling technique for partitioned solvers in FSI applications. *Computers & Structures*, 86(23):2224–2234, 2008.

[36] J. Degroote, R. Haelterman, S. Annerel, P. Bruggeman, and J. Vierendeels. Performance of partitioned procedures in fluid–structure interaction. *Computers & structures*, 88(7):446–457, 2010.

[37] J. Degroote, M. Hojjat, E. Stavropoulou, R. Wüchner, and K.-U. Bletzinger. Partitioned solution of an unsteady adjoint for strongly coupled fluid-structure interactions and application to parameter identification of a one-dimensional problem. *Structural and Multidisciplinary Optimization*, 47(1):77–94, 2013.

[38] J. Donea, S. Giuliani, and J. Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer methods in applied mechanics and engineering*, 33(1):689–723, 1982.

[39] J. Donea, A. Huerta, J.-P. Ponthot, and A. Rodriguez-Ferran. Encyclopedia of computational mechanics vol. 1: Fundamentals., chapter 14: Arbitrary lagrangian-eulerian methods, 2004.

[40] Y. Dubief and F. Delcayre. On coherent-vortex identification in turbulence. *Journal of turbulence*, 1(1):011–011, 2000.

[41] C. Farhat and M. Lesoinne. Improved staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems. *Center for Aerospace Structures-97-11, University of Colorado, Boulder, Colorado., AIAA Journal*, 1997.

[42] C. Farhat, M. Lesoinne, and P. Le Tallec. Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: momentum and energy conservation, optimal discretization and application to aeroelasticity. *Computer Methods in Applied Mechanics and Engineering*, 157(1):95–114, 1998.

[43] C. Farhat, K. G. Van der Zee, and P. Geuzaine. Provably second-order time-accurate loosely-coupled solution algorithms for transient nonlinear computational aeroelasticity. *Computer methods in applied mechanics and engineering*, 195(17):1973–2001, 2006.

[44] P. Farrell and J. Maddison. Conservative interpolation between volume meshes by local galerkin projection. *Computer Methods in Applied Mechanics and Engineering*, 200(1):89–100, 2011.

[45] C. Felippa and B. Haugen. A unified formulation of small-strain corotational finite elements: I. theory. *Computer Methods in Applied Mechanics and Engineering*, 194(21):2285–2335, 2005.

[46] C. Felippa, K. Park, and M. Ross. A classification of interface treatments for FSI. In *Fluid Structure Interaction II*, pages 27–51. Springer, 2010.

[47] C. A. Felippa, K. Park, and C. Farhat. Partitioned analysis of coupled mechanical systems. *Computer methods in applied mechanics and engineering*, 190(24):3247–3270, 2001.

[48] M. Á. Fernández and M. Moubachir. A Newton method using exact Jacobians for solving fluid–structure coupling. *Computers & Structures*, 83(2):127–142, 2005.

[49] M. Firl. *Optimal Shape Design of Shell Structures*. PhD thesis, Technische Universität München, 2010.

[50] R. Fisch. *Code Verification of Partitioned FSI Environments for Lightweight Structures*. PhD thesis, Technische Universität München, 2014.

[51] M. Fischer. *Finite Element Based Simulation, Design and Control of Piezoelectric and Lightweight Smart Structures*. PhD thesis, Technische Universität München, 2013.

[52] Functional mock-up interface. `https://www.fmi-standard.org/`. [Online; accessed 2015-09-18].

[53] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer graphics: principles and practice (2nd ed.)*. Addison-Wesley, 1990.

[54] C. Förster, W. A. Wall, and E. Ramm. Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows. *Computer methods in applied mechanics and engineering*, 196(7):1278–1293, 2007.

[55] T. Gallinger. *Effiziente Algorithmen zur partitionierten Lösung stark gekoppelter Probleme der Fluid-Struktur-Wechselwirkung*. PhD thesis, Technische Universität München, 2010.

[56] T. Gallinger, A. Kupzok, U. Israel, K. Bletzinger, and R. Wuchner. A computational environment for membrane-wind interaction. In *International Workshop on Fluid-Structure Interaction. Theory, Numerics and Applications*, page 97. kassel university press GmbH, 2009.

[57] B. Gatzhammer. *Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions*. PhD thesis, Technische Universität München, 2014.

[58] B. Gatzhammer, M. Mehl, and T. Neckel. A coupling environment for partitioned multiphysics simulations applied to fluid-structure interaction scenarios. *Procedia Computer Science*, 1(1):681–689, 2010.

[59] M. Glück, M. Breuer, F. Durst, A. Halfmann, and E. Rank. Computation of fluid–structure interaction on lightweight structures. *Journal of Wind Engineering and Industrial Aerodynamics*, 89(14):1351–1368, 2001.

[60] M. Glück, M. Breuer, F. Durst, A. Halfmann, and E. Rank. Computation of wind-induced vibrations of flexible shells and membranous structures. *Journal of Fluids and Structures*, 17(5):739–765, 2003.

[61] M. Hand, D. Simms, L. Fingersh, D. Jager, J. Cotrell, S. Schreck, and S. Larwood. Unsteady aerodynamics experiment phase VI: wind tunnel test configurations and available data campaigns. *NREL/TP-500-29955*, 2001.

[62] M. O. L. Hansen, J. N. Sørensen, S. Voutsinas, N. Sørensen, and H. A. Madsen. State of the art in wind turbine aerodynamics and aeroelasticity. *Progress in aerospace sciences*, 42(4):285–330, 2006.

[63] M. Heil. An efficient solver for the fully coupled solution of large-displacement fluid–structure interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 193(1):1–23, 2004.

[64] C. Hirt, A. A. Amsden, and J. Cook. An arbitrary lagrangian-eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14(3):227–253, 1974.

[65] M. Hojjat. *Node-based parametrization for shape optimal design*. PhD thesis, Technische Universität München, 2015.

[66] M. Hojjat, E. Stavropoulou, T. Gallinger, U. Israel, R. Wüchner, and K.-U. Bletzinger. Fluid-structure interaction in the context of shape optimization and computational wind engineering. In *Fluid Structure Interaction II*, pages 351–381. Springer, 2010.

[67] J. D. Holmes. *Wind loading of structures*. CRC Press, 2015.

[68] J. Hron and S. Turek. *A monolithic FEM/multigrid solver for an ALE formulation of fluid-structure interaction with applications in biomechanics*. Springer, 2006.

[69] M.-C. Hsu and Y. Bazilevs. Fluid–structure interaction modeling of wind turbines: simulating the full machine. *Computational Mechanics*, 50(6):821–833, 2012.

[70] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer methods in applied mechanics and engineering*, 194(39):4135–4195, 2005.

[71] B. M. Irons and R. C. Tuck. A version of the aitken accelerator for computer iteration. *International Journal for Numerical Methods in Engineering*, 1(3):275–277, 1969.

[72] R. I. Issa. Solution of the implicitly discretised fluid flow equations by operator-splitting. *Journal of computational physics*, 62(1):40–65, 1986.

[73] R. Jaiman, X. Jiao, P. Geubelle, and E. Loth. Conservative load transfer along curved fluid–solid interface with non-matching meshes. *Journal of Computational Physics*, 218(1):372–397, 2006.

[74] H. Jasak, A. Jemcov, and Z. Tukovic. Openfoam: A c++ library for complex physics simulations. In *International workshop on coupled methods in numerical dynamics*, volume 1000, pages 1–20, 2007.

[75] H. Jasak and Z. Tukovic. Automatic mesh motion for the unstructured finite volume method. *Transactions of FAMENA*, 30(2):1–20, 2006.

[76] W. Joppich and M. Kürschner. Mpccia tool for the simulation of coupled applications. *Concurrency and computation: Practice and Experience*, 18(2):183–192, 2006.

[77] C. Kassiotis, A. Ibrahimbegovic, R. Niekamp, and H. G. Matthies. Nonlinear fluid–structure interaction problem. Part I: implicit partitioned algorithm, nonlinear stability proof and validation examples. *Computational Mechanics*, 47(3):305–323, 2011.

[78] R. Kindmann and M. Kraus. *Steel Structures: Design using FEM*. Ernst&Sohn, 2004.

[79] T. Klöppel, A. Popp, U. Küttler, and W. A. Wall. Fluid–structure interaction for non-conforming interfaces based on a dual mortar formulation. *Computer Methods in Applied Mechanics and Engineering*, 200(45):3111–3126, 2011.

[80] D. A. Knoll and D. E. Keyes. Jacobian-free Newton–Krylov methods: a survey of approaches and applications. *Journal of Computational Physics*, 193(2):357–397, 2004.

[81] S. Krenk. *Non-linear modeling and analysis of solids and structures*. Cambridge University Press, 2009.

[82] D. Kuhl and M. Crisfield. Energy-conserving and decaying algorithms in non-linear structural dynamics. *International journal for numerical methods in engineering*, 45(5):569–599, 1999.

[83] P. K. Kundu and I. M. Cohen. *Fluid mechanics (3rd ed.)*. Elsevier, 2004.

[84] U. Küttler and W. A. Wall. Vector extrapolation for strong coupling fluid-structure interaction solvers. *Journal of Applied Mechanics*, 76(2):021205, 2009.

[85] Y. Li. *Coupled computational fluid dynamics/multibody dynamics method with application to wind turbine simulations*. PhD thesis, University of Iowa, 2014.

[86] Y. Li, K.-J. Paik, T. Xing, and P. M. Carrica. Dynamic overset cfd simulations of wind turbine aerodynamics. *Renewable Energy*, 37(1):285–298, 2012.

[87] Z. Li and L. Vu-Quoc. A mixed co-rotational 3D beam element formulation for arbitrarily large rotations. *Advanced Steel Construction*, 6(2):767–787, 2010.

[88] G. Link, M. Kaltenbacher, M. Breuer, and M. Döllinger. A 2d finite-element scheme for fluid–solid–acoustic interactions and its application to human phonation. *Computer Methods in Applied Mechanics and Engineering*, 198(41):3321–3334, 2009.

[89] F. Liu, J. Cai, Y. Zhu, H. Tsai, and A. F. Wong. Calculation of wing flutter by a coupled fluid-structure method. *Journal of Aircraft*, 38(2):334–342, 2001.

[90] D. J. Malcolm and D. L. Laird. Extraction of equivalent beam properties from blade models. *Wind Energy*, 10(2):135–157, 2007.

[91] J. Marshall and M. Imregun. A review of aeroelasticity methods with emphasis on turbomachinery applications. *Journal of fluids and structures*, 10(3):237–267, 1996.

[92] H. G. Matthies and J. Steindorf. Partitioned strong coupling algorithms for fluid–structure interaction. *Computers & Structures*, 81(8):805–812, 2003.

[93] K. Maute, M. Nikbay, and C. Farhat. Sensitivity analysis and design optimization of three-dimensional non-linear aeroelastic systems by the adjoint method. *International Journal for Numerical Methods in Engineering*, 56(6):911–933, 2003.

[94] F. R. Menter. Two-equation eddy-viscosity turbulence models for engineering applications. *AIAA journal*, 32(8):1598–1605, 1994.

[95] A. Michalski, P. Kermel, E. Haug, R. Löhner, R. Wüchner, and K.-U. Bletzinger. Validation of the computational fluid–structure interaction simulation at real-scale tests of a flexible 29m umbrella in natural wind flow. *Journal of Wind Engineering and Industrial Aerodynamics*, 99(4):400–413, 2011.

[96] C. Michler, S. Hulshoff, E. Van Brummelen, and R. De Borst. A monolithic approach to fluid–structure interaction. *Computers & fluids*, 33(5):839–848, 2004.

[97] C. Michler, E. Van Brummelen, and R. De Borst. An interface Newton–Krylov solver for fluid–structure interaction. *International Journal for Numerical Methods in Fluids*, 47(10-11):1189–1195, 2005.

[98] J.-O. Mo, A. Choudhry, M. Arjomandi, and Y.-H. Lee. Large eddy simulation of the wind turbine wake characteristics in the numerical wind tunnel model. *Journal of Wind Engineering and Industrial Aerodynamics*, 112:11–24, 2013.

[99] D. Mok and W. Wall. Partitioned analysis schemes for the transient interaction of incompressible flows and nonlinear flexible structures. *Trends in computational structural mechanics, Barcelona*, 2001.

[100] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36, 2014.

[101] N. M. Newmark. A method of computation for structural dynamics. *Journal of the Engineering Mechanics Division*, 85(3):67–94, 1959.

[102] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[103] E. Onate, S. R. Idelsohn, M. A. Celigueta, and R. Rossi. Advances in the particle finite element method for the analysis of fluid–multibody interaction and bed erosion in free surface flows. *Computer Methods in Applied Mechanics and Engineering*, 197(19):1777–1800, 2008.

[104] J. L. Ortiz, A. A. Barhorst, and R. D. Robinett. Flexible multibody systems–fluid interaction. *International journal for numerical methods in engineering*, 41(3):409–433, 1998.

[105] R. Palacios and C. S. Cesnik. Geometrically nonlinear theory of composite beams with deformable cross sections. *AIAA journal*, 46(2):439–450, 2008.

[106] R. Parent. *Computer animation: algorithms and techniques*. Newnes, 2012.

[107] V. C. Patel, W. Rodi, and G. Scheuerer. Turbulence models for near-wall and low reynolds number flows-a review. *AIAA journal*, 23(9):1308–1319, 1985.

[108] M. J. Patil, D. H. Hodges, and C. E. S. Cesnik. Nonlinear aeroelasticity and flight dynamics of high-altitude long-endurance aircraft. *Journal of Aircraft*, 38(1):88–94, 2001.

[109] S. Piperno and C. Farhat. Partitioned procedures for the transient solution of coupled aeroelastic problems–Part II: energy transfer analysis and three-dimensional applications. *Computer Methods in Applied Mechanics and Engineering*, 190(24):3147–3170, 2001.

[110] S. Piperno, C. Farhat, and B. Larrouturou. Partitioned procedures for the transient solution of coupled aroelastic problems Part I: Model problem, theory and two-dimensional application. *Computer methods in applied mechanics and engineering*, 124(1):79–112, 1995.

[111] A. Popp, A. Seitz, M. W. Gee, and W. A. Wall. Improved robustness and consistency of 3D contact algorithms based on a dual mortar approach. *Computer Methods in Applied Mechanics and Engineering*, 264:67–80, 2013.

[112] W. Press, S. Teukolsky, W. Vetterline, and B. P. Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[113] M. A. Puso. A 3D mortar method for solid mechanics. *International Journal for Numerical Methods in Engineering*, 59(3):315–336, 2004.

[114] M. A. Puso and T. A. Laursen. A mortar segment-to-segment frictional contact method for large deformations. *Computer Methods in Applied Mechanics and Engineering*, 193(45):4891–4913, 2004.

[115] P. Ryzhakov, R. Rossi, S. R. Idelsohn, and E. Oñate. A monolithic lagrangian approach for fluid–structure interaction problems. *Computational mechanics*, 46(6):883–899, 2010.

[116] Y. Saad and M. H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[117] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Future Generation Computer Systems*, 20(3):475–487, 2004.

[118] A. A. Shabana. *Dynamics of multibody systems*. Cambridge university press, 2013.

[119] Y. S. Shin. Ship shock modeling and simulation for far-field underwater explosion. *Computers & Structures*, 82(23):2211–2219, 2004.

[120] S. Sicklinger. *Stabilized co-simulation of coupled problems including fields and signals*. PhD thesis, Technische Universität München, 2014.

[121] E. Simiu and R. H. Scanlan. *Wind effects on structures*. Wiley, 1996.

[122] M. J. Smith, C. E. Cesnik, and D. H. Hodges. Evaluation of some data transfer algorithms for noncontiguous meshes. *Journal of Aerospace Engineering*, 13(2):52–58, 2000.

[123] C. Speziale. Turbulence modeling for time-dependent rans and vles: a review. *AIAA journal*, 36(2):173–184, 1998.

[124] S. Streiner. *Beitrag zur numerischen Simulation der Aerodynamik und Aeroelastik großer Windkraftanlagen mit horizontaler Achse*. Verlag Dr. Hut, 2011.

[125] T. E. Tezduyar, S. Sathe, and K. Stein. Solution techniques for the fully discretized equations in computation of fluid–structure interactions with the space–time formulations. *Computer Methods in Applied Mechanics and Engineering*, 195(41):5743–5753, 2006.

[126] R. Torii, M. Oshima, T. Kobayashi, K. Takagi, and T. E. Tezduyar. Fluid–structure interaction modeling of aneurysmal conditions with high and normal blood pressures. *Computational Mechanics*, 38(4-5):482–490, 2006.

[127] S. Turek and J. Hron. *Proposal for numerical benchmarking of fluid-structure interaction between an elastic object and laminar incompressible flow*. Springer, 2006.

[128] B. Uekermann, B. Gatzhammer, and M. Mehl. Coupling algorithms for partitioned multi-physics simulations. In *Informatik 2014, GI-Edition - Lecture Notes in Informatics (LNI)*, page 232, 2014.

[129] R. Unger, M. C. Haupt, and P. Horst. Application of lagrange multipliers for coupled problems in fluid and structural interactions. *Computers & structures*, 85(11):796–809, 2007.

[130] E. Van Brummelen, S. Hulshoff, and R. De Borst. Energy conservation under incompatibility for fluid–structure interaction problems. *Computer Methods in Applied Mechanics and Engineering*, 192(25):2727–2748, 2003.

[131] A. Varello, L. Demasi, E. Carrera, and G. Giunta. An improved beam formulation for aeroelastic applications. In *Proceedings of the 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, pages 12–15, 2010.

[132] H. K. Versteeg and W. Malalasekera. *An introduction to computational fluid dynamics: the finite volume method*. Pearson Education, 2007.

[133] J. Vierendeels, L. Lanoye, J. Degroote, and P. Verdonck. Implicit coupling of partitioned fluid–structure interaction problems with reduced order models. *Computers & structures*, 85(11):970–976, 2007.

[134] W. A. Wall. *Fluid-struktur-interaktion mit stabilisierten finiten elementen*. PhD thesis, Universität Stuttgart, 1999.

[135] W. A. Wall and T. Rabczuk. Fluid–structure interaction in lower airways of ct-based lung geometries. *International Journal for Numerical Methods in Fluids*, 57(5):653–675, 2008.

[136] T. Wang, S. Sicklinger, R. Wüchner, and K.-U. Bletzinger. Concept and realization of coupling software empire in multi-physics co-simulation. In *Computational Methods in Marine Engineering V MARINE 2013*, pages 289–298. CIMNE, 2013.

[137] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics*, 12(6):620–631, 1998.

[138] D. C. Wilcox et al. *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA, 1998.

[139] B. I. Wohlmuth. A mortar finite element method using dual spaces for the lagrange multiplier. *SIAM Journal on Numerical Analysis*, 38(3):989–1012, 2000.

[140] P. Wriggers. *Nonlinear finite element methods*. Springer Science & Business Media, 2008.

[141] W. Wunderlich and G. Kiener. *Statik der Stabtragwerke*. Vieweg+Teubner Verlag, 2004.

[142] Y. Young. Fluid–structure interaction analysis of flexible composite marine propellers. *Journal of Fluids and Structures*, 24(6):799–818, 2008.