National Technical University of Athens

School of Mechanical Engineering
Fluids Department
Laboratory of Thermal Turbomachines
Parallel CFD & Optimization Unit

Development of a Topology to Shape Transition Software through Marching Algorithms

Diploma Thesis
by

Efstratios Trachanias

Supervisor:
Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

To my love,
Panagiota

Acknowledgements

At this point, I feel the need to express my deepest gratitude to my supervisor, Professor Kyriakos C. Giannakoglou, for his trust, guidance, and steadfast support throughout the journey of completing this thesis. His expertise, insightful feedback, and encouragement played a decisive role in shaping the direction and quality of my work. Working under his supervision has been an invaluable privilege for me.

I would also like to express my special thanks to Dr. Varvara Asouti, Dr. Marina Kontou and PhD candidate Nikolaos Galanos. They were an invaluable resource, always ready to lend a helping hand whenever I encountered technical difficulties or had questions regarding the intricate details of my study. Their patience, knowledge, and willingness to assist me, greatly contributed to the successful completion of this thesis.

I shall further extend my sincere gratitude to my family, my devoted parents, Panos and Myrsini, and my sister, Maria, whose unwavering support, encouragement, and understanding sustained me throughout this academic endeavor. I also thank my beloved grandparents, who supported me in every way during my life.

I couldn't forget my muse, Panagiota. Her belief in my abilities, motivation during challenging times, determination and constant support provided the foundation for me to conquer any challenge.

National Technical University of Athens
School of Mechanical Engineering
Fluids Department
Parallel CFD & Optimization Unit

Development of a Topology to Shape Transition Software through Marching Algorithms

Diploma Thesis

Efstratios Trachanias

Supervisor: Kyriakos C. Giannakoglou, Professor NTUA

Athens, 2024

Abstract

This diploma thesis programs a software tool for post-processing Topology Optimization (TopO) results in Fluid Mechanics, facilitating the transition from the real-valued fluid or solid indicators, known as porosity fields in the design domain of TopO, to explicit surface mesh representations of the corresponding shape.

TopO is a powerful tool used to determine the optimal material distribution for a performance objective, within a given design domain and under specific constraints. To do that, an artificial porosity field is introduced in the flow equations, restricting fluid flow in solidified areas of the domain. Despite its many advantages, this technique is prone to inaccuracies due to the lack of an exact interface between the solid and fluid regions, where boundary conditions must be imposed. This thesis addresses the critical step of converting the above mentioned porosity fields into surface meshes in the form of STL files, enabling the manufacturing of designs or their evaluation through Computational Fluid Dynamics (CFD) analysis on body–fitted grids. Link with an ensuing Shape Optimization (ShpO) is also possible to further refine a design.

By adopting the Marching Cubes algorithm, an established framework originally developed for medical data visualization, a methodological innovation is introduced by extending its concept for the case of unstructured/hybrid grids with a variety of elements. This gives rise to generalized Marching Algorithms that are able to handle both structured and unstructured grids. The implementation is performed in C++ and is compatible with the CFD–based Optimization software (OpenFOAM©and in–house) of the Parallel CFD & Optimization Unit of NTUA (PCopt/NTUA). The programmed tools are demonstrated through a series of use–case scenarios and real–world industrial applications.

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης

Ανάπτυξη Λογισμικού Μετάβασης από Τοπολογία σε Σχήμα μέσω Αλγορίθμων Προέλασης

Διπλωματική εργασία

Ευστράτιος Τραχανιάς

Επιβλέπων: Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Περίληψη

Η διπλωματική εργασία παρουσιάζει την ανάπτυξη λογισμικού για την επεξεργασία των αποτελεσμάτων Βελτιστοποίησης Τοπολογίας (ΒεΤο) σε θέματα ρευστομηχανικής/αεροδυναμικής, με σκοπό την αναπαράσταση του τοιχώματος μεταξύ στερεού και ρευστού σε μορφή επιφανειακού πλέγματος.

Η ΒεΤο είναι μέθοδος που χρησιμοποιείται για τον προσδιορισμό της βέλτιστης κατανομής υλικού για συγκεκριμένο στόχο, εντός ενός σχεδιαστικού χωρίου και υπό συγκεκριμένους περιορισμούς. Για τον σκοπό αυτόν, εισάγεται ενα τεχνητό πεδίο πορώδους στις εξισώσεις ροής, εμποδίζοντας τη ροή σε περιοχές που στερεοποιούνται. Παρά τα πλεονεκτήματα της, η μέθοδος αυτή είναι επιρρεπής σε ανακρίβειες λόγω έλλειψης ενός καθορισμένου τοιχώματος μεταξύ του στερεού και του ρευστού, όπου θα επιβληθούν οι οριακές συνθήκες. Η εργασία αντιμετωπίζει το κρίσιμο βήμα της μετάβασης από τα πεδία πορώδους σε επιφανειακά πλέγματα των υπό εξέταση γεωμετριών σε μορφή αρχείων STL, επιτρέποντας την κατασκευή τους ή την αξιολόγησή τους μέσω υψηλής πιστότητας ανάλυσης Υπολογιστικής Ρευστοδυναμικής (ΥΡ) σε σωματόδετα πλέγματα. Είναι επίσης δυνατή η περαιτέρω βελτίωση της γεωμετρίας μέσω Βελτιστοποίησης Σχήματος.

Υιοθετώντας τον αλγόριθμο Marching Cubes, μια διαδεδομένη μέθοδο που αναπτύχθηκε αρχικά για την τριδιάστατη απεικόνιση τομογραφιών, εισάγεται η επέκτασή της στην περίπτωση μη–δομημένων πλεγμάτων με στοιχεία διαφόρω τύπων. Γεννώντας κατ΄ αυτό τον τρόπο γενικευμένους Αλγορίθμους Προέλασης ικανούς να διαχειρίζονται δομημένα αλλά και μη–δομημένα/υβριδικά πλέγματα. Η υλοποίηση γίνεται σε γλώσσα C++ και είναι πλέον συμβατή με τα λογισμικά Βελτιστοποίησης μέσω ΥΡ (OpenFOAM© και οικείο) της Μονάδας Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης ΕΜΠ (ΜΠΥΡ&Β/ΕΜΠ). Το λογισμικό που προγραμματίστηκε πιστοποιείται σε μια σειρά σεναρίων δοκιμής και πραγματικών βιομηχανικών εφαρμογών.

Acronyms

CFD	Computational Fluid Dynamics
NTUA	National Technical University of Athens
PCOpt	Parallel CFD & Optimization unit
GMB	Gradient Based Method
SD	Sensitivity Derivatives
TopO	Topology Optimization
ShpO	Shape Optimization
SPTopO	Standard Porosity–based Topology Optimization
IBM	Immersed Boundary Method
FSI	Fluid-Solid Interface
TtoST	Topology–to–Shape Transition
MC	Marching Cubes
STL	STereoLithography
CPU	Central Processing Unit
a.k.a.	also known as
w.r.t.	with respect to

ΕΜΠ	Εθνικό Μετσόβιο Πολυτεχνείο
ΕΘΣ	Εργαστήριο Θερμικών Στροβιλομηχανών
ΜΠΥΡ&Β	Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης
ΥΡ	Υπολογιστική Ρευστοδυναμική
ΒεΤο	Βελτιστοποίηση Τοπολογίας
ΒεΜο	Βελτιστοποίηση Μορφής

Contents

Contents **xi**

1 Introduction **1**

1.1 Background in CFD–based Optimization 2

1.1.1 Optimization and the Adjoint Method 2

1.1.2 Shape and Topology Optimization in Fluid Mechanics 3

1.1.3 Motivation - Thesis Scope 3

1.2 Methodology Overview 4

1.2.1 Marching Cubes Algorithm 4

1.2.2 Generalized Marching Algorithms 4

1.2.3 Thesis Contributions 4

1.3 Thesis Outline 6

2 Shape and Topology Optimization in Fluid Mechanics **7**

2.1 Introduction to Shape Optimization 7

2.2 Introduction to Topology Optimization 8

2.2.1 Topology Optimization in Structural Mechanics 8

2.2.2 Topology Optimization in Fluid Mechanics 9

2.3 Thesis Methodological Approach 12

3 Marching Algorithms **13**

3.1 Review of the Marching Cubes algorithm 14

3.1.1 MC Algorithm overview 14

3.1.2 Handling Ambiguities 18

3.2 Adaptation for Hybrid Grids 21
3.2.1 Marching Elements - Formulation of cases 21
3.2.2 Thesis Impact 26

4 Algorithmic Implementation 27
4.1 Algorithm overview 27
4.2 Mesh Geometry and Connectivity Input 29
4.2.1 The in–house input file format of PCOpt/NTUA 29
4.2.2 The OpenFOAM© mesh description format 30
4.3 Volume weighted cell center to vertex interpolation 31
4.4 Data Structure 32
4.4.1 Edge numbering 32
4.4.2 Running Example 33
4.5 Element processing – Generation of Triangles 35
4.6 File output 38

5 Applications 39
5.1 Spheres 39
5.1.1 Sphere on cubic grid 40
5.1.2 Sphere on tetrahedral grid 42
5.1.3 Sphere on hybrid grid 44
5.1.4 Two spheres on cubic grid 46
5.1.5 Four unequal spheres on hybrid grid 47
5.2 Stanford 3D Scanning Repository's Models 48
5.2.1 The 'Stanford Bunny' 48
5.2.2 Dragon 50
5.3 Industrial Applications – Heat Exchangers 52
5.3.1 2D Heat Exchanger 1 Inlet 1 Outlet 53
5.3.2 Bi-fluid 3D Heat Exchanger 1 Inlet 1 Outlet 54
5.3.3 Bi-fluid 3D Heat Exchanger 1 Inlet 8 Outlets 56

6 Summary - Conclusions **59**

6.1 Overview 59

6.2 Future Work 60

A Marching Algorithms Triangulation Tables **61**

A.1 Hexahedra 62

A.2 Prisms 69

A.3 Pyramids 71

A.4 Tetrahedra 73

A.5 Quadrilaterals 74

Extensive summary in Greek – Εκτενής περίληψη στα Ελληνικά **75**

Bibliography – Βιβλιογραφία **89**

Chapter 1

Introduction

The increasing performance of modern computer systems alongside the advancements in computational methods has given rise to innovative ways of designing and simulating in a significant range of industrial applications. The widespread availability of simulation tools has taken over the preliminary design stages of many mechanical systems that incorporate complex structures, revolutionizing the way engineers approach the development of products and systems. In particular, Computational Fluid Dynamics (CFD), a branch of Fluid Mechanics to numerically analyze fluid flows, allows for the detailed simulation of fluid behavior in various conditions and environments, enabling designers to predict performance and identify potential issues before physical prototypes are ever built. The integration of CFD with optimization methods, such as Shape Optimization (ShpO) and Topology Optimization (TopO), seeks the optimal design targeting specific performance criteria. Contemporary approaches employ the synergistic use of TopO and ShpO, whereby a solution initially found by TopO is further refined by ShpO. The Parallel CFD & Optimization Unit (PCOpt) at the National Technical University of Athens (NTUA) is at the forefront of research in these domains, developing and applying adjoint-based TopO and ShpO methods in aerodynamic and hydrodynamic studies published in a number of papers [1, 2, 3, 4, 5] and PhD theses [6, 7, 8, 9, 10].

1.1 Background in CFD–based Optimization

1.1.1 Optimization and the Adjoint Method

Optimization refers to the process of finding the best solution to a problem, given specific constraints, from all feasible solutions. The integration of CFD analysis with optimization, gives rise to CFD-based optimization methods. In an optimization problem, a selection of the parameters that can be modified is performed, thus creating a set of variables referred to as *design* or *optimization variables*, that alter the flow solution and yield a quantifiable measure of performance or efficiency known as the *objective function*. The goal of the optimization method is to compute the values of the design variables that minimize or maximize the objective function in hand. In every optimization cycle the passage from one set of variables to another is subject to computations. CFD-based optimization methods can be classified into two main categories, according to the way the optimal set of design variables is computed: *stochastic* [11] and *deterministic* [12].

Stochastic optimization methods involve random sampling of the design space, enhanced with heuristics to select and assess a set of design variables [13]. Evolutionary Algorithms (EA) represent a significant subset of stochastic population-based optimization methods. By drawing inspiration from biological evolution, these algorithms employ mechanisms such as selection, crossover (recombination), and mutation to evolve a set of candidate solutions towards an optimal or near-optimal solution over successive generations.

The deterministic optimization algorithms improve a solution by computing the gradient of the objective function in question with respect to (w.r.t) the design variables, also known as the *sensitivity derivatives* (SD). An update to the design variables is thereafter computed based on the direction dictated by the sensitivity derivatives. Subsequently, the flow field, the objective function value and the new SD field are computed based on the updated design. This process is repeated until either the objective function has converged to its minimum value or the user–defined maximum number of optimization cycles is reached [14].

The *adjoint* method [15, 16] of computing the sensitivity derivatives is a method that has a cost practically independent from the number of the design variables N. As a result, this method is a perfect choice for large industrial optimization problems. In order to achieve this independence, an augmented objective function is defined, by adding the volume integrals of the residuals of the flow equation (also referred to as the *primal* or *state* equations), multiplied by the adjoint (or *co-state* or *dual*) variable fields, to F. Considering that the residuals of the primal equations must be zero, $F \equiv F_{aug}$. After differentiating the augmented objective function and rearranging the resulting terms, the system of adjoint equations and adjoint boundary conditions is formulated, the numerical solution of which leads to a N-independent computation of the SDs.

1.1.2 Shape and Topology Optimization in Fluid Mechanics

Shape Optimization (ShpO) and Topology Optimization (TopO) in fluid mechanics are two established frameworks [17, 18, 19] that belong to CFD–based optimization methods. ShpO involves the use of a number of control variables, that parameterize the shape under consideration, to refine the performance of the design. These could be, for instance, the coefficients of the Bézier–Bernstein polynomials or the control point coordinates of volumetric B-splines polynomials parameterizing the shape under consideration. In large scale ShpO problems, the control variables can be significant in number, and subsequently the SDs w.r.t. the control variables. This makes the adjoint method a particularly good choice for computing the necessary derivatives.

TopO, initially developed for structural mechanics [20], focuses on optimizing material distribution within a given space to achieve specific performance goals. It has been adapted to fluid mechanics, where a real–valued porosity dependent term is introduced in the flow equations (Brinkman penalization method). This method is the predominant method in fluids referred to as *Standard Porosity–based TopO (SPTopO)*. The porosity terms in each node (for vertex–centered) or cell (for cell-centered solvers) act as the design variables of the optimization problem. Since the number of the design variables coincides with the number of the grid nodes or cells, the adjoint method is the perfect choice for computing sensitivities of the objective function w.r.t. the porosity values.

Although TopO and ShpO are two disciplines that have traditionally been viewed as distinct and separate approaches, it is conceivable that they can find better solutions in tandem with ShpO enchancing and refining a solution initially produced by TopO.

1.1.3 Motivation - Thesis Scope

Despite its many advantages, SPTopO is prone to inaccuracies due to the imprecise depiction of the Fluid–Solid Interface (FSI). The boundary conditions regarding the FSI are not directly enforced, but are instead weakly treated through the porosity term. Quite often when TopO's solutions are re–evaluated with a flow solver running on a body–fitted grid to the computed FSI, they do not perform as expected [21]. This is why, the extraction of the body shape from the optimized porosity field becomes necessary. The latter constitutes the topic that is addressed in the current thesis, proposing a tool to extract the FSI geometry from the porosity field solution of TopO, using Marching Algorithms. The extracted geometry can be post-processed to generate body–fitted grids for the successive implementation of ShpO or used as a manufacturable solution without considering a continuation to ShpO.

1.2 Methodology Overview

1.2.1 Marching Cubes Algorithm

To accomplish the above we adopt the Marching Cubes (MC) algorithm [22], an established formulation in the Computer Graphics area, initially proposed for the visualization of medical data. The algorithm is used to generate surface meshes of triangular elements from scalar fields of data on the nodes of a structured (containing cubes) grid. The workflow includes the processing of each cube in the grid, based on the values of its nodes. A threshold value, known as the *isovalue* is introduced by the user, and each node of the cube is flagged (above the threshold value) or unflagged (below the threshold) value. The state of each node is encoded into an 8–bit index, leading to 256 possible configurations. However, by employing symmetries the number of unique configurations is significantly reduced. These configurations encode the arrangement of triangles for the created surface and the position of the new vertices of the triangles on the edges of the grid's cube. These configurations are precompiled and saved in look–up tables prior to the algorithm's execution, avoiding calculations during the processing of each individual element of the grid. The output file takes the form of a CAD–compatible file of the surface mesh, such as the STereo Lithography (STL) file.

1.2.2 Generalized Marching Algorithms

In the case of grids used in CFD analysis, they can be either structured or unstructured/hybrid grids. The former refers to the case of grids containing only hexahedral elements whereas the latter regards grids that contain hexahedra as well as other elements such as prisms, pyramids and tetrahedra. In order to be able to extract the FSI from TopO's solutions on hybrid grid, an expansion of the MC algorithm's concept for other element types is necessary. In the current thesis, the formulation of appropriate algorithms for other element types is introduced, giving rise to generalized Marching Algorithms for Hexahedra, Prisms, Pyramids and Tetrahedra. Also the boundary reconstruction is treated with a Marching Quadrilaterals scheme.

1.2.3 Thesis Contributions

This thesis presents a study and implementation of the proposed methodology, leading to several key contributions.

MC Algorithm Implementation and Formulation of 'generalized Marching Algorithms'. The adaptation of the MC triangulation look–up tables from the literature [23] is performed and the algorithm is implemented in C++. The appropriate Data Structure for the case of structured cubic grids is formulated for registering of all the edges of the grid and the storage of the generated vertices of

the triangles. Then the triangulation look–up tables (see Appendix A) for the other element types and the boundary adaptation of the Data Structure and revised algorithm implementation for the case of unstructured & hybrid grids is performed. The algorithm is then rendered compatible with the CFD–based Optimization software (OpenFOAM©and in–house) of the Parallel CFD & Optimization Unit of NTUA (PCopt/NTUA).

The in–house GPU-accelerated CFD solver, PUMA [24, 25], numerically solves the Navier-Stokes equations for compressible and incompressible fluids. The flow and their adjoint equations are discretized on unstructured/hybrid meshes, using the vertex–centered finite volume method. The solution of a TopO run is readily available for the TtoST process using the method proposed in this thesis, that requires information on the nodes of the grid.

Volume weighted cell–center to vertex interpolation for compatibility with OpenFOAM©. The OpenFOAM© (Open Field Operation and Manipulation) is an open–source CFD software initially released in 2004 by the OpenCFD company. It offers a comprehensive set of tools and solvers for simulating and analyzing fluid flows. It employs a cell–centered finite volume method for discretizing the governing equations of fluid flow, constituting a cell-centered code. The PCOpt/NTUA, with expertise in the adjoint method, has developed adjoint optimization tools inside the OpenFOAM© software. The solution of a TopO run has to be interpolated to the grid's vertices in order to employ the proposed method for the TtoST. A volume weighted cell center to vertex interpolation scheme is incorporated in the algorithm to interpolate the field solutions, from OpenFOAM© stored on the cell centers, to the nodes of the grid.

Validation and Assessment in a variety of scenarios. The programmed tools are then assessed and demonstrated through a series of use–case scenarios and real–world industrial applications. First the simple case of spheres is explored in three different types of sample grids, a structured grid of hexahedral elements, an unstructured grid of tetrahedral elements and a hybrid grid containing hexahedra, prisms, pyramids, and tetrahedra. It is demonstrated that the algorithms effectiveness in capturing the geometry of the sphere relies on the resolution of the background grid, and poor resolution of the grid results in poor reconstruction of the geometry. However the radius of the generated surface never exceeds the nominal radius of the original sphere. The addition of more separated spheres in the grid is explored to investigate how the algorithm treats the interface between them. It is shown that if the background grid has adequate resolution and the spheres are separated within different cells of the grid, the algorithm is able to distinguish between them.

The application of the algorithm is then explored for models taken from the Stanford 3D Scanning Repository [26]. The geometries are introduced within sample grids, with the use of the *searchableSurface* tool of OpenFOAM©. The topology field is then processed with the TtoST tool proposed and the resulting geometries are compared with the original ones.

Tool application for the design of Heat Exchangers. Finally, the algorithm is applied in the extraction of the FSI from TopO solutions regarding the design of heat exchangers. Three cases are explored, a 2D heat exchanger, a bi–fluid 3D heat exchanger of one inlet and one oulet and a bi–fluid 3D heat exchanger of 1 inlet and 8 outlets. In all cases the extraction of the FSI from the porosity field solution of TopO is demonstrated along with the grid's boundary integration.

1.3 Thesis Outline

Following the Introduction, the structure of this thesis is organized as follows:

- **Chapter 2** provides an introduction to ShpO and TopO in fluid mechanics, along with the challenges in current methods and the need for a TtoST Tool
- **Chapter 3** explores the fundamental theory of the MC algorithm and expands its concept giving rise to the 'generalized Marching Algorithms'
- **Chapter 4** details the methodology followed and its implementation in C++.
- **Chapter 5** presents the case results from 2D and 3D application of the implemented tool.
- **Chapter 6** concludes the thesis with key findings and future research directions.
- **Appendix A** the full pre–compiled look up tables for the triangulation of elements (as incorporated into the programmed software) are listed.

Chapter 2

Shape and Topology Optimization in Fluid Mechanics

As mentioned in the introduction, optimization refers to the process of finding the best solution to a problem from all feasible solutions. The integration of CFD analysis with optimization, gives rise to CFD-based optimization methods. Most optimization problems in the field of fluid mechanics can be classified as either shape or topology optimization, with their respective disciplines Shape Optimization (ShpO) and Topology Optimization (TopO). ShpO involves the use of a number of control variables, that parameterize the shape under consideration, to refine the performance of the design. On the other hand, TopO, is a method that can produce designs independent of an initial shape, exploring a wider range of design possibilities, often leading to more efficient and innovative solutions that might not be intuitively apparent. Although TopO and ShpO have traditionally been viewed as distinct and separate approaches since their origins, it is possible that their most effective solutions could emerge through a collaborative process, where ShpO enhances and refines a solution initially identified by TopO.

2.1 Introduction to Shape Optimization

According to the control theory adapted for the needs of CFD-based optimization [27], the shape to be optimized is controlled by a number of variables, referred to as *design* or *optimization variables*. These could be, for instance, the coefficients of the Bézier–Bernstein polynomials or the control point coordinates of volumetric B-splines polynomials which parameterize the shape under consideration. The quality of the shape to be optimized is evaluated by computing a usually integral quantity,

known as the *objective function*. The objective function can be defined either on the boundaries, such as the total pressure losses which at the inlet and outlet of the geometry, or in a volume inside the geometry, such as the noise induced in that region. Goal of the optimization is to compute the values of the design variables that minimize or maximize the objective function. The iterative process is usually deterministic, computing the gradient of the objective function in question w.r.t the design variables, also known as the *sensitivity derivatives*. A shape update is performed based on the direction dictated by the sensitivity derivatives. CFD–based shape optimization is used on a regular basis to design aerodynamic structures [17, 18, 28, 29].

2.2 Introduction to Topology Optimization

TopO is an established computational method used to determine the optimal material distribution within a given design space, under specific conditions and constraints, to achieve a predetermined performance objective. First introduced in structural mechanics by Bendsøe & Kikuchi [20] over thirty–five years ago, TopO has since been extended to various scientific fields, including Fluid Mechanics.

2.2.1 Topology Optimization in Structural Mechanics

As a structural optimisation method, it distinguishes itself from the more classical discipline shape optimisation, by the fact that there does not need to be an initial structure defined a priori [19]. In structural mechanics, instead of adjusting the shape of the structure's boundary to achieve better performance, TopO identifies the material density field at the design domain, with values ranging from zero (presence of void/absence of material) to one (solid). The procedure, thus, determines where material should be added to increase structural stiffness under certain load. In Fig. 2.1 the concept of TopO in structural mechanics is illustrated for the design of a cantilevered beam.

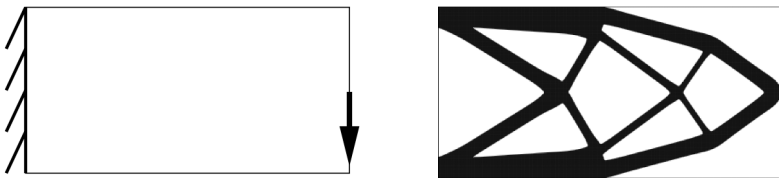


Figure 2.1: *The concept of TopO in structural optimization: Left: Computational domain and the applied loads and boundaries for the design of a cantilever beam. Right: Solution from TopO. White areas correspond to void (zero material density) whereas dark areas correspond to the designed structure (high material density). From [30].*

2.2.2 Topology Optimization in Fluid Mechanics

In fluid mechanics the same idea was successfully expanded for Stokes [31, 32] and laminar [33], turbulent [34], steady and unsteady flows [35, 21]. TopO's predominant method in fluids is achieved by introducing a porosity dependent term, also referred as blockage term (Brinkman penalization method), to the flow equations. Hereafter this method will be referred to as the *Standard Porosity-based TopO* (SPTopO). Solid areas are identified by low porosity or high impermeability, effectively restricting fluid flow through enforced zero velocity conditions. On the other hand, in areas were porosity values are high, the flow encounters no resistance. These areas correspond to the flow passage. TopO seeks the optimal porosity values at each node (for vertex centered codes) or cell (cell-centered codes) in order to minimize the objective function in hand. Hence, the number of design variables coincides with the number of grid nodes or cells [7].

This technique has been extensively used in fluid mechanics due to its flexibility and ability to produce innovative and efficient designs that may not be intuitive or achievable through traditional design methods. The ability to design unconventional shapes and to dramatically alter solution topology during optimization, makes it ideal for designing fluid paths, when there is no knowledge of the connectivity of the inlets and outlets. In Fig. 2.2 the concept of TopO in fluid mechanics is illustrated for the design of an HVAC duct. Also, since the computational domain is static, the computational cost is reduced and the cells' quality is maintained, while still allowing big changes in the design via the porosity field [8].

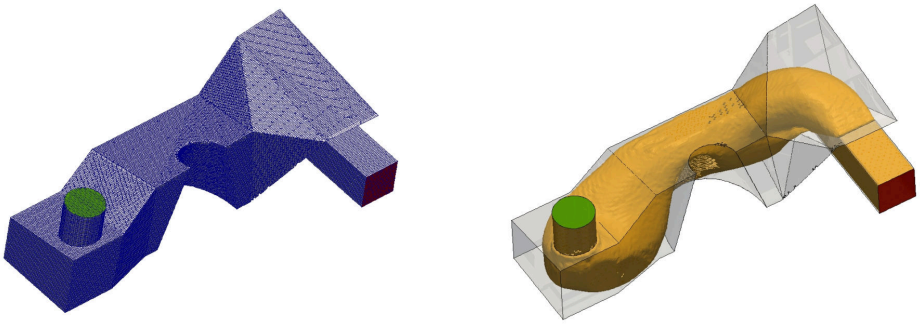
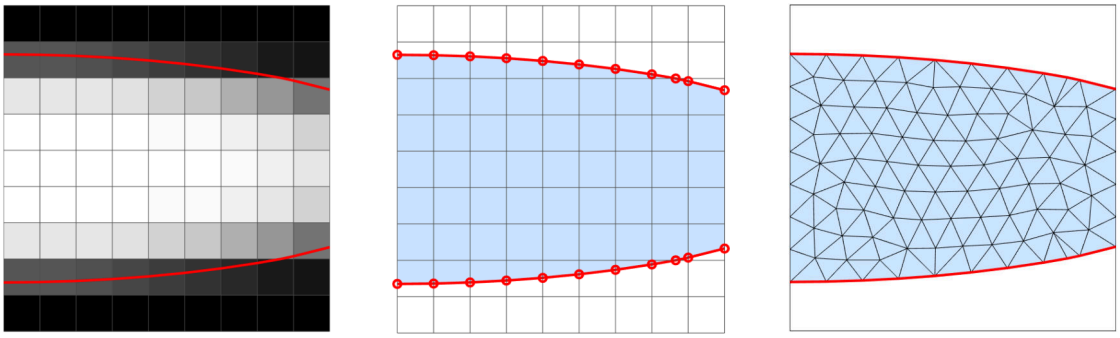


Figure 2.2: *The concept of topology optimization in fluid mechanics: Left: Computational domain for the design of an HVAC duct. Fluid flow from the inlet (green) to the outlet (red) subjected to performance objectives (e.g. minimun pressure losses). Right: Solution from topology optimization. Yellow area corresponds to the Fluid-Solid Interface. From [8].*

Fluid-Solid Interface Depiction

In Fig. 2.3 the three general options for representing the design in TopO are presented for the case of a simple nozzle.



(a) *(b)* *(c)*

Figure 2.3: *Fluid nozzle illustrating the basic differences among design representations in topology optimisation: (b) Standard Porosity–based representation (b) level set cut-cells representation (c) explicit boundary representation (body fitted mesh). Reproduced from [19].*

The first representation [(a) in Fig. 2.3] is that of the standard porosity–based TopO where the flow is penalised in the solid (black) domain. This method has the ability to change topology and change the design dramatically during optimization. The cost of introducing the design is relatively low, as only an the extra porosity term needs to be integrated, with no special interface treatment being necessary. Despite its many advantages, one of the primary limitations of the Standard Porosity– based TopO in Fluid Mechanics is its imprecise depiction of the Fluid-Solid Interface (FSI). The solid wall boundary conditions in the porosity field are not directly enforced but are instead weakly represented through blockage terms in the flow equations. This approach fails to fully capture the influence of solid walls on the flow field and can permit some degree of flow leakage through the solidified parts of the domain. Also, the velocity and pressure fields are present in the entire domain, both fluid and solid regions, which may cause spurious flows and leaking pressure fields, if not penalized sufficiently. In the context of turbulent flows, in which wall functions require accurate wall distances, the lack of a clearly defined FSI introduces inaccuracies [8]. As a result, this can introduce inaccuracies to the optimization process, sometimes yielding sub-optimal solutions. Quite often, when the TopO solutions are re-evaluated using flow solvers running on a grid fitted to the computed FSI, they do not perform as expected [21]. This is why a transition from the optimized porosity field to shape representation is sought. In Fig. 2.4 the inaccuracy of TopO is demonstrated by the increase in the objective function when the TtoST is evaluated on a body–fitted grids and ShpO begins.

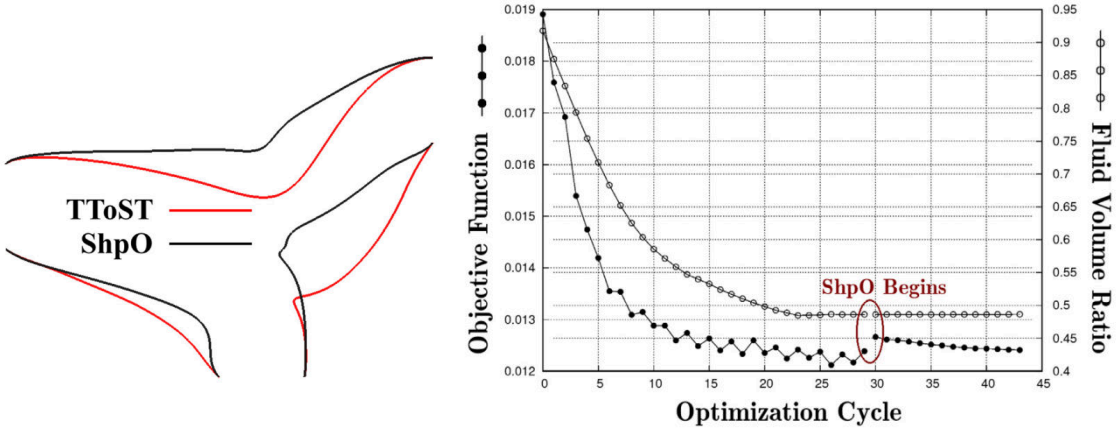


Figure 2.4: *(Left) Comparison of an example's case TtoST boundary as input to ShpO and the optimized ShpO boundary. (Right) The objective and constraint (Fluid Volume Ratio) functions' progression between TopO and ShpO. Overshoot of the objective function when ShpO begins due to inaccuracies in the TopO. From [3]*

The second representation [(b) in Fig. 2.3] is that of an *Immersed Boundary Method* (IBM) [9, 36] where a fixed grid encompasses both the the fluid and the solid regions. A surface-capturing method, the Cut–Cells [10] is integrated using a special scheme and the interface boundary conditions or the coupling between different physics can be introduced. Due to the nature of the method, the design sensitivities are located only at the interface. This means that design changes can only propagate from the interface and no new holes appear automatically. Another disadvantage of IBMs stems from their difficulty to control grid resolution at the vicinity of body surfaces. Since the grid lines are not aligned with the body boundary, creating the high–aspect ratio poses a challenge. As such, IBMs require more dense Cartesian grids to resolve the boundary layer. This problem becomes more pronounced as the Reynolds number increases since the boundary layer thickness reduces in size [9].

The third representation [(c) in Fig. 2.3] is an explicit surface-boundary representation of the geometry based on a body-fitted grid adopting to the nominal geometry shown in red. The explicit boundary method represents the physics well, has the ability to impose boundary conditions directly and permits the control of grid density in areas of interest such as near solid walls. However if the design is changed, the boundary nodes must be moved and the grid must be updated or the domain must be entirely re-meshed. That is why such a representation is suggested as a last step of the TopO run. An algorithms to perform the transition from the topology solution to the surface mesh boundary representation is necessary, in order to extract the FSI. With the extraction of the FSI, it is possible to proceed to manufacturing of the design or to link the problem to ShpO to further refine the solution. The combined use of TopO and ShpO is used in the literature, to mitigate the inaccuracies of the Standard Porosity–based TopO [3, 4, 5].

2.3 Thesis Methodological Approach

The extraction of the optimal body shape from the optimized porosity field becomes necessary. The latter constitutes the topic that is addressed in the current thesis, proposing a tool to extract the FSI geometry from the porosity field solution of TopO, using Marching Algorithms. The extracted geometry can be post-processed to generate body–fitted grids for the successive implementation of ShpO or used as a manufacturable solution without considering a continuation to ShpO.

To accomplish the above we adopt the Marching Cubes algorithm [22], an established formulation in the Computer Graphics area, initially proposed for the visualization of medical data. In the case of CFD grids we are routinely dealing with both structured and unstructured grids. The former refers to the case of grids containing only hexahedral elements whereas the later regards grids that involve hexahedra as well as prisms, pyramids and tetrahedra. In the current thesis, implementations of the Marching Algorithms for both types of grids are presented.

Chapter 3

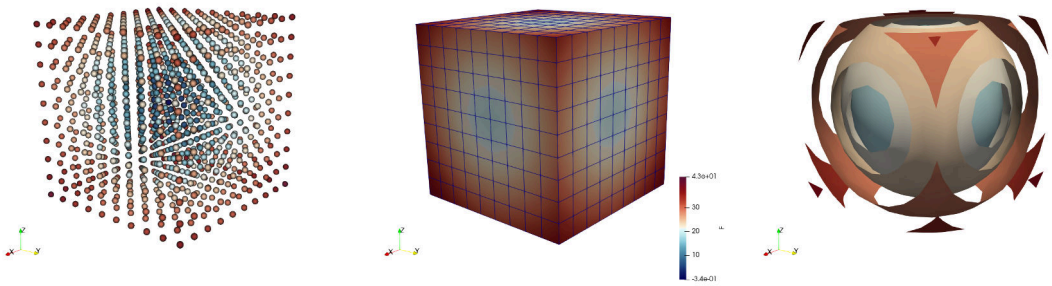
Marching Algorithms

As discussed in Chapter 2, in the field of CFD and TopO, Fluid–Solid Interface (FSI) definition is a challenging topic. The transition from the topology solution, that is the field of real-valued solid/fluid indicators, a.k.a. porosity fields, to shape representation is important, in order to overcome limitations of the standard porosity–based TopO. By extracting the FSI and generating body–fitted grids, we can accurately impose boundary conditions and distances from the solid walls, and hence obtain high–fidelity values of objective functions. With the shape representation, a successive implementation of ShpO is possible, or the design can be sent for manufacturing if ShpO is not desirable.

Marching Algorithms are commonly used in the scientific visualization and contouring of complex geometries in various fields [37]. Starting from the fundamental theory of the so–called *Marching Cubes* algorithm [22] and extending it to other geometric elements (hexahedra, prisms, pyramids, tetrahedra, quadrilaterals) we give rise to *Generalized Marching Algorithms* that can handle 2D and 3D hybrid grids. In this diploma thesis the implementation of these algorithms is programmed in C++, and their successive employment for the Topology-to-Shape Transition (TtoST) in TopO is explored in a number of applications.

3.1 Review of the Marching Cubes algorithm

In 1987, Lorenson and Cline introduced the Marching Cubes (MC) algorithm [22], which has since become a well–established method for surface extraction, particularly in the field of medical imaging. This algorithm is used in 3D structured (cubic) grids with scalar fields applied on the nodes, constituting a vertex–centered code. Such a field, describing a spherical field growing outwards, can be seen in Fig. 3.1.



(a) *(b)* *(c)*

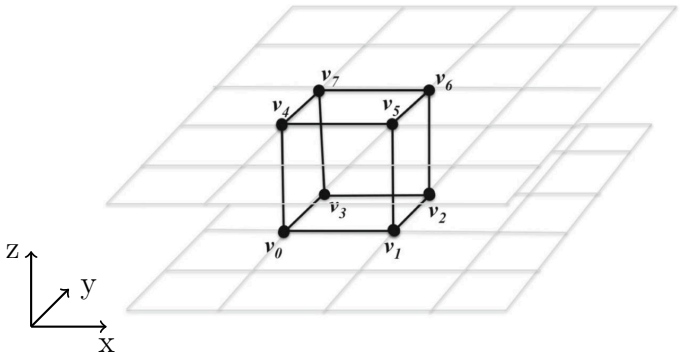
Figure 3.1: *A spherical field growing outwards applied on a structured cubic grid. (a) Scalar values on the nodes (b) the structured cubic grid, (c) sample isosurfaces.*

From this 3D array of data, the MC algorithm creates a polygonal representation of constant–value surfaces, called *isosurfaces*. An *isosurface* can be defined as follows. Given a scalar field $F(\mathbf{P})$, with F a scalar function on $\mathbb{R}^3$, the surface that satisfies $F(\mathbf{P}) = a$, where a is a constant, is called the *isosurface* defined by a. The value a is called the *isovalue*. In practice, isosurface extraction usually involves generation of an approximate, piece–wise linear isosurface (usually composed of a collection of triangles) on a sampled scalar field. The isosurface could consist of a single component or of multiple, disjoint ones, as seen in Fig. 3.1(c). By treating the data in a cube–by–cube manner, the algorithm determines how the desired isosurface intersects a cube and then moves, 'marches', to the next cube. The topology of the isosurface within the cube is determined, by computing which of the vertices of the cube are 'inside' the surface and which are 'outside', yielding a configuration of triangles for each cube.

3.1.1 MC Algorithm overview

The MC algorithm processes the data set sequentially, one cube at a time. During this processing, each node of the grid that constitutes the current cube's vertex v_i that has a value equal to or above the isovalue α is flagged; all other vertices with values below α remain unflagged. Consequently, an 8–bit index for each cube is generated, encoding the status of its vertices as either flagged (boolean 1) or unflagged (boolean 0). Since each of the eight vertices of a cube can be either

flagged or unflagged, there are $2^8 = 256$ possible scenarios for a cube. The cube in the grid, its vertex numbering and its index are shown in Fig. 3.2.



Cube index =	v7	v6	v5	v4	v3	v2	v1	v0

Figure 3.2: *A cube in the grid, the numbering of its vertices, and its index representation.*

The isosurface intersects only the cube edges that have a flagged vertex in one end and an unflagged vertex in the other end. Each cube marking scenario encodes a cube–isosurface intersection pattern or configuration, that correspond to a specific set of triangles that represent part of the surface within that cube. However, if complementary, rotational and reflective–mirror symmetries are considered the number of the intersection pattern topologies can greatly be reduced. These types of symmetries are illustrated in Fig. 3.3.

In particular, cubes C and $\hat{C}$ exhibit *complementary* symmetry when each corresponding vertex across the cubes has an inverse marking status. For example, if a vertex is marked in cube C, then the corresponding vertex in cube $\hat{C}$ is unmarked, and vice versa. For the MC algorithm, complementary symmetric cubes yield identical patterns of intersections between the cube and the isosurface, although the surface normals are opposite.

Additionally, cubes C and C' are considered *rotationally* symmetric if cube C can be rotated through a series of transformations R, so that the vertex markings at each transformed vertex in the new orientation are identical to the vetrex markings at the same position in C'. Rotationally symmetric cubes have equivalent cube–isosurface intersection patterns and the triangulation patterns vary only rotationally by the aligning transformation R.

Lastly, cubes Q and Q^* are *reflection* or *mirror* symmetric if their vertex markings are symmetric about some face of Q. [38, 39]. These relationships are illustrated in Fig. 3.3, where cubes C and $\hat{C}$ demonstrate complementary symmetry, C and C' rotational symmetry and Q and Q^* reflection/mirror symmetry.

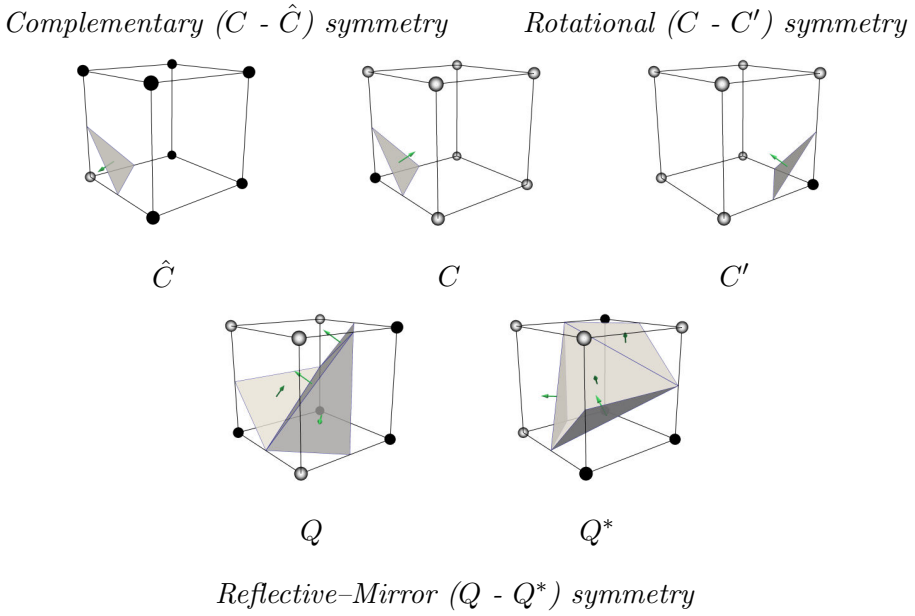


Figure 3.3: *Three Types of symmetries (Complementary, Rotational, Reflective–Mirror) in the case scenarios.*

The number of cube–isosurface intersection patterns or configurations, under various symmetries, is outlined in Table 3.1, following the methodology outlined by Roberts and Hill. [40].

Table 3.1: *Number of intersection patterns (configurations) considering different symmetries*

Symmetry exploited	# topologies
None	256
Complementation	128
Rotation	23
Rotation + reflection	22
Rotation + complementation	15
Rotation + complementation + reflection	14

The original MC considers rotational and complimentary symmetry, which results in the reduction from 256 to 15 unique cube–isosurface intersection cases. These intersection topologies are built offline prior to the application of the algorithm. In Fig. 3.4, the original MC look–up table is presented as introduced in [22]. Note that mirror symmetry is not considered: configurations 11 and 14 are treated independently.

The isosurface–edge intersection locations can be estimated using an interpolation technique. Standard MC employs linear interpolation to estimate the intersection

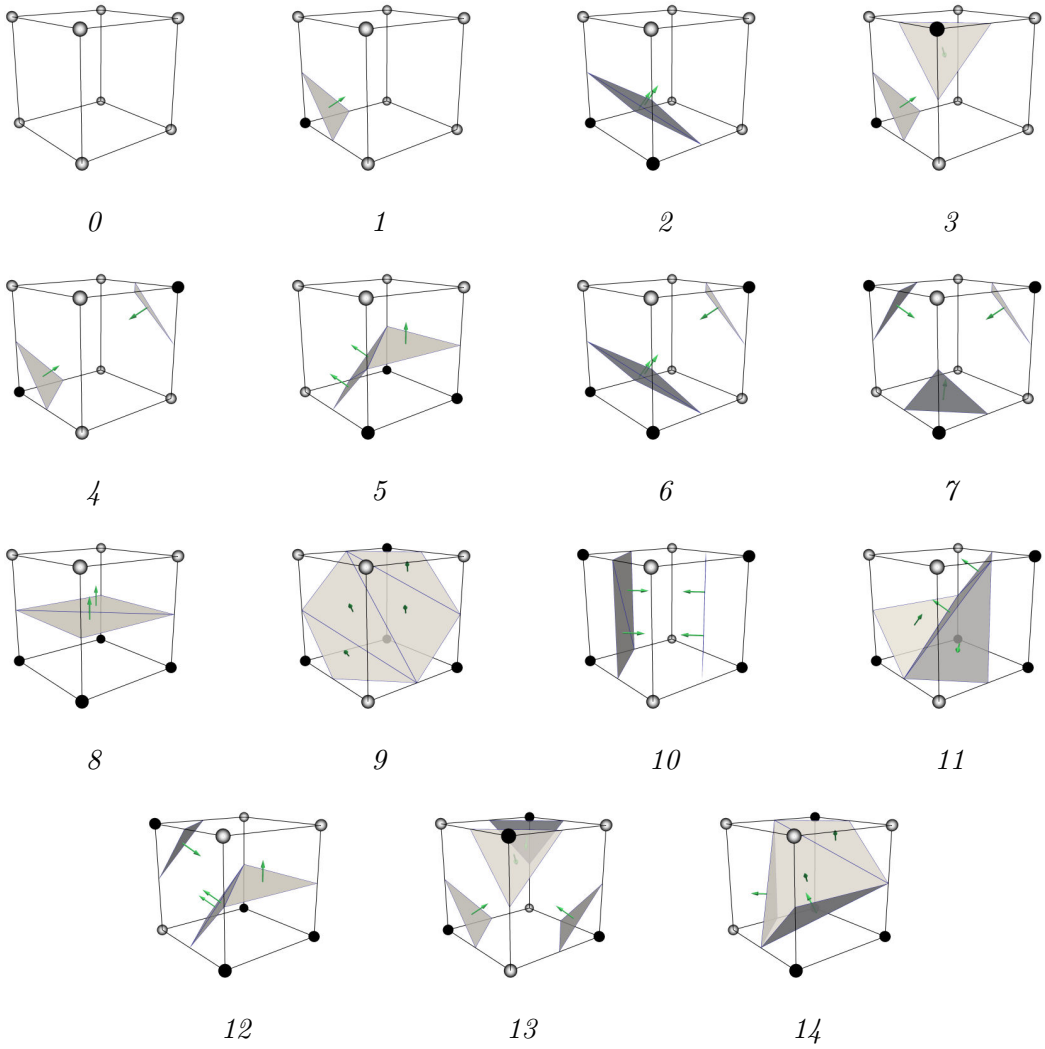


Figure 3.4: *The original 15 triangulation configurations of the Marching Cubes algorithm (similar to [22]). Nodes in white are outside the isosurface, whereas in black inside the isosurface. Generated triangles in grey with their respective normals depicted as a green arrow. A maximum of 4 triangles can be generated from the original MC algorithm.*

point for each intersected edge. The same interpolation scheme is incorporated in this thesis, and is expressed as follows: If a unit-length edge e has end points V_1 and V_2 each with their own scalar value f_1 and f_2, respectively, then given an isovalue α, the position vector of the intersection $\vec{r}_{\text{is}} = (r_{\text{is}_x}, r_{\text{is}_y}, r_{\text{is}_z})$ has coordinates of the form:

$$\vec{r}_{\text{is}} = \vec{r}_{V1} + \rho(\vec{r}_{V2} - \vec{r}_{V1}) \tag{3.1}$$

where

$$\rho = \frac{\alpha - f_1}{f_2 - f_1} \tag{3.2}$$

The last step in MC regards the generation of triangular facets that represent an approximation to the isosurface. The vertices of the triangles are the isosurface–edge intersection points and, hence, the collection of the triangular facets across all the cubes forms the triangular surface mesh (or meshes) that defines the isosurface.

3.1.2 Handling Ambiguities

After the introduction of the MC algorithm, extended literature has been published regarding various inaccuracies of the topologies produced by the algorithm [41, 42, 43, 44, 45, 46]. The latter are due to the multiple ways some of the scenarios can be facetized. Accordingly, these topologies are referred to as *ambiguous*. Ambiguities are classified as *face ambiguities* and *internal ambiguities* and relative resolution schemes are proposed.

Ambiguous face

Any face of the cube where there are both two diagonally opposite marked grid points and two diagonally opposite unmarked nodes is an ambiguous face. In this case all four edges of the ambiguous face are intersected by the isosurface. For the ambiguous face, the information is insufficient to decide how to connect the vertices on the edges. The face ambiguity is represented in 2D in figure 3.5. An ambiguous face shared by two adjacent cubes will result in inconsistent facetizations as depicted in figure 3.6 which is referred in literature as the *'hole problem'*, initially highlighted in [41].

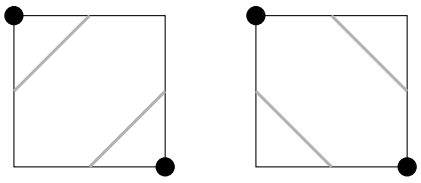


Figure 3.5: *An ambiguous face. Nodes inside the isosurface in black. Two different ways the isosurface (gray lines) can intersect the face.*

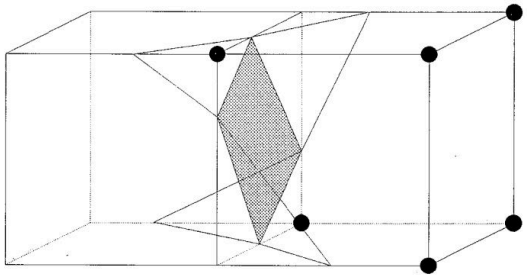


Figure 3.6: *A hole 'grey' generated by the traditional MC algorithm.*

Many resolution schemes have been proposed for constructing topologically correct isosurfaces when encountering ambiguous faces. Some approaches propose criteria for the disambiguation of the cases, based on an analysis of the variation of the scalar variable across the ambiguous face [42, 43]. Since face ambiguity primarily arises from the exploitation of complementary symmetry, a straight forward approach is to only use rotational symmetry [44]. A simple and effective solution extends the original 15 MC cases by adding additional complementary cases. These cases are designed to be compatible with neighboring cases and prevent the creation of holes

on the isosurface [45]. The 8 complementary cases required, namely 0c, 1c, 2c, 3c, 4c, 5c, 6c, 7c are reproduced in Fig. 3.9. Three examples of the 'hole' problem and their resolution with the extended triangulation are shown in Fig. 3.7.

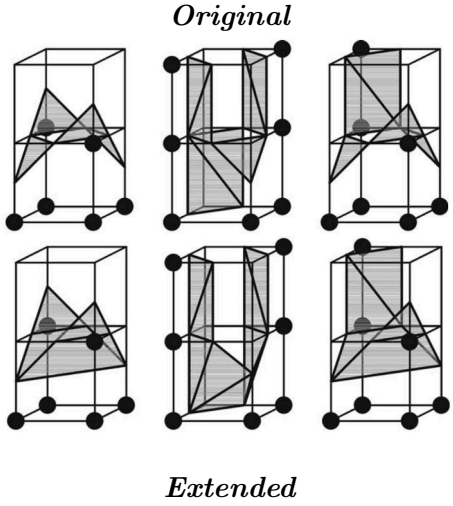


Figure 3.7: *Three examples (from left to right) of the 'hole problem' and the resolution with the extended triangulation cases. Nodes in black considered inside the isosurface, generated triangles in grey. From [39].*

Internal ambiguity

Even in the absence of ambiguous faces, a cube can still exhibit internal ambiguity. This type of ambiguity doesn't lead to topological errors; however, it can result in an inaccurate representation of the isosurface. Internal ambiguity may occur in cases that two diagonally opposed vertices of the cube are inside the isosurface (Cases 4, 6, 7, 10, 12, and 13 of the original MC algorithm [22]). Given the absence of information regarding the isosurface's behavior within the cube, one might hypothesize that the vertices are connected within the cell. Some approaches use 'linking' facets to create 'tubes' or 'tunnels' that bridge different segments of the isosurface. Figure 3.8 depicts the standard facetization for Case 4 and an alternative facetization approach that avoids disjoint facets [46]. In this thesis, internal ambiguity within the MC algorithm is not addressed. This is due to its infrequent occurrence and the high resolution of the samples being processed, which mitigates the potential for such ambiguity. Furthermore, the computational processing time required to handle internal ambiguity is deemed excessive.

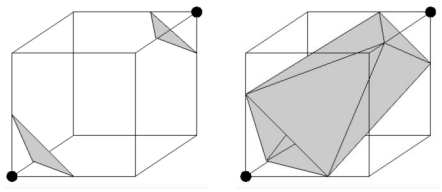


Figure 3.8: *Case 4 internal ambiguity: (a) disjoint faces, (b) joint faces. From [43].*

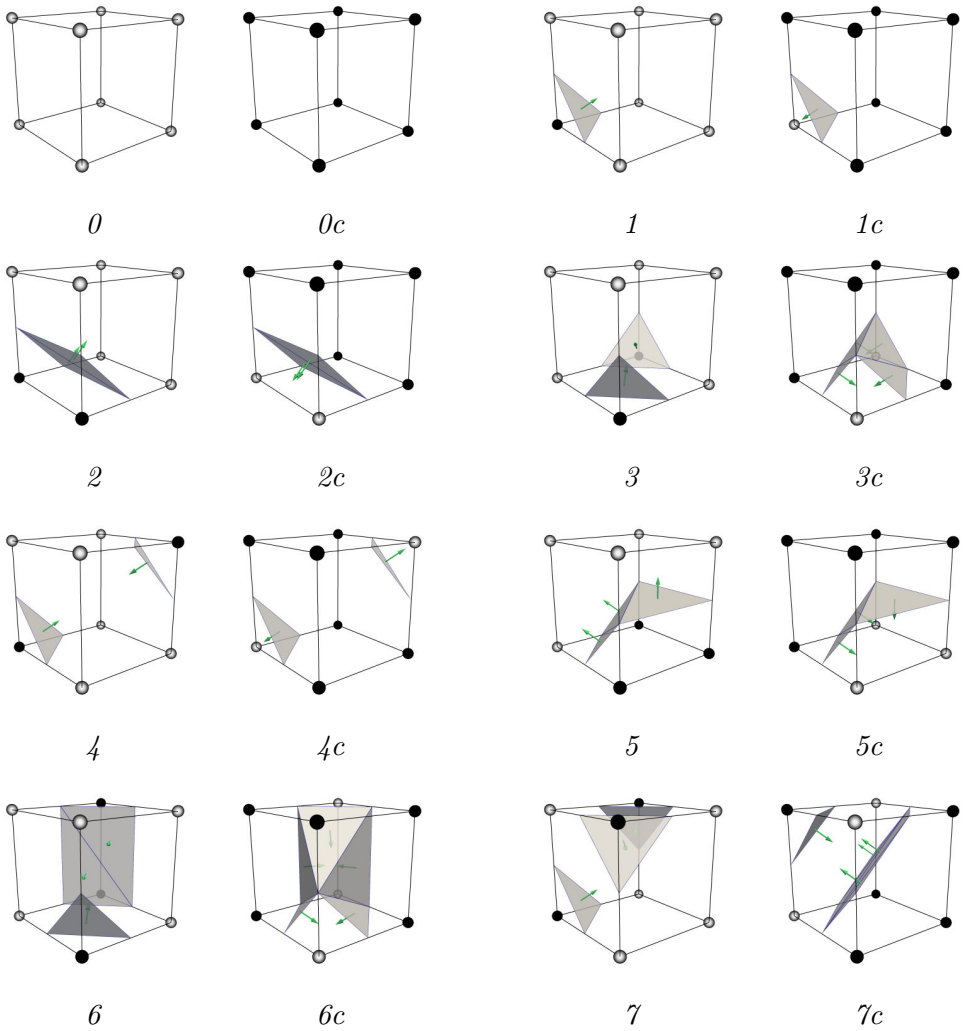
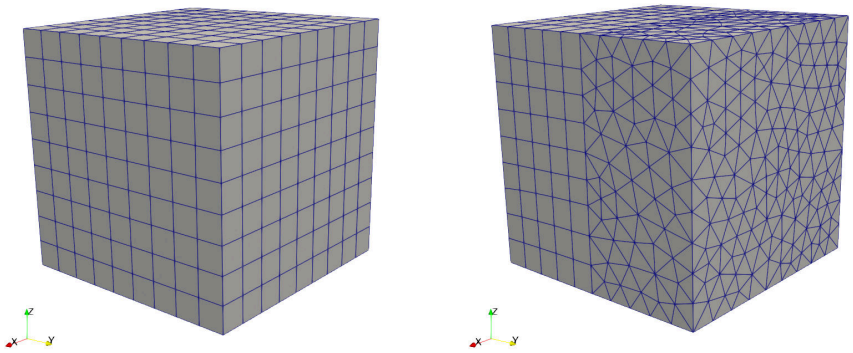


Figure 3.9: *Complementary cases of the original 0, 1, 2, 3, 4, 5, 6, 7 configurations to directly resolve ambiguities.*

3.2 Adaptation for Hybrid Grids

In the context of CFD analysis and TopO, one often deals with hybrid grids that combine various element types. While the MC algorithm primarily focuses on cubic grids, its principles can be adapted for different cell shapes [37, 47]. In the context of this thesis, the adaptation of the algorithm to also handle hybrid grids, which contain hexahedra, prisms–wedges, pyramids, tetrahedra, and quadrilaterals is presented. Both types of grids are visually depicted in Fig. 3.10.



(a) *(b)*

Figure 3.10: *Two types of computational grids. (a) A 3D structured grid with only hexahedral elements. (b) A 3D unstructured grid containing hexahedra, prisms, pyramids & tetrahedra.*

3.2.1 Marching Elements - Formulation of cases

Hexahedra

Hexahedral grids consist of six–faced, eight–node cells essentially deformed, but topologically consistent, cubes. The algorithm's adaptation for hexahedra grids adheres to the MC algorithm's core outline. Each vertex of the hexahedron is assigned a binary value, 1 for inside, 0 for outside the isosurface, hence the 8–bit hexahedron index is determined. For hexahedron edges that have one vertex inside and one vertex outside the surface, the exact intersection points of the surface with these edges are calculated by means of linear interpolation. The intersection points are then used to create the triangles that represent the section of the isosurface inside the hexahedron. The 23 extended triangulation configurations for the cubes are utilized for the hexahedra, presented in Fig. 3.11.

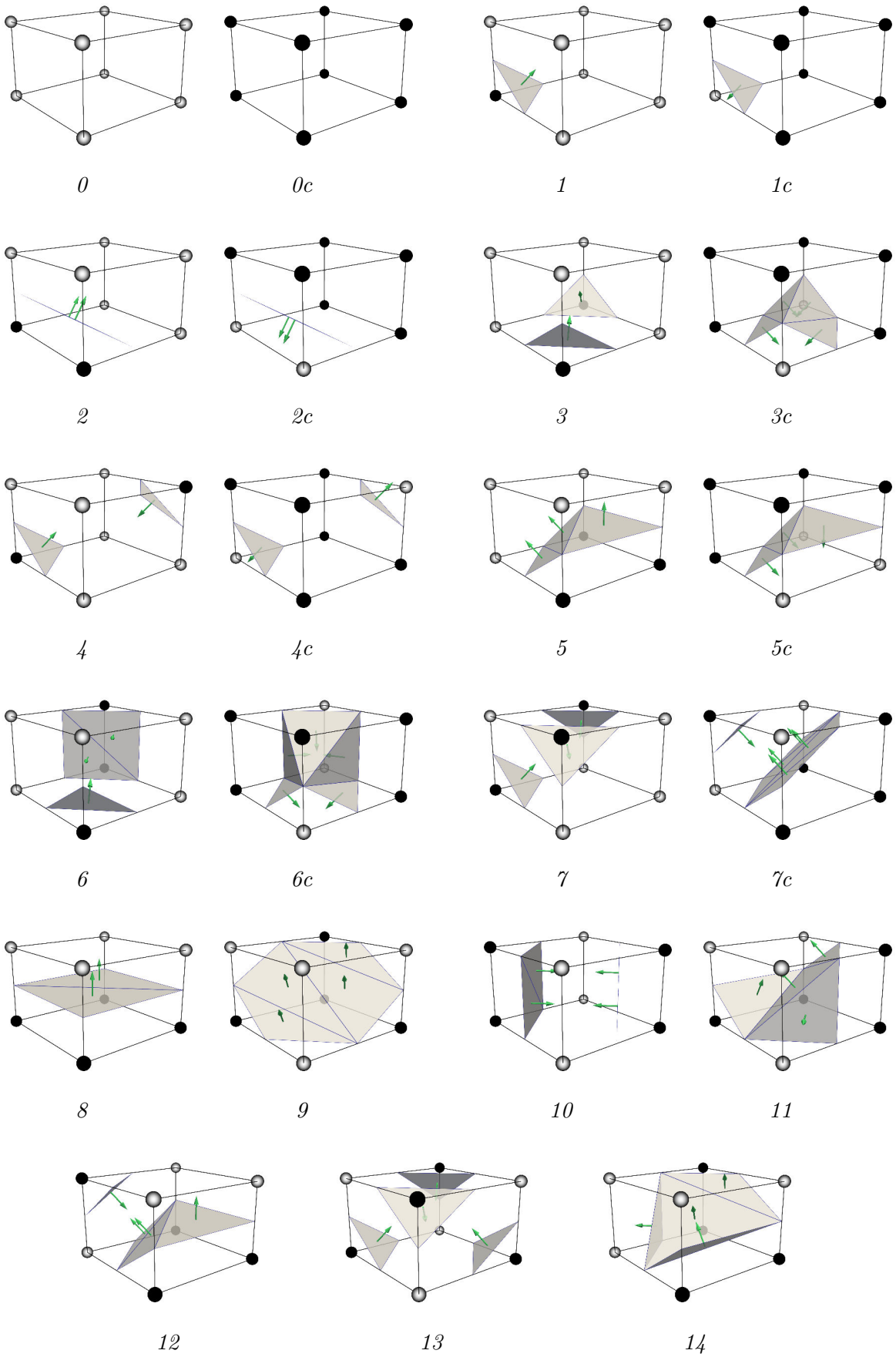


Figure 3.11: *The Marching Hexahedra 23 triangulation configurations. Generated triangles in grey with their respective normals depicted as a green arrow. A maximum of 5 triangles can be generated from the extended MH algorithm.*

Prisms

A similar formulation is assumed in the case of five–faced, six–node prisms, or often referred as wedges. A 6–bit prism index is used to access the $2^6 = 64$ cases, reduced to 15 when rotational symmetries are incorporated. The Marching Prisms 15 triangulation configurations are depicted in Fig. 3.12.

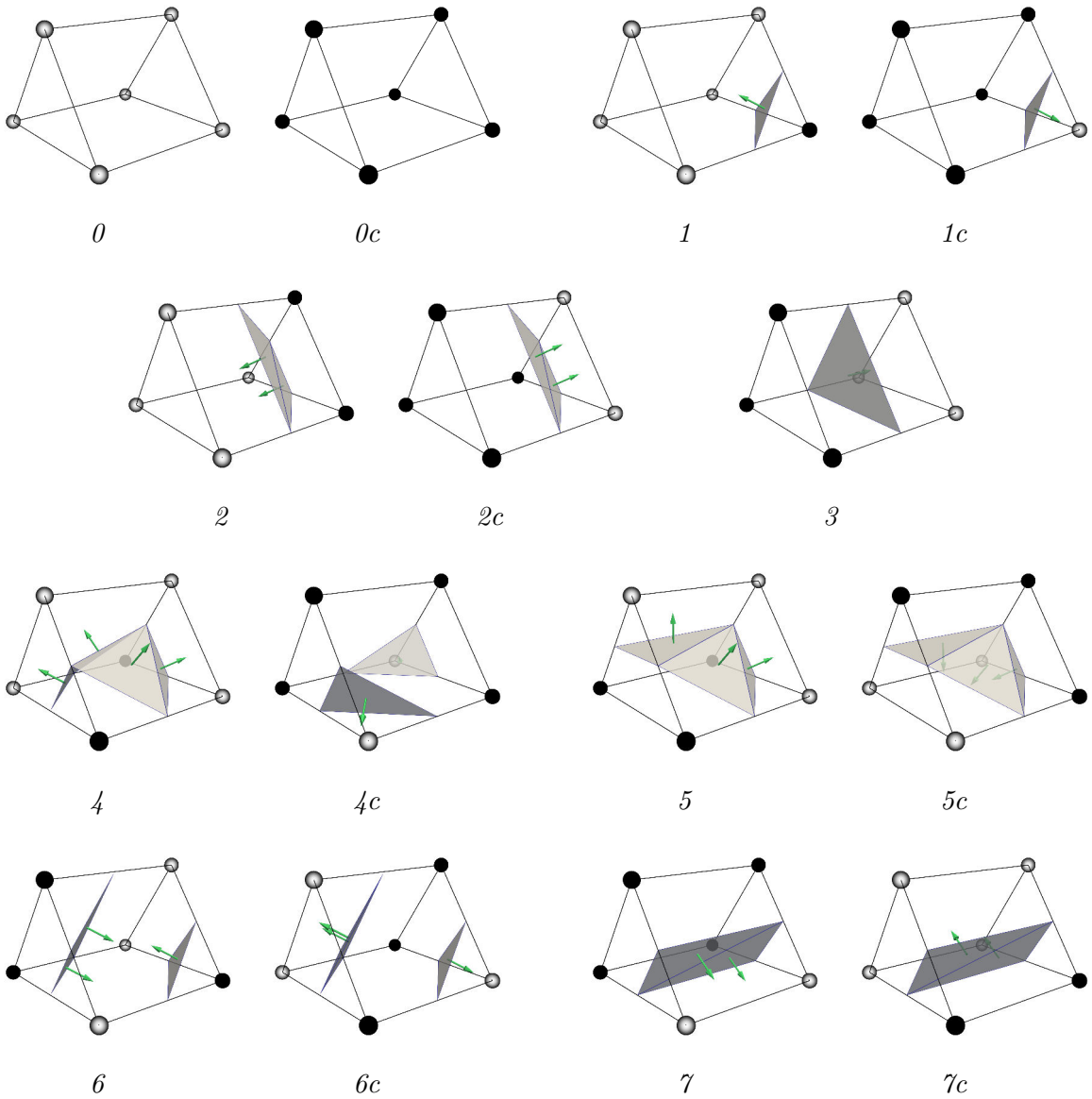


Figure 3.12: *The Marching Prisms 15 triangulation configurations*

Pyramids

Pyramids are often used in grids as transitional elements between hexahedra and tetrahedra. Each pyramid is made up of 5 vertices, therefore in Marching Pyramids a 5–bit index is incorporated, encoding the $2^5 = 32$ cases which are reduced to 12 when rotational symmetries are accounted for. The Marching Pyramids 12 triangulation configurations are demonstrated in Fig. 3.13.

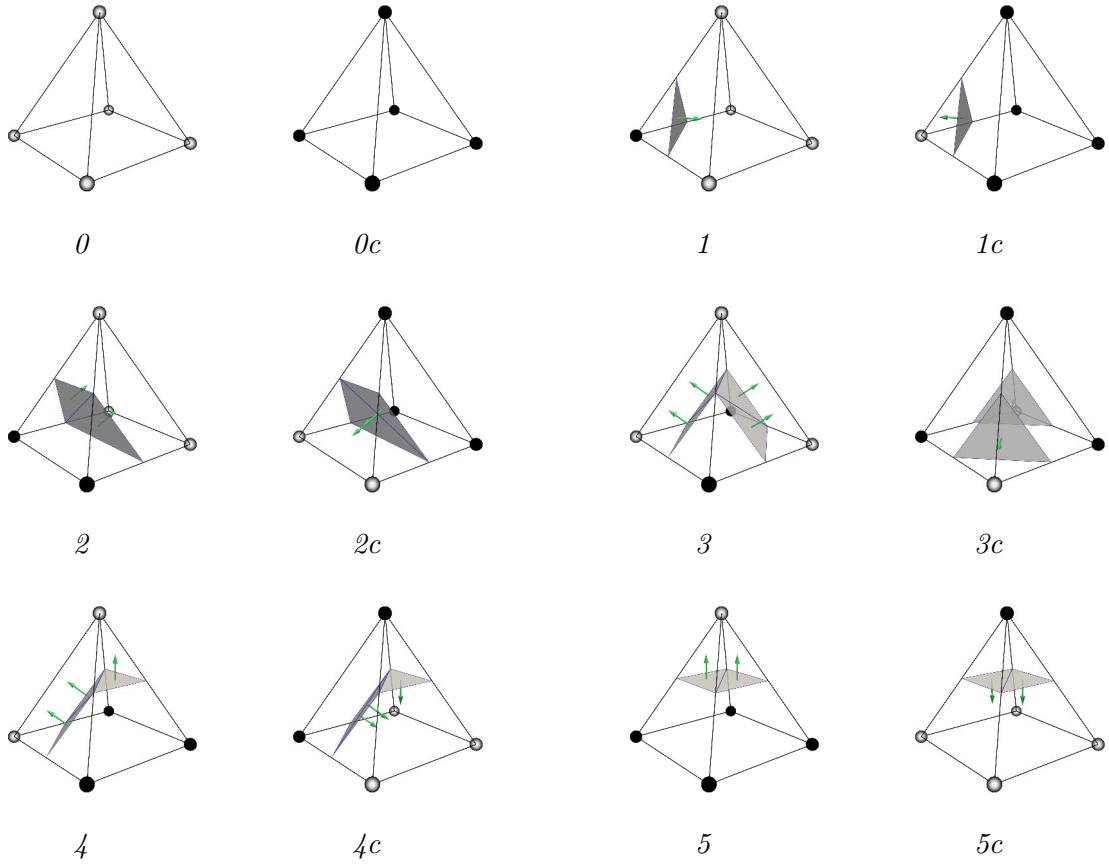


Figure 3.13: *The Marching Pyramids 12 triangulation configurations*

Tetrahedra

Tetrahedra are the simplest 3D polyhedra, with four nodes and four faces. For a tetrahedral cell, $2^4 = 16$ cases exist, of which only 5 configurations are unique due to rotational symmetries. Figure 3.14 reproduces the 5 unique tetrahedron–isosurface configurations.

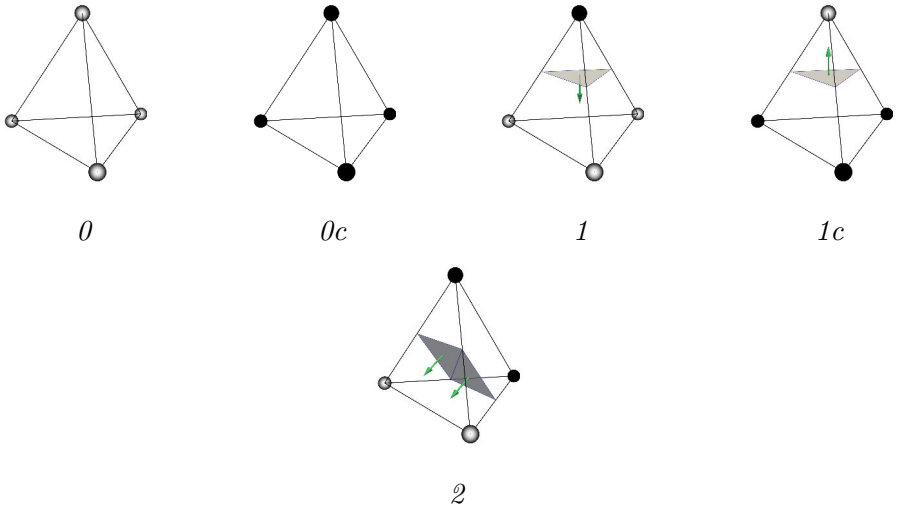


Figure 3.14: *The Marching Tetrahedra 5 triangulation configurations*

Quadrilaterals

In CFD analysis, wall boundary conditions are used to simulate the interaction between the fluid and solid boundaries within the computational domain. They are crucial for accurately predicting the flow behavior and properties near the walls. In order to include the defined boundary, a Marching Quadrilaterals scheme is proposed. Since there are four vertices in quadrilaterals there exist $2^4 = 16$ possible marking scenarios for the vertices. Attention is given so that the configurations are compatible with existing configurations of the boundary cells of the grid, since the boundary is composed of the boundary faces of those cells. Note that in this formulation of quadrilateral cases the vertices also take part in the triangulation alongside the edge–isosurface intersections. Considering rotations, 6 unique configurations exist depicted in Fig. 3.15.

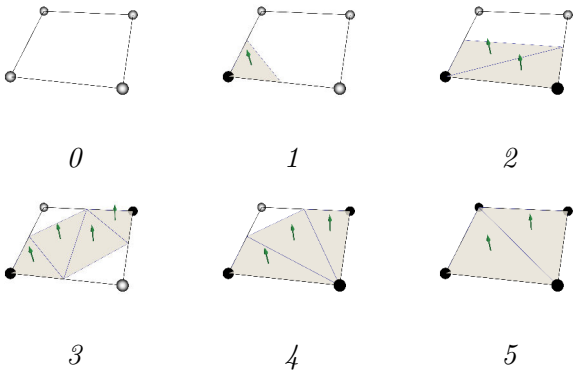


Figure 3.15: *The Marching Quadrilaterals 6 triangulation configurations. Boundary nodes inside the isosurface in black, outside the isosurface in white. Generated triangles in grey with their normals (green arrow).*

3.2.2 Thesis Impact

The above methodology has been rigorously studied and implemented in the current thesis, thus resulting in specific contributions summarized below:

- Adaptation of the Marching Cubes triangulation look–up tables from the literature [23] in the C++ implementation. Formulation of the appropriate Data Structure and implementation of the algorithm for the case of cubic structured grids.
- Formulation of triangulation look–up tables A for the other element types and the boundary. Adaptation of the Data Structure and revised algorithm implementation for the case of hybrid grids.
- Algorithm integration in two relevant environments, namely laboratory in–house environment & within the open–source CFD toolbox of OpenFOAM©.
- Assessment of implemented algorithm in a variety of use case scenarios.

Chapter 4

Algorithmic Implementation

In the current chapter we present in detail the implementation of the Topology–to–Shape Transition using Marching Algorithms. At first a general overview of the algorithm is given, and its components are analyzed in detail. Also the mathematical schemes and the Data Structure incorporated in this thesis are outlined. Attention is payed in the capability for parallelization and a running example is introduced as a vehicle to demonstrate and explain the algorithm's operation.

4.1 Algorithm overview

The first stage of the algorithm is the input of the computational mesh. The mesh geometry can be loaded either from the in–house format or from the OpenFOAM© environment. Following that the porosity β field is loaded as a cell–centered field or as a vertex–centered field. In cases where the β field is cell-centered, an additional step is required to make the data compatible with Marching Algorithms, that require the scalar field at the vertices as discussed in Chapter 3. A volume–weighted interpolation scheme is employed to estimate the field at the vertices. The Data Structure is also generated and unique edgeIDs are attributed to the edges of the mesh. Each element of the mesh is then processed with the appropriate Marching Algorithm, according to its type. The final step of the process is the generation of the output file that contains the collection of the triangles that form the FSI surface mesh representation. The entire process is summarized in a block diagram presented in Fig. 4.1.

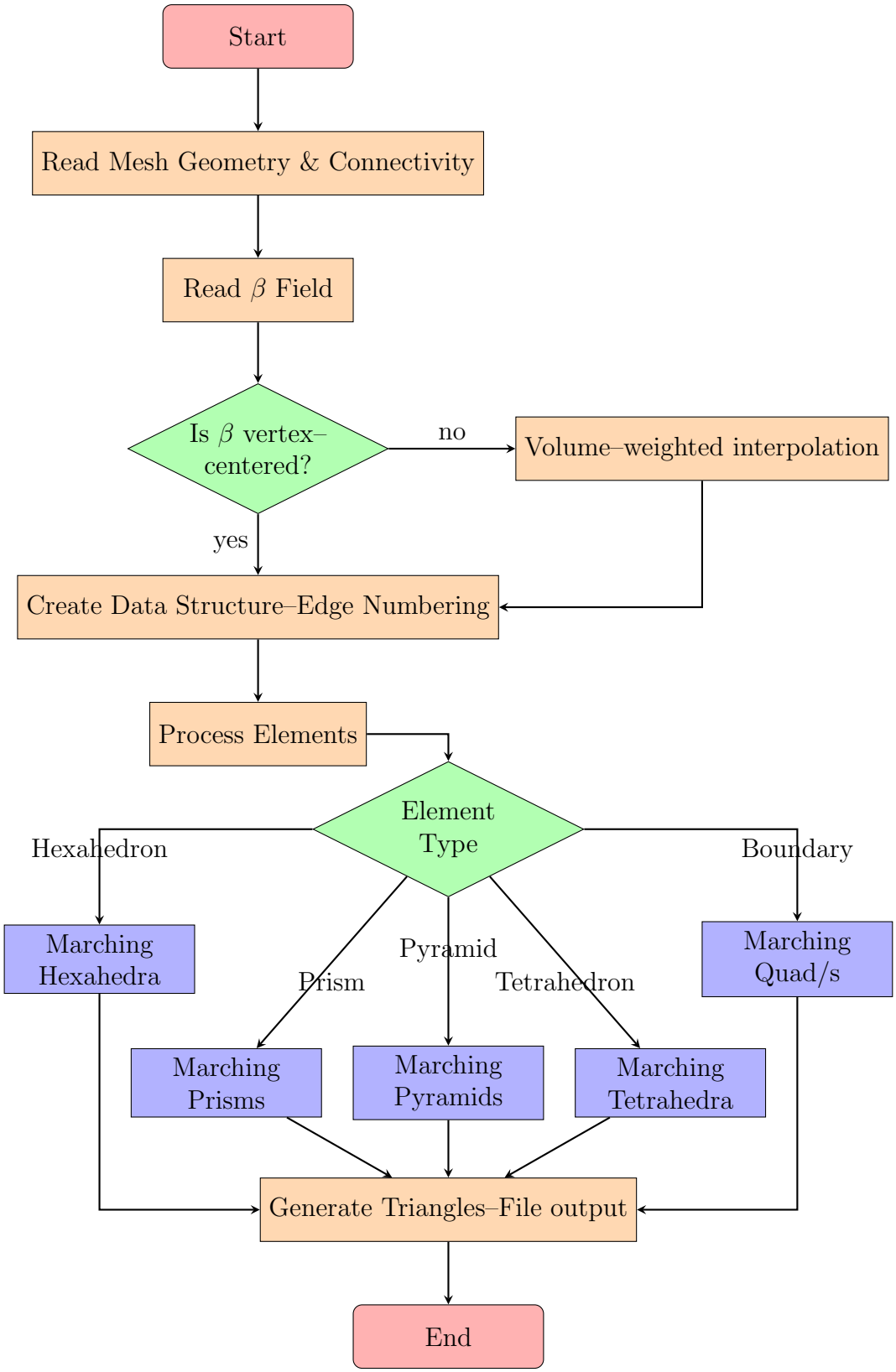


Figure 4.1: *Flow chart of the proposed algorithm for the Topology-to-Shape transition. Processes in orange, decisions in green, Marching Algorithms in blue.*

Each process of the Block Diagramm of Fig. 4.1 is described in detail.

4.2 Mesh Geometry and Connectivity Input

The developed algorithm is designed read and handle two input file formats, (1) the in-house input file format of the PCOpt/NTUA or (2) the OpenFOAM© file format from the working directory of the case.

4.2.1 The in–house input file format of PCOpt/NTUA

The in–house input file format is describes the computational grid in a set of two files, which are:

- the `.nod` file, containing the x, y, z coordinates of all the nodes.
- the `.hyb` file, containing the *element connectivity* of the grid.

The .nod file is the first file describing the computational grid holding the x, y, z coordinates of all the nodes. In the first line of the file the total number of nodes is written, while in the next three lines the x, y and z coordinates of each node are written as shown in Fig 4.2.

```
1:  nbNodes
2:  nodeX^1 nodeX^2 nodeX^3 ... nodeX^nb
3:  nodeY^1 nodeY^2 nodeY^3 ... nodeY^nb
4:  nodeZ^1 nodeZ^2 nodeZ^3 ... nodeZ^nb
```

Figure 4.2: *Representation of the* `.nod` *file structure where* $nodeX^i$, $nodeY^i$ *and* $nodeZ^i$ *are the x, y and z coordinates, respectively, of the* i^{th} *node.*

The .hyb file is the second file describing the computational grid, containing the element connectivity. In the first line of the file the total numbers of tetrahedra ($nbTet$), pyramids ($nbPyr$), prisms ($nbPri$) and hexahedra ($nbHex$) is written. In the following lines, the nodal IDs of each element are written as shown in Fig 4.3, where $iTet^1$ $iTet^2$ $iTet^3$ $iTet^4$, for example, are the nodal IDs of the nodes of the i^{th} tetrahedron. The nodal IDs of each element type are written in a separate line.

1: $nbTet\ nbPyr\ nbPri\ nbHex$
2: $\ldots i^{th}Tet^1\ i^{th}Tet^2\ i^{th}Tet^3\ i^{th}Tet^4\ \ldots$
3: $\ldots i^{th}Pyr^1\ i^{th}Pyr^2\ i^{th}Pyr^3\ i^{th}Pyr^4\ i^{th}Pyr^5\ \ldots$
4: $\ldots i^{th}Pri^1\ i^{th}Pri^2\ i^{th}Pri^3\ i^{th}Pri^4\ i^{th}Pri^5\ i^{th}Pri^6\ \ldots$
5: $\ldots i^{th}Hex^1\ i^{th}Hex^2\ i^{th}Hex^3\ i^{th}Hex^4\ i^{th}Hex^5\ i^{th}Hex^6\ i^{th}Hex^7\ i^{th}Hex^8\ \ldots$

Figure 4.3: *Representation of the .hyb file structure.*

4.2.2 The OpenFOAM© mesh description format

A mesh description in OpenFOAM© consists of the following:

- **List of points:** Point index is determined from its position in the list
- **List of faces:** A face is an ordered list of points, the order defines the face normal.
- **List of owner-neighbour** addressing (defines owner cell of face and its neighbour pointed by its normal)
- **List of boundary patches**, grouping external faces

The OpenFOAM© mesh is stored in the constant/polyMesh sub-directory of a case. This polyMesh directory contains files that provide all the information above in order to define the mesh. The schematic representation of the mesh description is presented in Fig. 4.4. The algorithm can be integrated within an OpenFOAM© case environment to derive a mesh description and proceed with the FSI extraction.

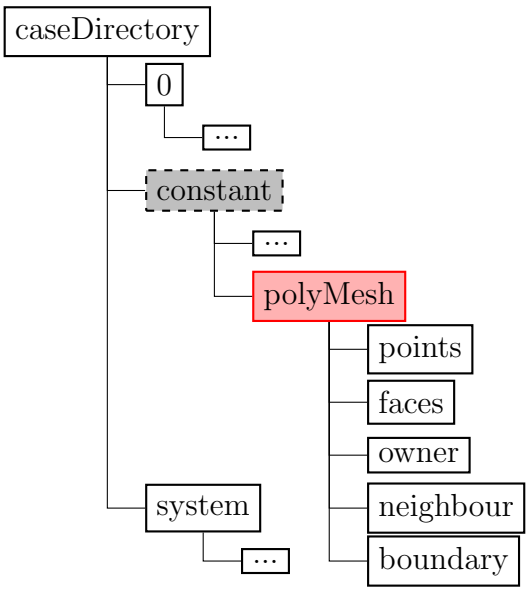


Figure 4.4: *An OpenFOAM© case mesh description.*

4.3 Volume weighted cell center to vertex interpolation

To interpolate a field from the cell centers to the vertices of the computational grid, a volume weighted interpolation scheme is used. The values at the vertices are calculated by the contributions from adjacent cells based on the volume.

Volume calculation

The volume of a cell in a three-dimensional space can be calculated using the *Divergence Theorem*, also known as the *Flux Theorem*. The theorem relates the flux of a vector field through a closed surface to the divergence of the field within the volume enclosed by the surface. The divergence theorem is given by:

$$\int_V \nabla \cdot \vec{F}\, dV = \int_S \vec{F} \cdot \vec{n}\, dS \tag{4.1}$$

where $\vec{F}$ is a vector field and S is the closed surface enclosing the volume V and $\vec{n}$ is the unit vector perpendicular to S at each point, pointing outwards of the cell.

When the surface S is composed of triangular elements, the calculation can be simplified.

For a triangular face with vertices at $\vec{r_A}$, $\vec{r_B}$, and $\vec{r_C}$, the centroid $\vec{r_G}$ is at:

$$\vec{r_G} = \frac{\vec{r_A} + \vec{r_B} + \vec{r_C}}{3} \tag{4.2}$$

and the area vector dS is:

$$dS = \frac{1}{2}(\vec{r_B} - \vec{r_A}) \times (\vec{r_C} - \vec{r_A}) \tag{4.3}$$

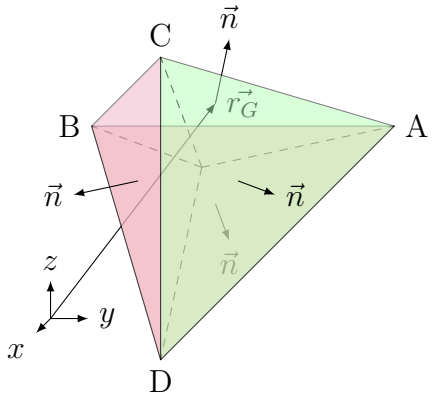


Figure 4.5: *Tetrahedron composed of triangular faces demonstrating the divergence theorem for volume calculation.*

Using the vector field $\vec{F} = \vec{r}$ (position vector) and its constant divergence in three dimensions:

$$\nabla \cdot \vec{r} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}\right) \cdot (r_x, r_y, r_z) = \frac{\partial r_x}{\partial x} + \frac{\partial r_y}{\partial y} + \frac{\partial r_z}{\partial z} = 1 + 1 + 1 = 3 \tag{4.4}$$

the volume V can be calculated via the individual contributions $\vec{r_G} \cdot \vec{n}\, dS$ of all the

triangles of the surface, yielding:

$$\int_V \nabla \cdot \vec{r}\, dV = \int_S \vec{r} \cdot \vec{n}\, dS \rightarrow \boxed{V = \frac{1}{3} \sum_{\text{triangles}} \vec{r_G} \cdot \vec{n}\, dS} \tag{4.5}$$

Interpolation Scheme

The interpolated value of β at a node N, denoted as β_N, is calculated using a weighted average of the values β_i at the centers of the neighboring cells with volumes V_i:

$$\beta_N = \frac{\sum_{\text{cells}} \beta_i \cdot V_i}{\sum_{\text{cells}} V_i} \tag{4.6}$$

4.4 Data Structure

A great advantage of the element–by–element processing of the proposed algorithm is that each edge–isosurface intersection location $\vec{r}_{\text{is}}$ (Eq. 3.1) needs to be computed only once. The intersection point can be reused during later processing of neighbouring elements that share the edge. This observation dictates the need for a *Data Structure* to map all the edges of the computational grid and hold information for those that are intersected. Consequently, following the initial loading of the computational grid, the subsequent step involves establishing this data structure, which facilitates the comprehensive mapping of all grid edges.

4.4.1 Edge numbering

A local node and edge numbering is considered for each element type. In Fig. 4.6 the local numbering for Hexahedra, Prisms, Pyramids, Tetrahedra, and Quadrilaterals is established, whereas an algorithm that implements the edge numbering is outlined in Algorithm 1. The local numbering of an element's edges would be attributed to the edge map for the first element being processed. Each successive element being processed would consider at first the local numbering plus the current edgeID, kept track through a *counter*. If the edge is shared with processed elements and already numbered it would retain its original numbering. The edges of the elements are represented by the pairs of the nodeIDs that they connect, ensuring that the smaller nodeID is always first. The pair of nodeIDs is used as a key in a hash table to access the edgeMap that contains the edgeIDs. The above procedure ensures that each edge is assigned a unique edgeID.

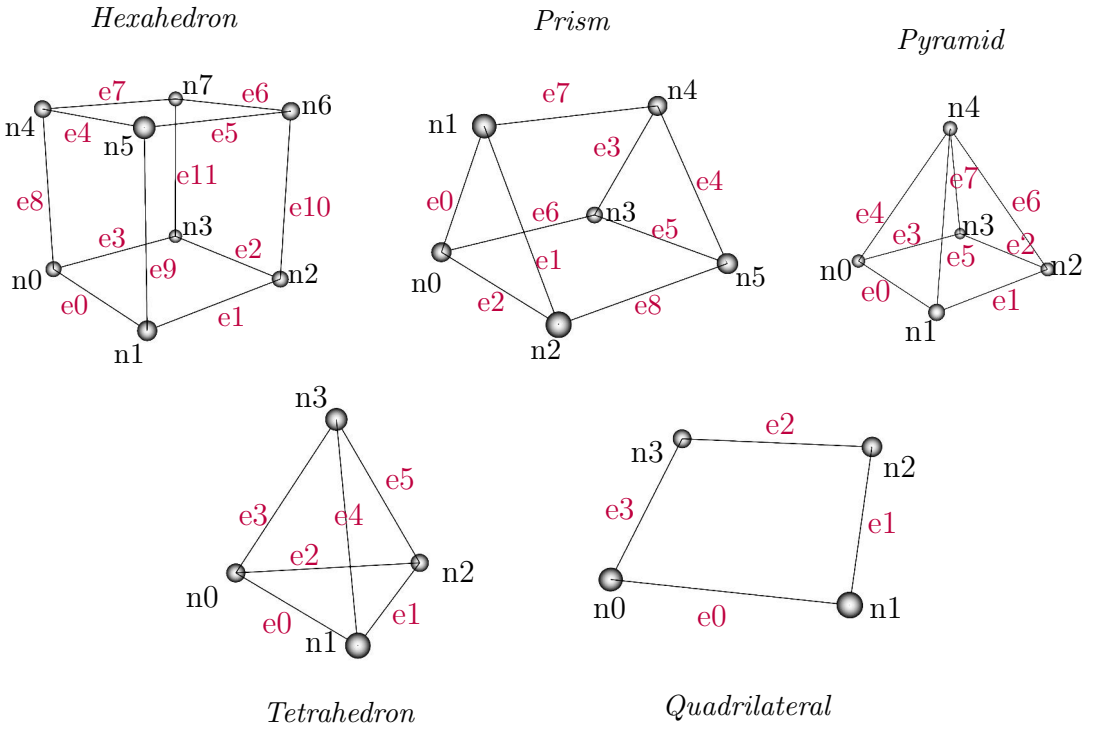


Figure 4.6: *Nodes and edges local numbering for each element type.*

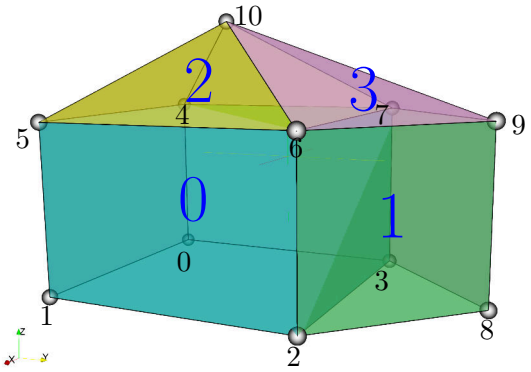
Algorithm 1 Data Structure Processing for Edges

```
1: procedure EdgeNumbering(elements, edgeMap)
2:     counter ← 0
3:     for all elements do
4:         for every edge in element.edges do
5:             nodePair ← {min(edge.nodeID), max(edge.nodeID)}
6:             if nodePair not in edgeMap then
7:                 edgeMap[nodePair] ← edgeId
8:                 edgeId ← counter
9:                 counter ← counter + 1
10:            end if
11:        end for
12:    end for
13: end procedure
```

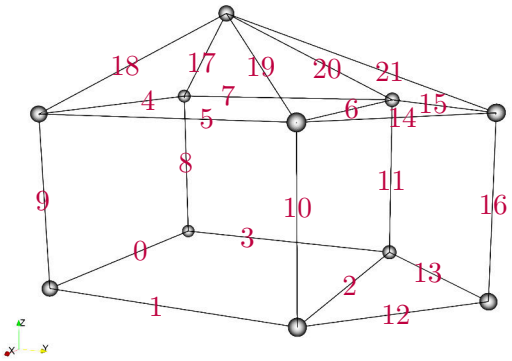
4.4.2 Running Example

Without loss of generality, a running example that refers to a simple object consisting of 4 element types (an hexahedron, a prism, a pyramid and a tetrahedron) is introduced to demonstrate the edge numbering produced by the Data Structure. In Fig. 4.7 the process of assigning unique edgeIDs to all the edges of the running example's grid is presented.

Element	Nodes
0:	(0, 1, 2, 3, 4, 5, 6, 7)
1:	(2, 8, 3, 6, 9, 7)
2:	(4, 5, 6, 7, 10)
3:	(6, 9, 7, 10)



(a) Element numbering in blue and Node numbering in black for running example



(b) Edge numbering in purple for the edgeMap created by the Data Structure for running example

edgeMap

Edge ID	Node ID Pair
0	(0, 1)
1	(1, 2)
2	(2, 3)
3	(0, 3)
4	(4, 5)
5	(5, 6)
6	(6, 7)
7	(4, 7)
8	(0, 4)
9	(1, 5)
10	(2, 6)
11	(3, 7)
12	(2, 8)
13	(3, 8)
14	(6, 9)
15	(7, 9)
16	(8, 9)
17	(4, 10)
18	(5, 10)
19	(6, 10)
20	(7, 10)
21	(9, 10)

(c) EdgeMap created by the Data structure where unique edgeIDs are assigned to the edges of the running example. Each edge is identified by the pair of nodeIDs it connects with the minimum nodeID first.

Figure 4.7: *Running example containing 4 elements, an hexahedron in blue, a prism in green, a pyramid in yellow and a tetrahedron in pink. Each element is characterized by its nodes in the specific order. To create the Data Structure the algorithm parses each element and assigns unique edgeIDs to newly encountered edges. The element and node numbering presented in (a) and the edge numbering in (b). The created edgeMap that contains the information of the edges is presented in (c).*

4.5 Element processing – Generation of Triangles

After the computational grid is loaded from the input files and the Data Structure has been created, an initialization of three arrays takes place. These arrays facilitate the comprehensive tracking of which edges have been processed, hence a new vertex or Iso–Node has been calculated on them, and the organized storage of those Iso–Nodes for later retrieval. In more detail the algorithm initializes:

- An array of booleans, `edgesBool`, to track processed edges.
- An array, `edgesCounter`, which stores IDs of vertices calculated by means of linear interpolation between the pair of nodes of an edge.
- A array, `Isonodes`, holds the coordinates of these vertices.

This approach ensures that no duplicate nodes are produced and allows for the parallel processing of elements, accessing and storing information for the current state in a unified environment for the whole grid. For each element being processed a binary Index is calculated based on which nodes are flagged or unflagged with the logic explained in Section 3.1.1. The procedure for the calculation of an element's Index is outlined in Algorithm 2.

Algorithm 2 Determine Element's Index

```
1: procedure ElementIndex(elements)
2:     for every element in elements do
3:         ElementIndex ← 0
4:         for every node in element.nodes do
5:             if node.value < isosurface_value then
6:                 ElementIndex ← ElementIdnex | (1 ≪ node.position)    ▷ Set bit
7:             end if
8:         end for
9:     end for
10: end procedure
```

The operation `ElementIndex ← ElementIdnex | (1 ≪ node.position)` at line 6 of the algorithm, sets the bit at 'node.position' in 'ElementIndex'. This is achieved by left–shifting 1 by 'node.position' bits, resulting in a binary number where only the bit at 'node.position' is set to 1. This number is then combined with the current 'ElementIndex' using the bitwise OR operator. Given the calculated index the algorithm consults pre–compiled look–up tables, listed in Appendix A, that dictate in which edges a vertex should be interpolated and how to triangulate the created vertices to obtain the configuration. With the running example introduced earlier in Fig. 4.7, the conduct of the algorithm will be explored. If we assume that node 7 of the grid is inside the isosurface, thus it is flagged, then every element will have a flagged node, as depicted in black in Fig. 4.8.

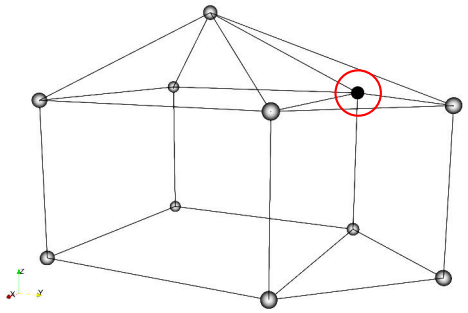


Figure 4.8: *Running example with the assumption of node 7 inside the isosurface/flagged.*

For simplicity in the example the elements will be processed sequentially, one element at a time. In reality many elements can be processed in parallel on different CPUs, all accessing and modifying the same Data Structure. The processing of the elements is depicted in Fig. 4.9. The processing of the element involves the (a) calculation of its index and the retrieval of the configuration, (b) the update of the edgeMap, (c) the interpolation of new nodes on the edges, (d) the generation of the appropriate triangles.

(a) Calculation of element's index, retrieval of configuration and transition to global edges.

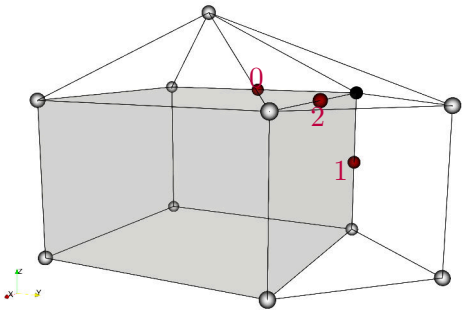
Nodes								
Local	7	6	5	4	3	2	1	0
Global	7	6	5	4	3	2	1	0
Index	1	0	0	0	0	0	0	0

Edges	
Local	7-11-6
Global	7-11-6

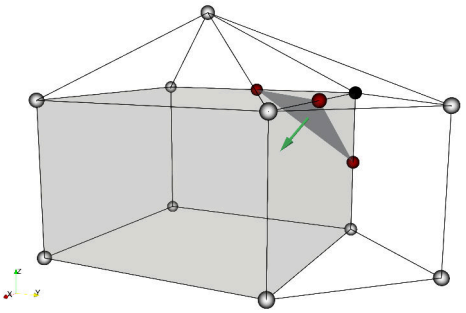
(b) EdgeMap components update

edgeMap

Edge ID	Bool	Counter
⋮	⋮	⋮
6	$\emptyset \rightarrow 1$	2
7	$\emptyset \rightarrow 1$	0
⋮	⋮	⋮
11	$\emptyset \rightarrow 1$	1
⋮	⋮	⋮



(c) Generation of new Nodes with interpolation scheme

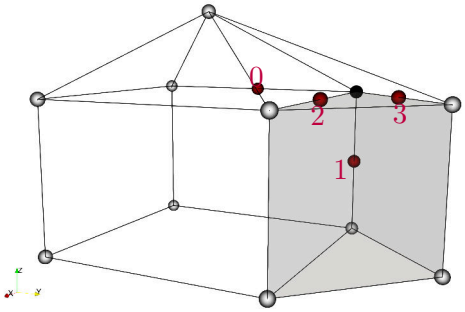


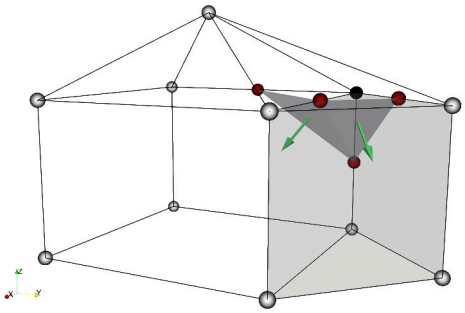
(d) Triangle formation with ordered connection of Nodes

Figure 4.9: *Processing of the elements of the running example. Hexahedron being processed first. Sequence of events in the algorithm (a)–(b)–(c)–(d).*

Local Nodes	5	4	3	2	1	0
Global Nodes	7	9	6	3	8	2
Index	1	0	0	0	0	0

Local edges	4-5-8
Global edges	15-6-11



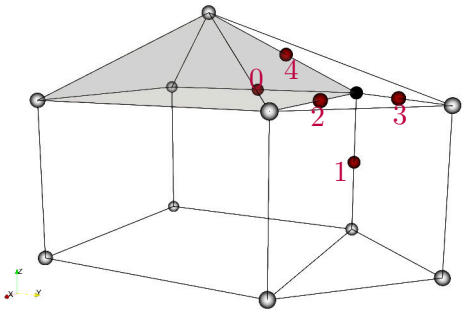


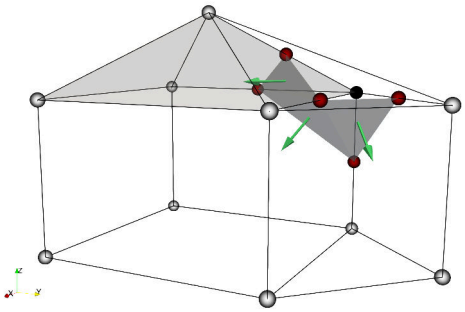
edgeMap

Edge ID	Bool	Counter
⋮	⋮	⋮
6	1	2
7	1	0
⋮	⋮	⋮
11	1	1
⋮	⋮	⋮
15	$\emptyset \rightarrow 1$	3
⋮	⋮	⋮

Local Nodes	4	3	2	1	0
Global Nodes	10	7	6	5	4
Index	0	1	0	0	0

Local edges	2-7-3
Global edges	6-20-7





edgeMap

Edge ID	Bool	Counter
⋮	⋮	⋮
6	1	2
7	1	0
⋮	⋮	⋮
11	1	1
⋮	⋮	⋮
15	1	3
⋮	⋮	⋮
20	$\emptyset \rightarrow 1$	4
⋮	⋮	⋮

Figure 4.9: *Processing of the next two elements: (Top) Prism, (Bottom) Pyramid.*

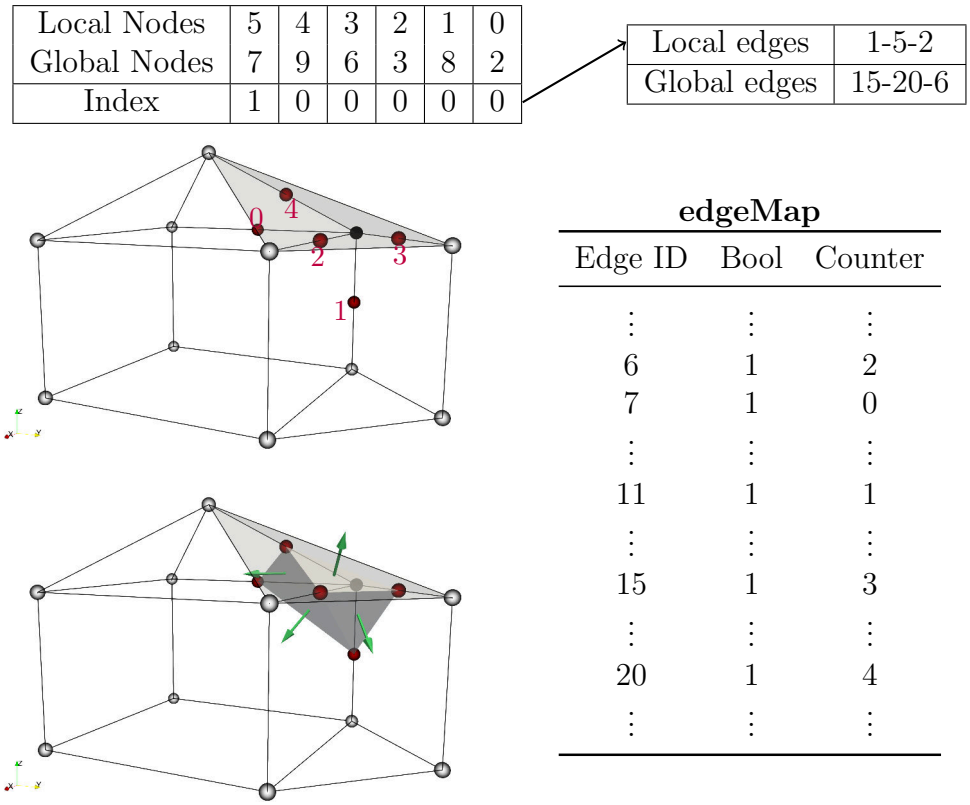


Figure 4.9: *Processing of the last element, the tetrahedron. No new nodes need to be generated and the algorithm utilizes the ones already calculated and stored at the edgeMap.*

4.6 File output

The output file generated by the algorithm contains the collection of triangles comprising the surface mesh of the FSI. The output file can take the form of the in-house file format or the form of an STL (STereoLithography) file compatible with a broad range of software and hardware used in Computer Aided Design (CAD) and Manufacturing (CAM). The STL also consists the most popular surface input format of (commercial) mesh generators. Thus, the surface can be provided directly to a mesh generator for re-meshing. It can also be exported to a VTK (Visualization Toolkit) file format for visualization in ParaView.

Chapter 5

Applications

In this chapter, the proposed algorithm is demonstrated in a number of sample computational grids and scenarios, ranging from simple geometric shapes to industrial applications. These applications illustrate the algorithm's versatility and effectiveness in different contexts. Starting with basic geometric forms, the algorithm is applied to extract spheres from scalar fields in cubic, tetrahedral, and hybrid grids. More intricate applications are then explored by re–extracting models from the Stanford 3D Scanning Repository imported within computational meshes. This tests the algorithm's precision and its capability to handle detailed geometrical data. The chapter concludes with industrial applications, focusing on the extraction of TopO designs of heat exchangers.

5.1 Spheres

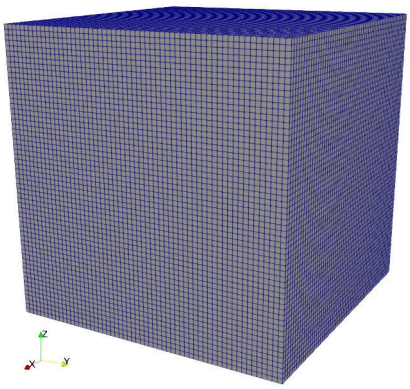
The first application of the proposed algorithm is performed on a scalar field decreasing outwards proportional to the distance (radius) from the center, to extract the shape of a sphere. This is performed in 3 types of grids,

- a cubic structured grid of 60x60x60 nodes
- a tetrahedral grid of 174 nodes containing 507 elements
- a hybrid grid of 1340 nodes containing 3740 elements (225 hexahedra, 540 prisms, 45 pyramids, 2930 tetrahedra.)

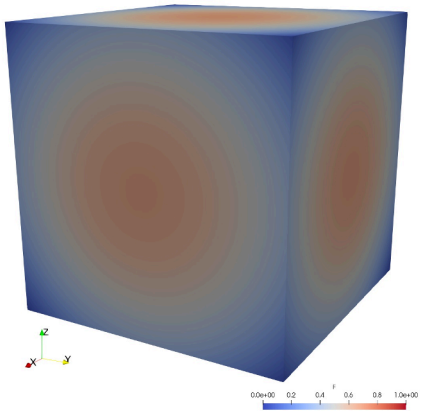
Following that, more spheres are added to the computational domain to investigate the interference of the surfaces.

5.1.1 Sphere on cubic grid

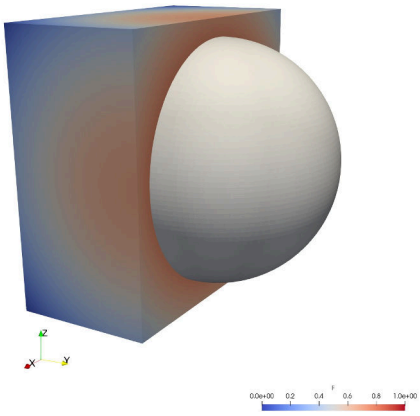
Herewith, the use case that regards a computational grid of $60 \times 60 \times 60$ nodes containing hexahedral elements with a field β applied to it, is treated. Figure 5.1 illustrates the extraction of the sphere's surface.



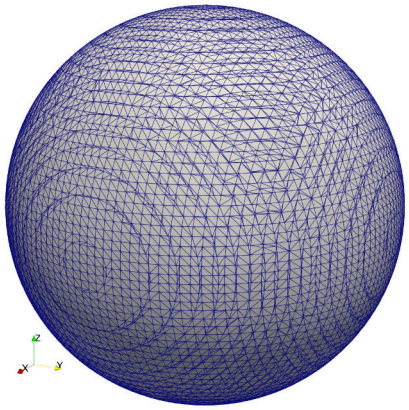
(a) Computational domain of cubic elements (60x60x60 nodes)



(b) Spherical β field



(c) Section of the field with isosurface $\beta = 0.45$



(d) Surface mesh of isosurface $\beta =$ 0.45 comprised of 12432 nodes and 24860 triangular elements

Figure 5.1: *Application of the algorithm on a cubic computational grid (60x60x60) with a field β applied to it. (a) The cubic computational grid and (b) its β field, (c) Section of the computational grid with the extraction of the* 0.45 *isosurface, (d) Surface mesh of triangular elements (STL file output) of the* 0.45 *isosurface comprised of 12432 nodes and 24860 triangular elements.*

Figure 5.2 provides a slice view of the computational domain with a comparison of the TtoST radius with the nominal. It is evident that the application of the method in the high resolution background grid yields a near perfect representation of the sphere with little deviation of its radius from the nominal (Table 5.1). Isolines of the nominal radius along with the resulting TtoST sphere are plotted in Fig. 5.3.

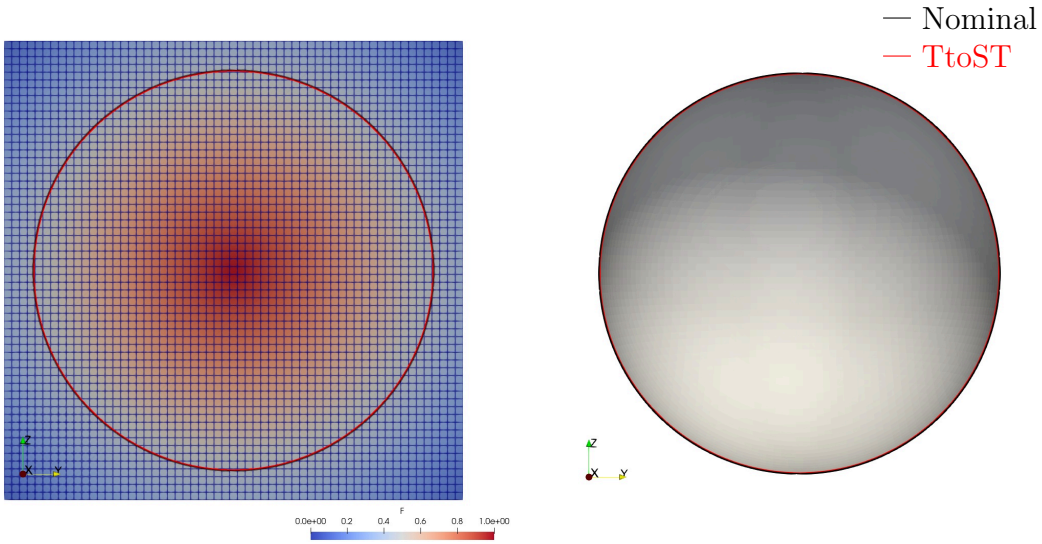


Figure 5.2: *Slice view of the nominal sphere radius and the extracted surface mesh from TtoST.*

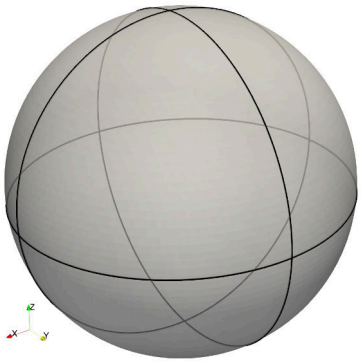


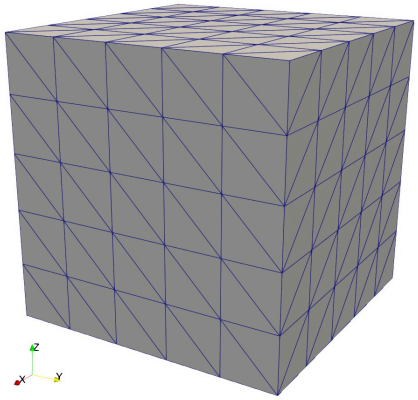
Figure 5.3: *Isolines of the nominal radius along with the extracted surface mesh from TtoST.*

Table 5.1: *Deviation of the TtoST radius from the nominal value*

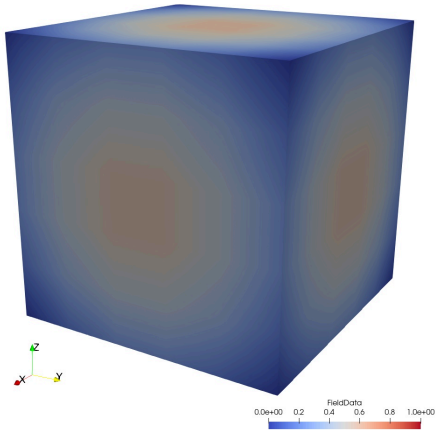
Deviation	TtoST - Nominal
Min	-8.05454×10^{-5}
Max	-1.44176×10^{-9}
Mean	-2.72957×10^{-5}
σ	2.17697×10^{-5}

5.1.2 Sphere on tetrahedral grid

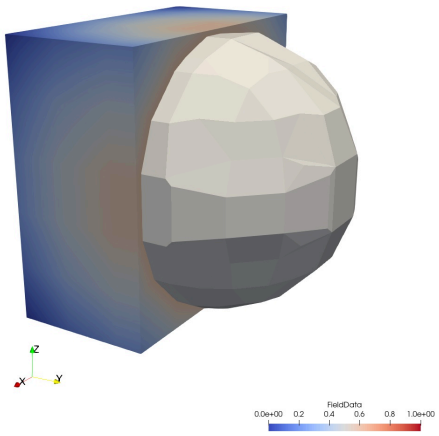
Herewith, the use case that regards a computational grid of tetrahedral elements, comprising of 174 nodes and 507 elements, with a field β applied to it, is treated. Figure 5.4 illustrates the extraction of the sphere's surface.



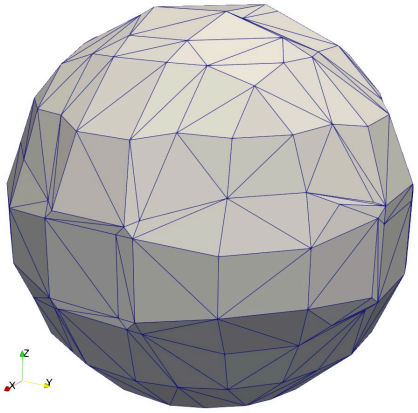
(a) Computational domain of 507 tetrahedral elements



(b) Spherical β field



(c) Section of the field with isosurface $\beta = 0.6$



(b) Isosurface 0.6 *depicting a Sphere*

Figure 5.4: *Application of the algorithm on a tetrahedral computational grid comprising of 174 nodes and 507 elements with a field β applied to it. (a) The hybrid computational grid and (b) its β field, (c) Section of the computational grid with the extraction of the* 0.6 *isosurface, (d) Surface mesh of triangular elements (STL file output) of the* 0.6 *isosurface comprised of 210 nodes and 416 triangular elements. Low resolution of the initial domain results in low quality of the resulting sphere.*

Figure 5.5 provides a slice view of the computational domain with a comparison of the TtoST radius with the nominal. The application of the method in this poor resolution background grid yields a low quality representation of the sphere with some deviation of its radius from the nominal (Table 5.2). Isolines of the nominal radius along with the resulting TtoST sphere are plotted in Fig. 5.6.

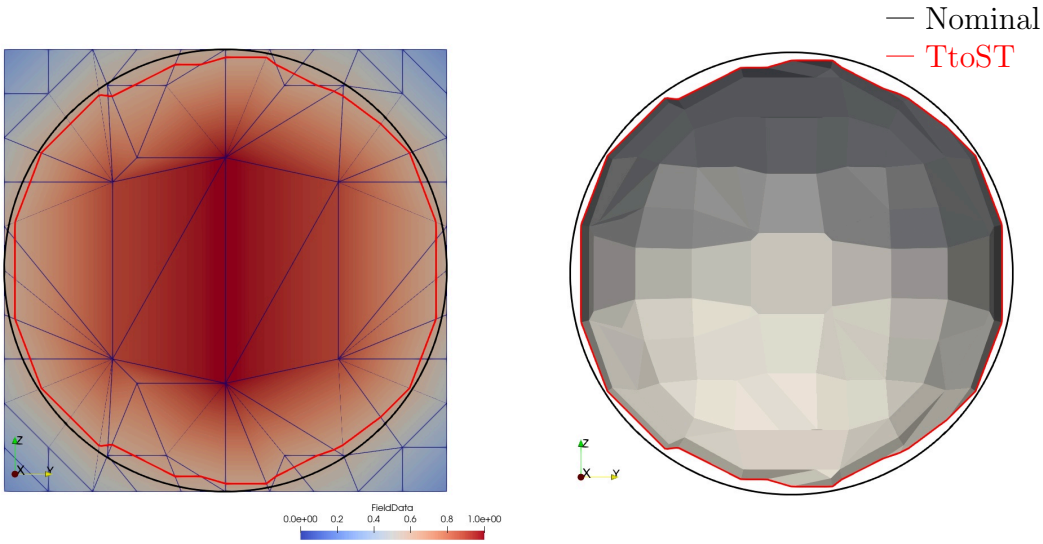


Figure 5.5: *Slice view of the nominal sphere radius and the extracted surface mesh from TtoST.*

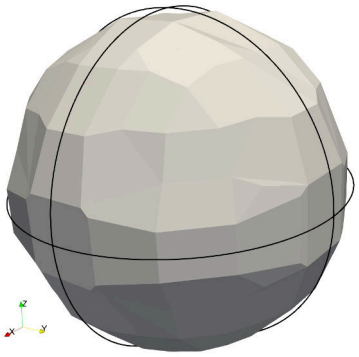


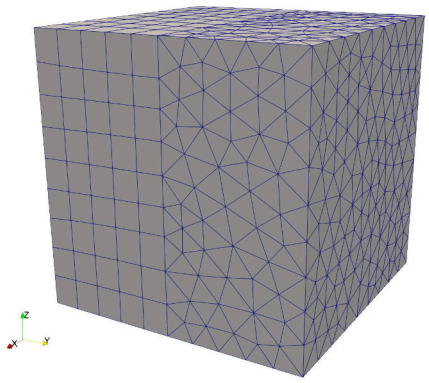
Figure 5.6: *Isolines of the nominal radius along with the extracted surface mesh from TtoST.*

Table 5.2: *Deviation of the TtoST radius from the nominal value*

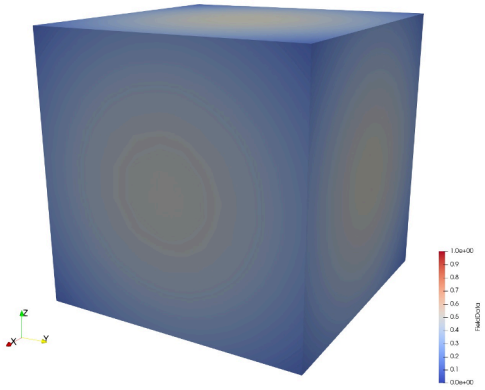
Deviation	TtoST - Nominal
Min	-2.9259×10^{-2}
Max	-6.8738×10^{-4}
Mean	-9.3212×10^{-3}
σ	6.6561×10^{-3}

5.1.3 Sphere on hybrid grid

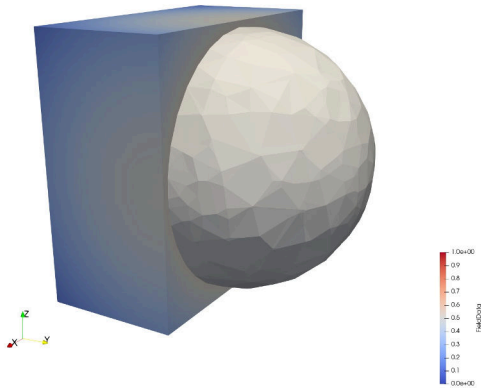
Herewith, the use case that regards a hybrid computational grid comprising of 1340 nodes and 3740 elements (225 hexahedra, 540 prisms, 45 pyramids, 2930 tetrahedra) with a field β applied to it, is treated. Figure 5.7 illustrates the extraction of the sphere's surface.



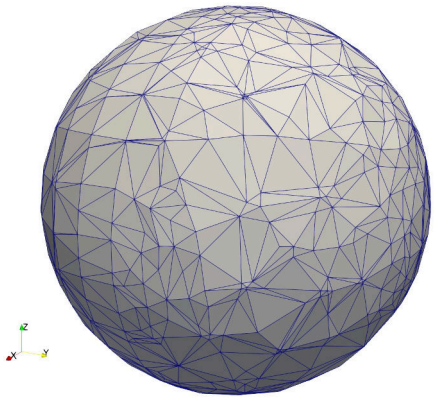
(a) Computational domain of 3740 elements



(b) Spherical β field



(c) Section of the field with isosurface $\beta = 0.52$



(d) Isosurface 0.52 depicting a Sphere

Figure 5.7: *Application of the algorithm on a hybrid computational grid comprising of 1340 nodes and 3740 elements (225 hexahedra, 540 prisms, 45 pyramids, 2930 tetrahedra) with a field β applied to it. (a) The hybrid computational grid and (b) its β field, (c) Section of the computational grid with the extraction of the 0.52 isosurface, (d) Surface mesh of triangular elements (STL file output) of the 0.52 isosurface comprised of 848 nodes and 1692 triangular elements.*

Figure 5.8 provides a slice view of the computational domain with a comparison of the TtoST radius with the nominal. The application of the method in this medium resolution background grid yields a good quality representation of the sphere with slight deviation of its radius from the nominal (Table 5.2). Isolines of the nominal radius along with the resulting TtoST sphere are plotted in Fig. 5.9.

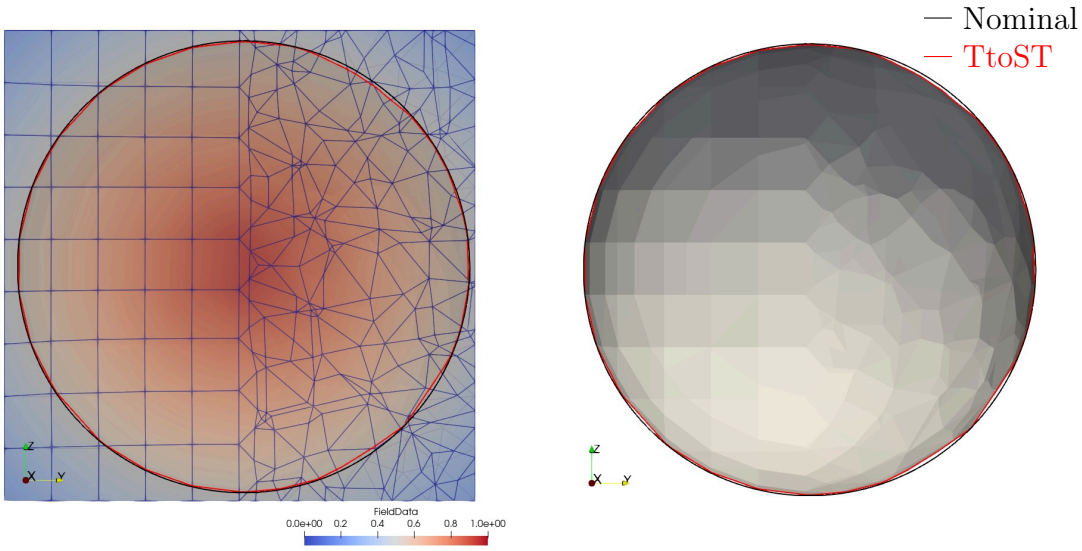


Figure 5.8: *Slice view of the nominal sphere radius and the extracted surface mesh from TtoST.*

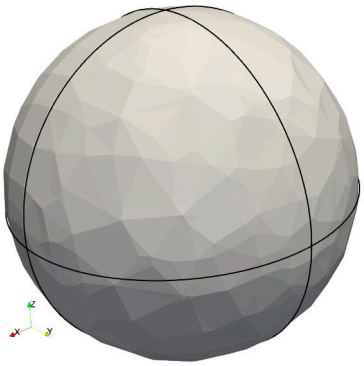


Figure 5.9: *Isolines of the nominal radius along with the extracted surface mesh from TtoST.*

Table 5.3: *Deviation of the TtoST radius from the nominal value*

Deviation	TtoST - Nominal
Min	-8.2492×10^{-3}
Max	-4.7228×10^{-7}
Mean	-1.3942×10^{-3}
σ	1.3625×10^{-3}

5.1.4 Two spheres on cubic grid

The same cubic computational grid (60x60x60 nodes) used in Section 5.1.1 is incorporated. A β field is introduced similar to the one used before, although this time for two distinct sources, depicting two spheres. Figure 5.10 presents the application of the algorithm for the extraction of two spheres.

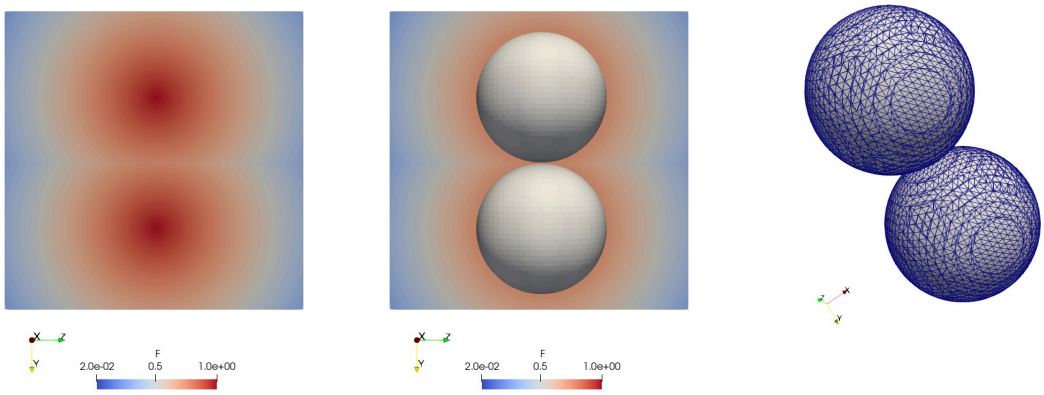


Figure 5.10: *Application of the algorithm for the extraction of two spheres. (Left) Section of the β field, (Middle) Section with isosurface $\beta = 0.725$, (Right) Surface mesh of isosurface $\beta = 0.725$.*

The scope is to demonstrate how the algorithm handles surfaces based on their relative positions within the grid's cells. If the surfaces of the spheres are separated in different cells, the algorithm can differentiate between the two surfaces, generating two distinct isosurfaces. If the surfaces of the spheres are separated within the same grid cell, the algorithm fails to distinguish them, since a grid cell's interior is not further subdivided. As a result, it generates a single connected isosurface across both spheres, interpreting them as a single entity, illustrated in Fig. 5.11.

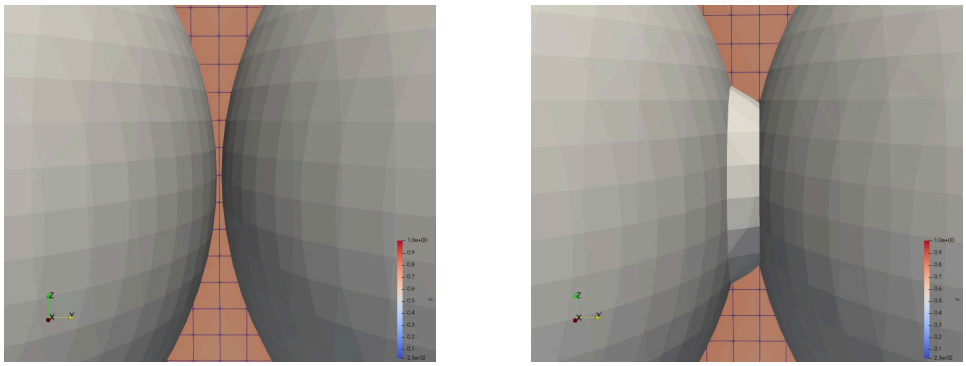
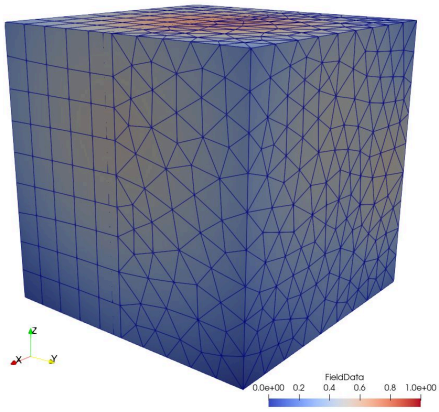


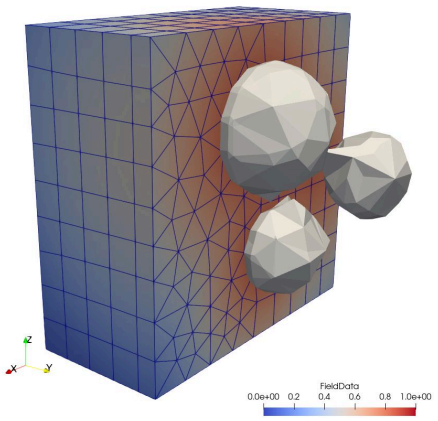
Figure 5.11: *(Left) Separation of the spheres when the isosurface intersects different cells of the grid. (Right) Merging of the spheres when isosurface intersects twice the same cells. Spheres equally spaced apart only their position in the grid is changed.*

5.1.5 Four unequal spheres on hybrid grid

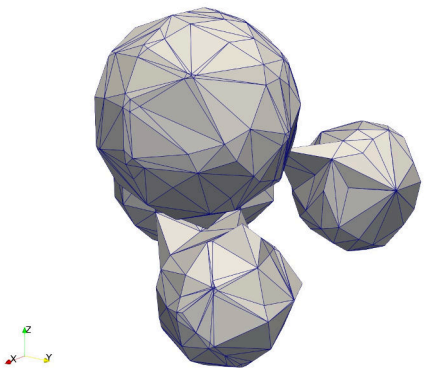
The same hybrid computational grid used in Section 5.1.3 is incorporated. A β field is introduced, although this time depicting four spheres. Figure 5.12 presents the application of the algorithm for the extraction of four spheres. In this example the low resolution of the background grid along with the relative size of the spheres and the size of the elements of the grid, results in merging of the surfaces of the spheres.



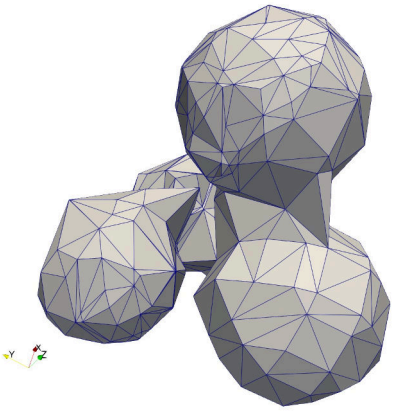
(a) Computational domain with β field of four spheres



(b) Section of the β field with isosurface $\beta = 0.8$ depicting four spheres



(c) View 1 of four spheres extracted from isosurface $\beta = 0.8$



(b) View 2 of four spheres extracted from isosurface $\beta = 0.8$

Figure 5.12: *Application of the algorithm on a hybrid computational grid with a four spheres field β applied to it. (a) The hybrid computational grid with its β field, (b) Section of the computational grid with the extraction of the* 0.8 *isosurface, (c,d) Two views of the surface mesh of triangular elements (STL file output) of the* 0.8 *isosurface depicting four spheres comprised of 401 nodes and 794 triangular elements. Due to the low resolution of the grid, the 4 spheres are interfering within common elements of the grid, leading to their merging from the algorithm.*

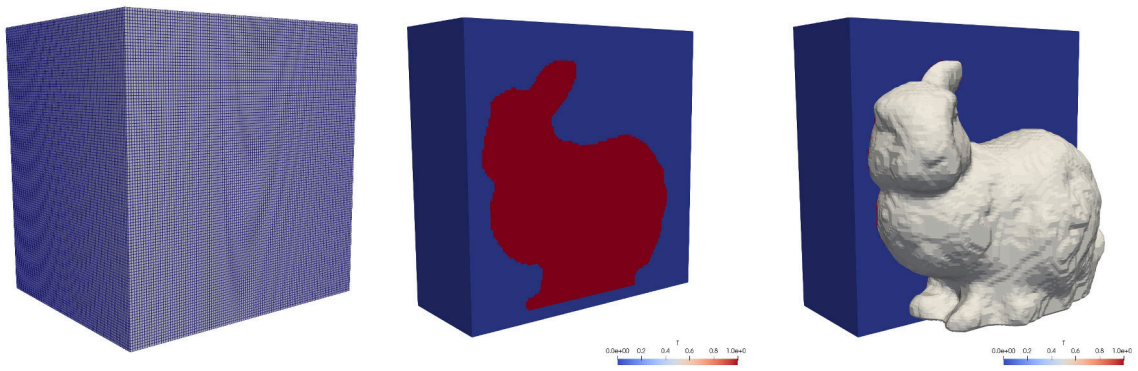
5.2 Stanford 3D Scanning Repository's Models

Two models from the Stanford 3D Scanning Repository [26] are imported into a computational domain with a cell-centered porosity field, β. The field is initially set to $\beta = 0$ throughout the domain. Then employing the *searchableSurface* tool, a feature of OpenFOAM©that allows the embedding of surfaces into the computational mesh, the cells within the mesh that are inside the geometry's surface are marked, adjusting their value to $\beta = 1$. The proposed algorithm is used to re–extract the the surface.

5.2.1 The 'Stanford Bunny'

The "Stanford Bunny" is a well–known 3D model, seen in Fig. 5.13, from the Stanford 3D Scanning Repository. This model is suitable to examine the algorithm's ability to capture the intricate details of the surface. The model is imported in a computational grid containing $115 \times 95 \times 105 = 1.256.375$ elements in the (x,y,z) directions. The computational domain, section of the β field and views of the resulting output geometry are shown in Fig. 5.14.

Figure 5.13: *The 'Stanford Bunny' (from [26]).*



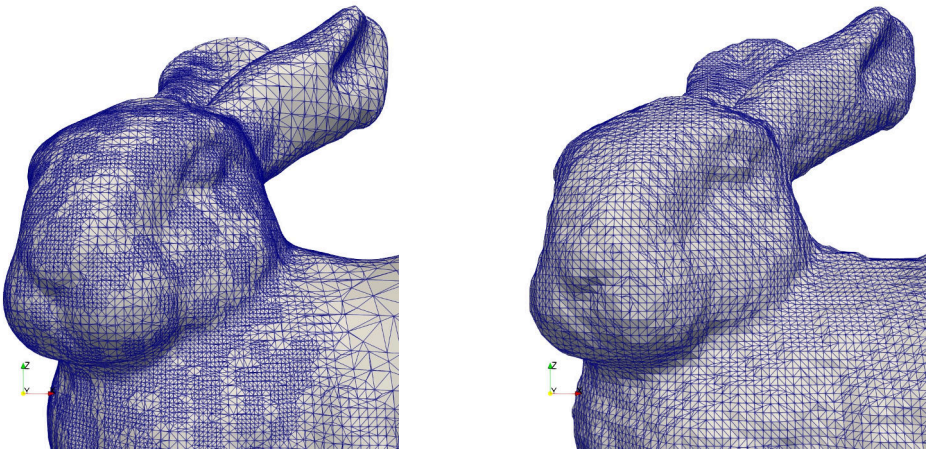
(a) *(b)* *(c)*

Figure 5.14: *(a) Computational domain, (b) section of the cell-centered β field with blue cells outside the surface ($\beta = 0$) and red cells inside the surface ($\beta = 1$). (c) Section of the β field with isosurface $\beta = 0.5$ extracted depicting the bunny model.*

(a) Front left *(b) Front right* *(c) Back left*

Figure 5.14: *Views of the extracted isosurface $\beta = 0.5$, depicting the bunny model. (a) Front left, (b) Front right and (c) back left views.*

Figure 5.15 provides a comparative detail view of the original (a) and the extracted (b) surface mesh associated with the bunny model. The original surface mesh, as it is imported into the computational domain, is characterized by localized refinement, which enriches the detail in more intricate regions of the model. The extracted mesh is characterized by a consistent resolution throughout, dependent on the grid's resolution. While this results in a less refined (pixelated) representation, particularly in areas that would benefit from higher resolution, the extracted mesh is nevertheless deemed acceptable. It should be noted that some detail loss is inherent, not only due to the mesh extraction process but also from the initial import into the computational domain and the inherent resolution of that domain.



(a) Original surface mesh *(b) Extracted surface mesh*

Figure 5.15: *Detail view of the (a) original surface mesh imported in the computational domain and (b) the extracted surface mesh from the field, readily available as the output of our algorithm. The extracted surface mesh assumes a uniform resolution and doesn't have refined regions.*

In Fig. 5.16 a slice view of the bunny model is presented for the comparison of the original surface mesh and the TtoST resulting geometry.

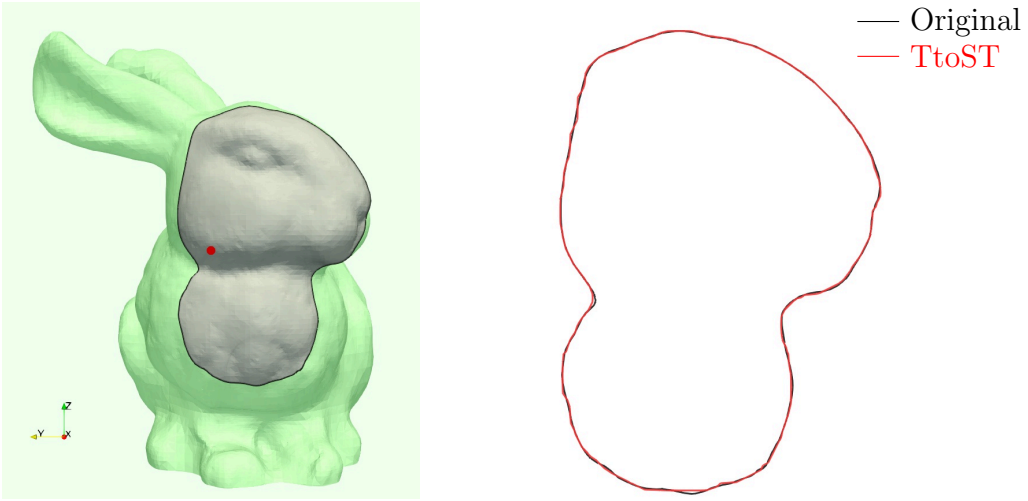


Figure 5.16: *(a) Slice view of the original surface mesh and (b) comparison of the slice outlines from the original and the extracted surface mesh from TtoST.*

5.2.2 Dragon

In the next example a dragon model, seen in Fig. 5.17, from the Stanford 3D Scanning repository is used. The model is imported in a computational grid containing $296 \times 140 \times 202 = 8.370.880$ elements in the (x,y,z) directions. The computational domain, section of the β field and views of the resulting output geometry are show in Fig. 5.18.

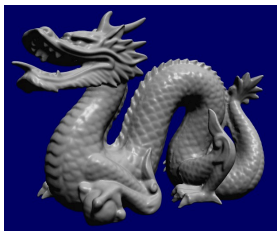
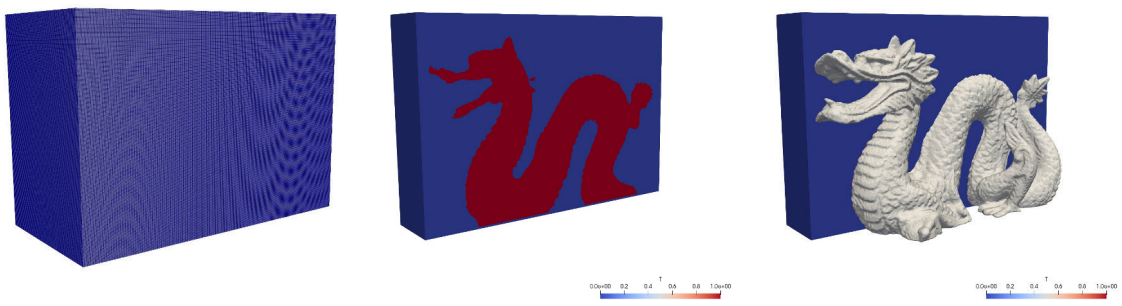


Figure 5.17: *Dragon model (from [26])*

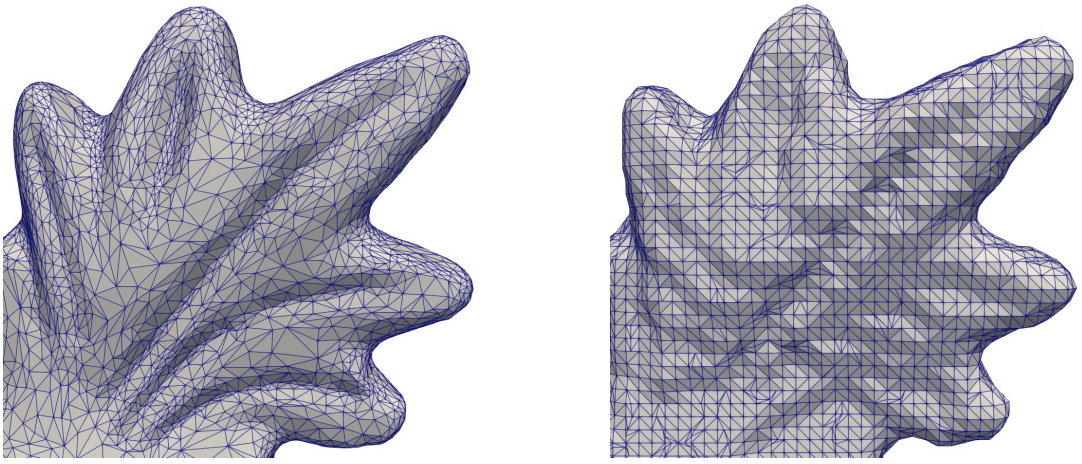


(a) *(b)* *(c)*

(d) *(e)* *(f)*

Figure 5.18: *(a) Computational domain, (b) Section of the β field, (c) Section of the β field with isosurface $\beta = 0.5$ depicting the dragon. (d) Back left, (b) Front right, (c) front left views.*

Similar to the bunny model (Section 5.2.1), Figure 5.19 provides a comparative detail view of the tail, of the original (a) and the extracted (b) surface mesh associated with the dragon model. The original surface mesh, as it is imported into the computational domain, is characterized by localized refinement, adapting to the details of the geometry. The extracted mesh is characterized by a consistent resolution throughout, dependent on the grid's resolution.



(a) Original surface mesh *(b) Extracted surface mesh*

Figure 5.19: *Detail view of the tail of (a) the original surface mesh imported in the computational domain and (b) the extracted surface mesh from the field, for the dragon model. It is evident that the original surface mesh is fine tuned to the characteristics of the surface. The extracted surface mesh, readily available as the output of our algorithm, assumes a uniform resolution throughout the mesh and doesn't have refined regions, still retaining most of the details of the model.*

In Fig. 5.20 a slice view of the bunny model is presented for the comparison of the original surface mesh and the TtoST resulting geometry.

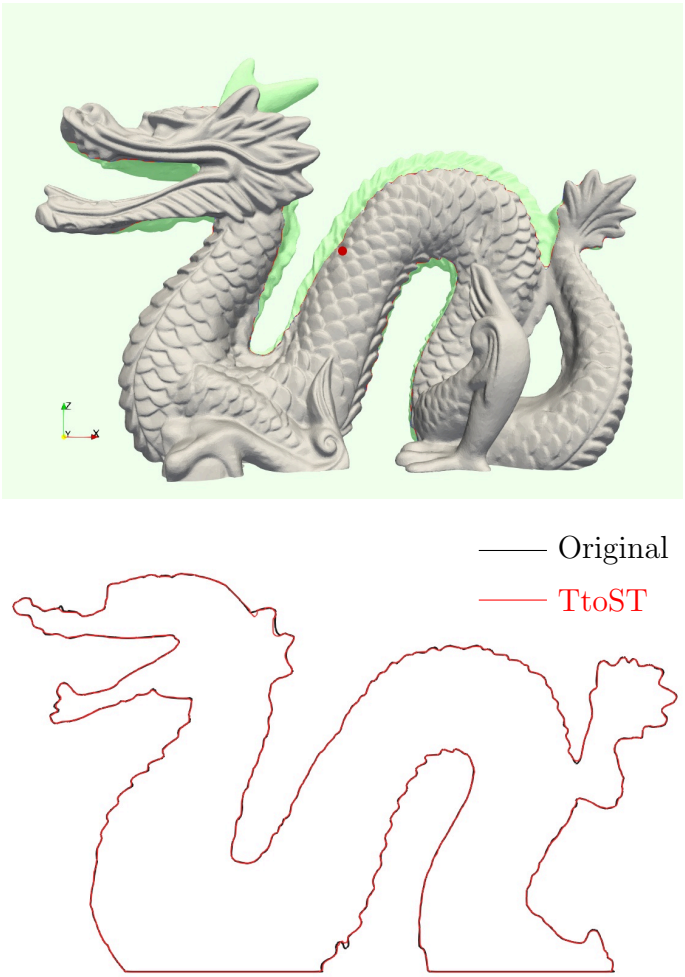


Figure 5.20: *(a) Slice view of the original surface mesh and (b) comparison of the slice outlines from the original and the extracted surface mesh from TtoST.*

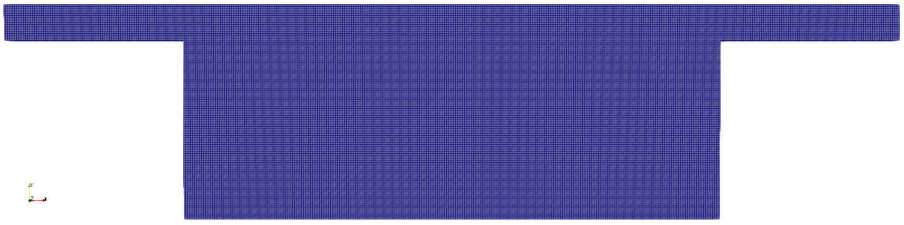
5.3 Industrial Applications – Heat Exchangers

The next applications regard the extraction of fluid paths produced from the TopO design of heat exchangers. Three applications are presented, which are

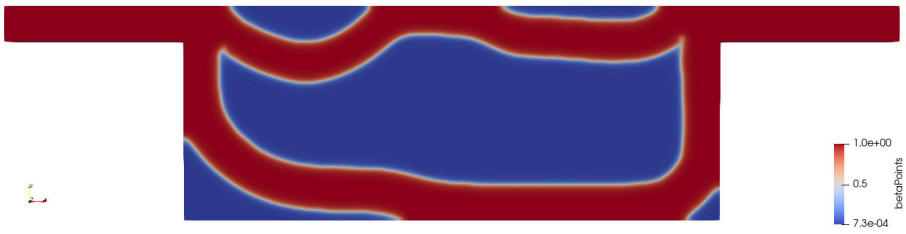
- a 2D heat exchanger of 1 inlet and 1 outlet
- a Bi–fluid 3D heat exchanger of 1 inlet and 1 outlet
- a Bi–fluid 3D heat exchanger of 1 inlet and 8 outlets

5.3.1 2D Heat Exchanger 1 Inlet 1 Outlet

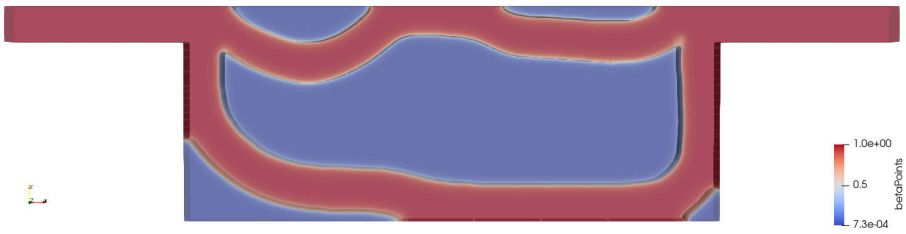
In Fig. 5.21 the TtoST process is illustrated for the case of a 2D heat exchanger of 1 inlet and 1 outlet. The heat exchanger is designed with a symmetry condition on the top boundary.



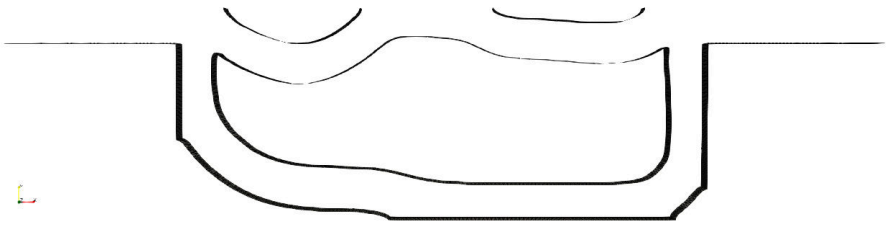
(a) Computational domain



(b) TopO porosity–β field result, fluid ($\beta = 1$) in red, solid in blue ($\beta = 0$)



(c) Isosurface $\beta = 0.5$

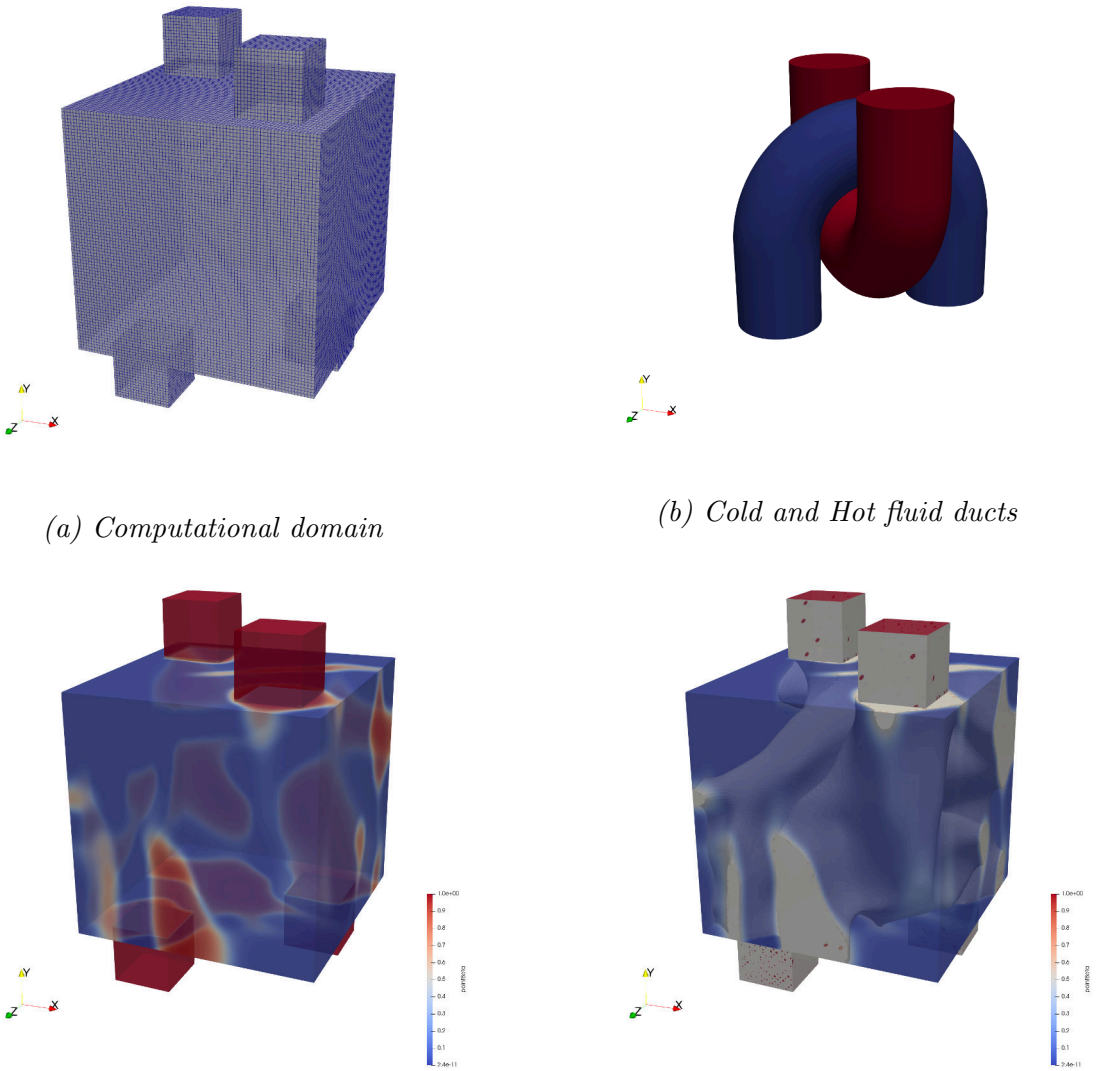


(d) Fluid–Solid Interface

Figure 5.21: *TopO of a 2D Heat Exchanger of 1 Inlet and 1 outlet. (a) Computational domain, (b) the β field computed from porosity–based TopO, (c) the Isosurface $\beta = 0.5$ under consideration and (d) the extracted FSI geometry using the proposed algorithm.*

5.3.2 Bi-fluid 3D Heat Exchanger 1 Inlet 1 Outlet

Application of the algorithm in the case of a Bi-fluid 3D Heat Exchanger of 1 inlet 1 outlet for each fluid. In Fig. 5.22 the TtoST process is illustrated and in Fig. 5.23 different views of the FSI depicting the ducts of the two fluids are presented.



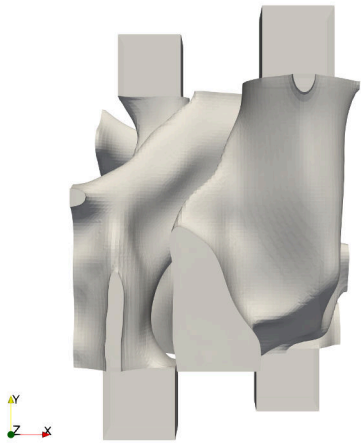
(a) Computational domain

(b) Cold and Hot fluid ducts

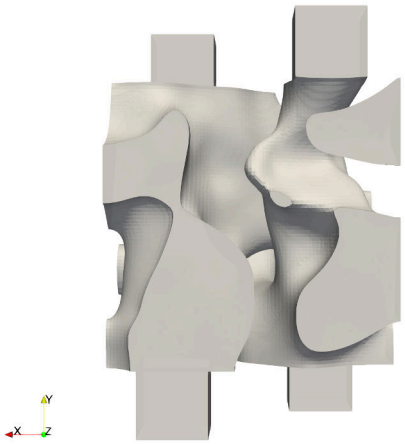
(c) β field

(d) Fluid–Solid Interface

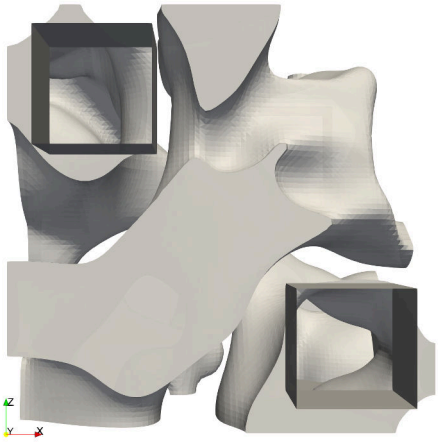
Figure 5.22: *TopO of a Bi–Fluid 3D Heat Exchanger of 1 inlet and 1 outlet for each fluid. (a) Computational domain and (b) initial geometry of cold (blue) and hot (red) fluid ducts. (c) The β field computed from porosity–based TopO and (d) the extracted FSI geometry using the proposed algorithm.*



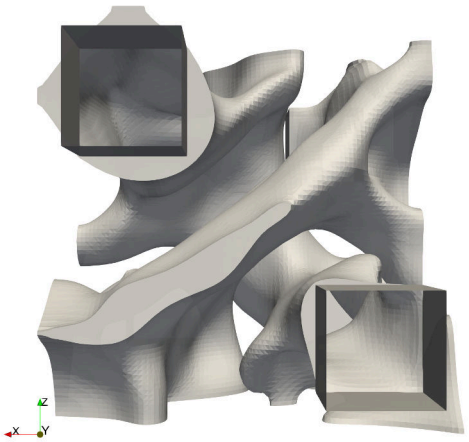
(a) Top view



(b) Bottom view



(c) Left view

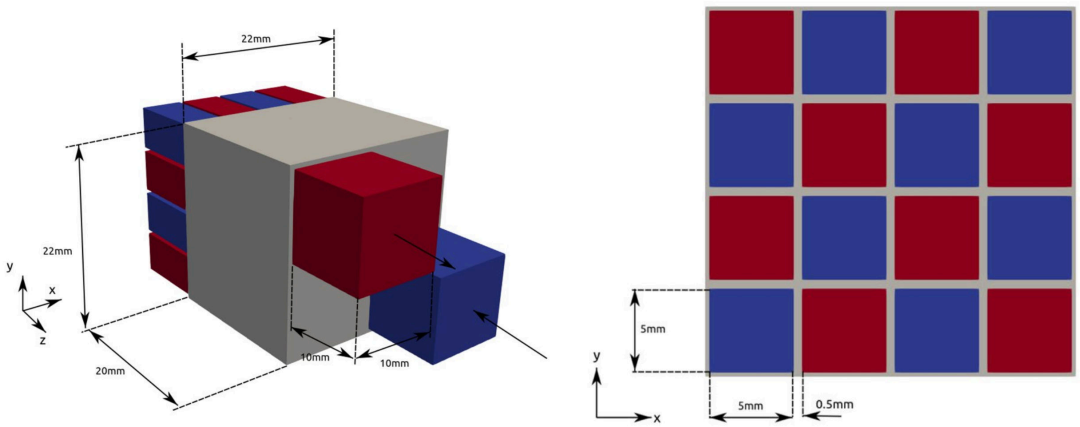


(d) Right view

Figure 5.23: *Views of the extracted FSI of the ducts of the two fluids for the heat exchanger. (a) Top, (b) bottom, (c) left, (d) right view.*

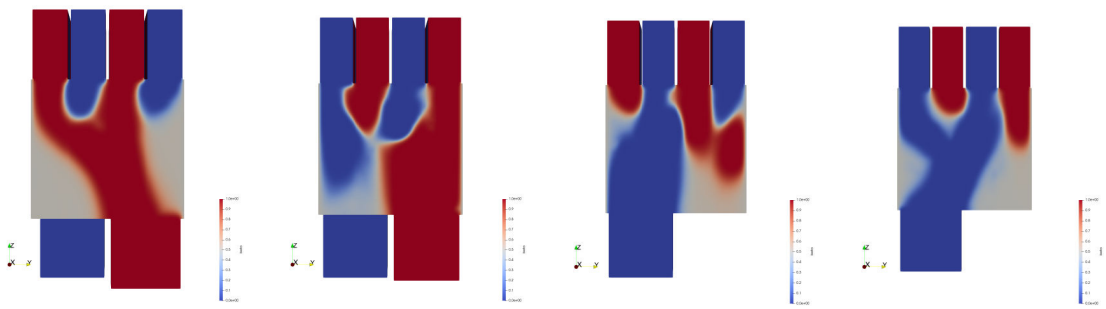
5.3.3 Bi-fluid 3D Heat Exchanger 1 Inlet 8 Outlets

This industrial application is derived from the work by Galanos et al. [5], where a Bi-fluid 3D Heat Exchanger of 1 Inlet and 8 Outlets for each fluid, is designed synergistically utilizing TopO and ShpO. The β field produced by TopO is provided by the authors and used in this thesis to perform the TtoST generating the shape representations of the two manifolds of the cold and hot fluids. Figure 5.24 depicts the initial sizing sizing of the design domain, the inlets for each fluid the outlets for each fluid. In Fig. 5.25 sections views of the resulting porosity β field are shown in sample iso–y planes.



(b) *(c)*

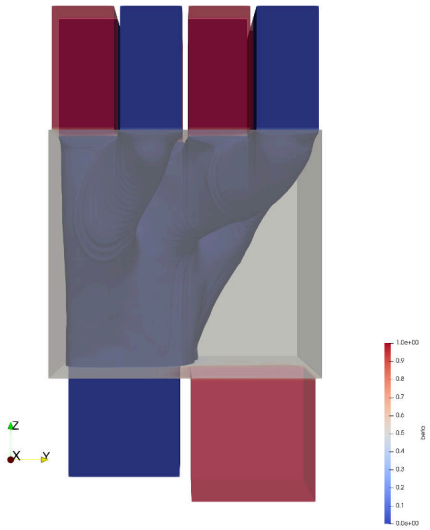
Figure 5.24: *Initial sizing of (a) the design domain, the inlet for each fluid, hot (red) and cold (blue), and (b) the 8 outlets for each fluid.*



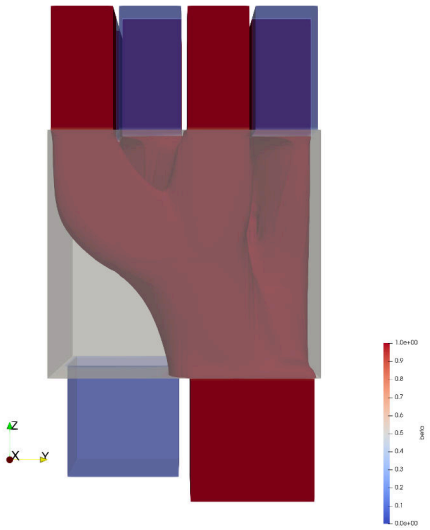
(a) Section view at $y = 20mm$ *(b) Section view at $y = 16.5mm$* *(c) Section view at $y = 11mm$* *(d) Section view at $y = 5.5mm$*

Figure 5.25: *Section views of the β field at different iso–y planes representing the sought fluid paths.*

In Fig. 5.26 the result from the TtoST process using the proposed algorithm is presented. The two FSIs are extracted for the cold fluid manifold ($\beta = 0.25$) and for the hot fluid manifold ($\beta = 0.75$) from the porosity field. Figure 5.27 depicts combined views of the extracted manifolds.



(a) Isosurface $\beta = 0.25$ (cold fluid)

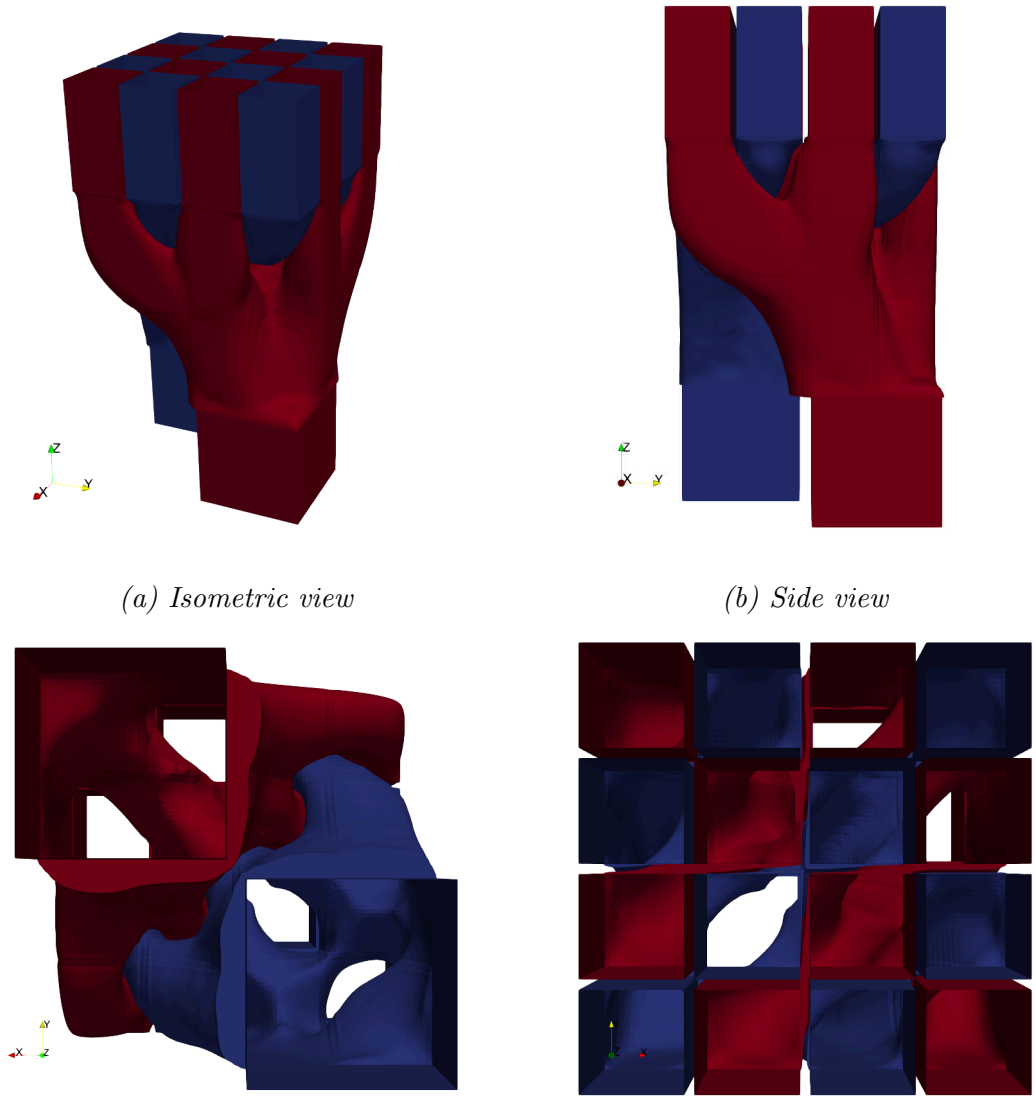


(b) Isosurface $\beta = 0.75$ (hot fluid)

(c) FSI of the cold fluid manifold

(d) FSI of the hot fluid manifold

Figure 5.26: *TtoST from the porosity field β for the two isosurfaces of the cold (blue, $\beta = 0.25$) and the hot (red, $\beta = 0.75$) fluid manifolds.*



(a) Isometric view *(b) Side view*

(c) Front view *(d) Back view*

Figure 5.27: *Combined views of the generated FSI for the cold (blue) and the hot (red) fluid manifolds. (a) Isometric, (b) Side, (c) Front, (d) Back view.*

Chapter 6

Summary - Conclusions

6.1 Overview

This diploma thesis programs a software tool for post-processing the outcome of Topology Optimization (TopO) runs in fluid mechanics, focusing on the transition from the resulting porosity fields to shape representations. This transition is important in order to finalize the design under consideration for manufacturing purposes or for overcoming inaccuracies of the standard porosity–based TopO, by enabling the imposition of explicit boundary conditions on the sought FSI. The performance of the design can then be evaluated with high–fidelity CFD solvers running on body-fitted grids, and a linking with ShpO is possible to further refine the design. The core of this work involved the detailed study of the Marching Cubes algorithm, an established framework originally developed for the visualization of medical data. A methodological innovation is introduced, expanding the idea of the algorithm to handle both structured and unstructured/hybrid 2D and 3D grids. This adaptation led to the development of 'generalized Marching Algorithms', enabling their application in a variety of computational grids and real industrial applications.

The software is implemented in C++ and is now compatible with the CFD–based Optimization (OpenFOAM©and in–house) software of the Parallel CFD & Optimization Unit of NTUA (PCopt/NTUA). The implementation involved the adaptation of the Marching Cubes triangulation look–up tables from the literature and the construction of the appropriate Data Structure for the case of cubic structured grids. Following that, the triangulation look–up tables for the other element types (hexahedra, prisms, pyramids, tetrahedra) and the boundary are formulated, and the adaptation of the Data Structure for the revised algorithm for unstructured and hybrid grids is developed.

The software is showcased in a number of applications ranging from simple geometric forms, such as spheres, to complex industrial designs for the case of heat exchangers. First the proposed tool was tested on cubic, tetrahedral, and hybrid sample grids to extract the shape of spheres. This demonstrated the tool's capacity to handle different grid types and is was evident that the fidelity of the extracted shapes was influenced by the grid's resolution, with a notable difference in detail and accuracy as the resolution of the background grid increases. Further tests involved extracting multiple spheres to assess the algorithm's capability to distinguish between close or overlapping geometries. It was found that the algorithm could differentiate between separate entities if they occupied distinct cells, but separate geometries within the same cells were interpreted as a single entity. The tool's precision and its ability to manage detailed geometrical data were further tested by re-extracting models from the Stanford 3D Scanning Repository imported within computational grids. The 'Stanford Bunny' and a dragon model were used to highlight the tool's capability to capture intricate details of the model. Finally, in the context of industrial application, the tool showcased the ability to extract the sought FSI from the porosity fields produced by TopO. The boundary was successfully incorporated in the resulting surface mesh.

6.2 Future Work

Based on the implementation of the proposed TtoST method for hybrid grids, the following future work guidelines are proposed:

- Refinement of the algorithm to address internal ambiguities more effectively, at a low cost for efficient processing of large–scale industrial applications.
- Automation of the transition process from TopO to ShpO within a unified framework, streamlining the design process for complex systems. This would include:
  - Integration of the algorithm in the TopO workflow to extract the resulting FSI as a final step of the TopO run
  - Employment of a body–fitted grid generator on the resulting FSI to run CFD analysis and obtain high–fidelity (ground truth) value of the objective function.
  - Importing of the extracted FSI in the ShpO workflow for further refinement.
- Streamlining the manufacturing of the results from TopO, by selective appropriate manufacturing technique.

Appendix A

Marching Algorithms

Triangulation Tables

In this section the precompiled look–up tables for the triangulation of elements are listed. The calculated index (corresponding nodes 0 for outside, 1 for inside the isosurface) for each element corresponds to the generation of triangles defined by the interpolated nodes on the local edges of the element. The notation "(e1–e2–e3)" indicates a triangle generated with nodes interpolated on the local edges e1, e2, and e3 of an element. The order of the nodes on these edges is significant and is related to ensuring the correct orientation of the triangle's normal, a vector perpendicular to the triangle's surface and pointing outwards.

A.1 Hexahedra

Triangulation table for the hexahedra adapted from [23].

Local Nodes	7	6	5	4	3	2	1	0
$Index_2$								

Figure A.1: *8–bit Hexahedron index (left) and local edge and node numbering (right).*

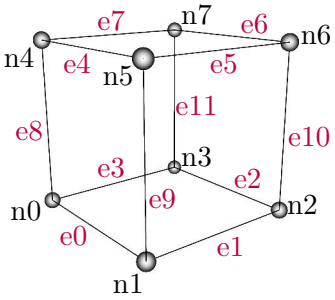


Table A.1: *Marching Hexahedra Triangulation Table*

Hexahedron Index	Triangles Edge Connectivity
00000000	-
00000001	(0-3-8)
00000010	(0-9-1)
00000011	(1-3-8) (9-1-8)
00000100	(1-10-2)
00000101	(0-3-8) (1-10-2)
00000110	(9-10-2) (0-9-2)
00000111	(2-3-8) (2-8-10) (10-8-9)
00001000	(3-2-11)
00001001	(0-2-11) (8-0-11)
00001010	(1-0-9) (2-11-3)
00001011	(1-2-11) (1-11-9) (9-11-8)
00001100	(3-1-10) (11-3-10)
00001101	(0-1-10) (0-10-8) (8-10-11)
00001110	(3-0-9) (3-9-11) (11-9-10)
00001111	(9-10-8) (10-11-8)
00010000	(4-8-7)
00010001	(4-0-3) (7-4-3)
00010010	(0-9-1) (8-7-4)
00010011	(4-9-1) (4-1-7) (7-1-3)
00010100	(1-10-2) (8-7-4)
00010101	(3-7-4) (3-4-0) (1-10-2)
00010110	(9-10-2) (9-2-0) (8-7-4)
00010111	(2-9-10) (2-7-9) (2-3-7) (7-4-9)
00011000	(8-7-4) (3-2-11)
00011001	(11-7-4) (11-4-2) (2-4-0)
00011010	(9-1-0) (8-7-4) (2-11-3)

Continued on next page

Table A.1 – continued from previous page

Hexahedron Index	Triangles Edge Connectivity
00011011	(4-11-7) (9-11-4) (9-2-11) (9-1-2)
00011100	(3-1-10) (3-10-11) (7-4-8)
00011101	(1-10-11) (1-11-4) (1-4-0) (7-4-11)
00011110	(4-8-7) (9-11-0) (9-10-11) (11-3-0)
00011111	(4-11-7) (4-9-11) (9-10-11)
00100000	(9-4-5)
00100001	(9-4-5) (0-3-8)
00100010	(0-4-5) (1-0-5)
00100011	(8-4-5) (8-5-3) (3-5-1)
00100100	(1-10-2) (9-4-5)
00100101	(3-8-0) (1-10-2) (4-5-9)
00100110	(5-10-2) (5-2-4) (4-2-0)
00100111	(2-5-10) (3-5-2) (3-4-5) (3-8-4)
00101000	(9-4-5) (2-11-3)
00101001	(0-2-11) (0-11-8) (4-5-9)
00101010	(0-4-5) (0-5-1) (2-11-3)
00101011	(2-5-1) (2-8-5) (2-11-8) (4-5-8)
00101100	(10-11-3) (10-3-1) (9-4-5)
00101101	(4-5-9) (0-1-8) (8-1-10) (8-10-11)
00101110	(5-0-4) (5-11-0) (5-10-11) (11-3-0)
00101111	(5-8-4) (5-10-8) (10-11-8)
00110000	(9-8-7) (5-9-7)
00110001	(9-0-3) (9-3-5) (5-3-7)
00110010	(0-8-7) (0-7-1) (1-7-5)
00110011	(1-3-5) (3-7-5)
00110100	(9-8-7) (9-7-5) (10-2-1)
00110101	(10-2-1) (9-0-5) (5-0-3) (5-3-7)
00110110	(8-2-0) (8-5-2) (8-7-5) (10-2-5)
00110111	(2-5-10) (2-3-5) (3-7-5)
00111000	(7-5-9) (7-9-8) (3-2-11)
00111001	(9-7-5) (9-2-7) (9-0-2) (2-11-7)
00111010	(2-11-3) (0-8-1) (1-8-7) (1-7-5)
00111011	(11-1-2) (11-7-1) (7-5-1)
00111100	(9-8-5) (8-7-5) (10-3-1) (10-11-3)
00111101	(5-0-7) (5-9-0) (7-0-11) (1-10-0) (11-0-10)
00111110	(11-0-10) (11-3-0) (10-0-5) (8-7-0) (5-0-7)
00111111	(11-5-10) (7-5-11)
01000000	(10-5-6)
01000001	(0-3-8) (5-6-10)
01000010	(9-1-0) (5-6-10)
01000011	(1-3-8) (1-8-9) (5-6-10)

Continued on next page

Table A.1 – continued from previous page

Hexahedron Index	Triangles Edge Connectivity
01000100	(1-5-6) (2-1-6)
01000101	(1-5-6) (1-6-2) (3-8-0)
01000110	(9-5-6) (9-6-0) (0-6-2)
01000111	(5-8-9) (5-2-8) (5-6-2) (3-8-2)
01001000	(2-11-3) (10-5-6)
01001001	(11-8-0) (11-0-2) (10-5-6)
01001010	(0-9-1) (2-11-3) (5-6-10)
01001011	(5-6-10) (1-2-9) (9-2-11) (9-11-8)
01001100	(6-11-3) (6-3-5) (5-3-1)
01001101	(0-11-8) (0-5-11) (0-1-5) (5-6-11)
01001110	(3-6-11) (0-6-3) (0-5-6) (0-9-5)
01001111	(6-9-5) (6-11-9) (11-8-9)
01010000	(5-6-10) (4-8-7)
01010001	(4-0-3) (4-3-7) (6-10-5)
01010010	(1-0-9) (5-6-10) (8-7-4)
01010011	(10-5-6) (1-7-9) (1-3-7) (7-4-9)
01010100	(6-2-1) (6-1-5) (4-8-7)
01010101	(1-5-2) (5-6-2) (3-4-0) (3-7-4)
01010110	(8-7-4) (9-5-0) (0-5-6) (0-6-2)
01010111	(7-9-3) (7-4-9) (3-9-2) (5-6-9) (2-9-6)
01011000	(3-2-11) (7-4-8) (10-5-6)
01011001	(5-6-10) (4-2-7) (4-0-2) (2-11-7)
01011010	(0-9-1) (4-8-7) (2-11-3) (5-6-10)
01011011	(9-1-2) (9-2-11) (9-11-4) (7-4-11) (5-6-10)
01011100	(8-7-4) (3-5-11) (3-1-5) (5-6-11)
01011101	(5-11-1) (5-6-11) (1-11-0) (7-4-11) (0-11-4)
01011110	(0-9-5) (0-5-6) (0-6-3) (11-3-6) (8-7-4)
01011111	(6-9-5) (6-11-9) (4-9-7) (7-9-11)
01100000	(10-9-4) (6-10-4)
01100001	(4-6-10) (4-10-9) (0-3-8)
01100010	(10-1-0) (10-0-6) (6-0-4)
01100011	(8-1-3) (8-6-1) (8-4-6) (6-10-1)
01100100	(1-9-4) (1-4-2) (2-4-6)
01100101	(3-8-0) (1-9-2) (2-9-4) (2-4-6)
01100110	(0-4-2) (4-6-2)
01100111	(8-2-3) (8-4-2) (4-6-2)
01101000	(10-9-4) (10-4-6) (11-3-2)
01101001	(0-2-8) (2-11-8) (4-10-9) (4-6-10)
01101010	(3-2-11) (0-6-1) (0-4-6) (6-10-1)
01101011	(6-1-4) (6-10-1) (4-1-8) (2-11-1) (8-1-11)
01101100	(9-4-6) (9-6-3) (9-3-1) (11-3-6)

Continued on next page

Table A.1 – continued from previous page

Hexahedron Index	Triangles Edge Connectivity
01101101	(8-1-11) (8-0-1) (11-1-6) (9-4-1) (6-1-4)
01101110	(3-6-11) (3-0-6) (0-4-6)
01101111	(6-8-4) (11-8-6)
01110000	(7-6-10) (7-10-8) (8-10-9)
01110001	(0-3-7) (0-7-10) (0-10-9) (6-10-7)
01110010	(10-7-6) (1-7-10) (1-8-7) (1-0-8)
01110011	(10-7-6) (10-1-7) (1-3-7)
01110100	(1-6-2) (1-8-6) (1-9-8) (8-7-6)
01110101	(2-9-6) (2-1-9) (6-9-7) (0-3-9) (7-9-3)
01110110	(7-0-8) (7-6-0) (6-2-0)
01110111	(7-2-3) (6-2-7)
01111000	(2-11-3) (10-8-6) (10-9-8) (8-7-6)
01111001	(2-7-0) (2-11-7) (0-7-9) (6-10-7) (9-7-10)
01111010	(1-0-8) (1-8-7) (1-7-10) (6-10-7) (2-11-3)
01111011	(11-1-2) (11-7-1) (10-1-6) (6-1-7)
01111100	(8-6-9) (8-7-6) (9-6-1) (11-3-6) (1-6-3)
01111101	(0-1-9) (11-7-6)
01111110	(7-0-8) (7-6-0) (3-0-11) (11-0-6)
01111111	(7-6-11)
10000000	(7-11-6)
10000001	(3-8-0) (11-6-7)
10000010	(0-9-1) (11-6-7)
10000011	(8-9-1) (8-1-3) (11-6-7)
10000100	(10-2-1) (6-7-11)
10000101	(1-10-2) (3-8-0) (6-7-11)
10000110	(2-0-9) (2-9-10) (6-7-11)
10000111	(6-7-11) (2-3-10) (10-3-8) (10-8-9)
10001000	(7-3-2) (6-7-2)
10001001	(7-8-0) (7-0-6) (6-0-2)
10001010	(2-6-7) (2-7-3) (0-9-1)
10001011	(1-2-6) (1-6-8) (1-8-9) (8-6-7)
10001100	(10-6-7) (10-7-1) (1-7-3)
10001101	(10-6-7) (1-10-7) (1-7-8) (1-8-0)
10001110	(0-7-3) (0-10-7) (0-9-10) (6-7-10)
10001111	(7-10-6) (7-8-10) (8-9-10)
10010000	(6-4-8) (11-6-8)
10010001	(3-11-6) (3-6-0) (0-6-4)
10010010	(8-11-6) (8-6-4) (9-1-0)
10010011	(9-6-4) (9-3-6) (9-1-3) (11-6-3)
10010100	(6-4-8) (6-8-11) (2-1-10)
10010101	(1-10-2) (3-11-0) (0-11-6) (0-6-4)

Continued on next page

Table A.1 – continued from previous page

Hexahedron Index	Triangles Edge Connectivity
10010110	(4-8-11) (4-11-6) (0-9-2) (2-9-10)
10010111	(10-3-9) (10-2-3) (9-3-4) (11-6-3) (4-3-6)
10011000	(8-3-2) (8-2-4) (4-2-6)
10011001	(0-2-4) (4-2-6)
10011010	(1-0-9) (2-4-3) (2-6-4) (4-8-3)
10011011	(1-4-9) (1-2-4) (2-6-4)
10011100	(8-3-1) (8-1-6) (8-6-4) (6-1-10)
10011101	(10-0-1) (10-6-0) (6-4-0)
10011110	(4-3-6) (4-8-3) (6-3-10) (0-9-3) (10-3-9)
10011111	(10-4-9) (6-4-10)
10100000	(4-5-9) (7-11-6)
10100001	(0-3-8) (4-5-9) (11-6-7)
10100010	(5-1-0) (5-0-4) (7-11-6)
10100011	(11-6-7) (8-4-3) (3-4-5) (3-5-1)
10100100	(9-4-5) (10-2-1) (7-11-6)
10100101	(6-7-11) (1-10-2) (0-3-8) (4-5-9)
10100110	(7-11-6) (5-10-4) (4-10-2) (4-2-0)
10100111	(3-8-4) (3-4-5) (3-5-2) (10-2-5) (11-6-7)
10101000	(7-3-2) (7-2-6) (5-9-4)
10101001	(9-4-5) (0-6-8) (0-2-6) (6-7-8)
10101010	(3-2-6) (3-6-7) (1-0-5) (5-0-4)
10101011	(6-8-2) (6-7-8) (2-8-1) (4-5-8) (1-8-5)
10101100	(9-4-5) (10-6-1) (1-6-7) (1-7-3)
10101101	(1-10-6) (1-6-7) (1-7-0) (8-0-7) (9-4-5)
10101110	(4-10-0) (4-5-10) (0-10-3) (6-7-10) (3-10-7)
10101111	(7-10-6) (7-8-10) (5-10-4) (4-10-8)
10110000	(6-5-9) (6-9-11) (11-9-8)
10110001	(3-11-6) (0-3-6) (0-6-5) (0-5-9)
10110010	(0-8-11) (0-11-5) (0-5-1) (5-11-6)
10110011	(6-3-11) (6-5-3) (5-1-3)
10110100	(1-10-2) (9-11-5) (9-8-11) (11-6-5)
10110101	(0-3-11) (0-11-6) (0-6-9) (5-9-6) (1-10-2)
10110110	(11-5-8) (11-6-5) (8-5-0) (10-2-5) (0-5-2)
10110111	(6-3-11) (6-5-3) (2-3-10) (10-3-5)
10111000	(5-9-8) (5-8-2) (5-2-6) (3-2-8)
10111001	(9-6-5) (9-0-6) (0-2-6)
10111010	(1-8-5) (1-0-8) (5-8-6) (3-2-8) (6-8-2)
10111011	(1-6-5) (2-6-1)
10111100	(1-6-3) (1-10-6) (3-6-8) (5-9-6) (8-6-9)
10111101	(10-0-1) (10-6-0) (9-0-5) (5-0-6)
10111110	(0-8-3) (5-10-6)

Continued on next page

Table A.1 – continued from previous page

Hexahedron Index	Triangles Edge Connectivity
10111111	(10-6-5)
11000000	(11-10-5) (7-11-5)
11000001	(11-10-5) (11-5-7) (8-0-3)
11000010	(5-7-11) (5-11-10) (1-0-9)
11000011	(10-5-7) (10-7-11) (9-1-8) (8-1-3)
11000100	(11-2-1) (11-1-7) (7-1-5)
11000101	(0-3-8) (1-7-2) (1-5-7) (7-11-2)
11000110	(9-5-7) (9-7-2) (9-2-0) (2-7-11)
11000111	(7-2-5) (7-11-2) (5-2-9) (3-8-2) (9-2-8)
11001000	(2-10-5) (2-5-3) (3-5-7)
11001001	(8-0-2) (8-2-5) (8-5-7) (10-5-2)
11001010	(9-1-0) (5-3-10) (5-7-3) (3-2-10)
11001011	(9-2-8) (9-1-2) (8-2-7) (10-5-2) (7-2-5)
11001100	(1-5-3) (3-5-7)
11001101	(0-7-8) (0-1-7) (1-5-7)
11001110	(9-3-0) (9-5-3) (5-7-3)
11001111	(9-7-8) (5-7-9)
11010000	(5-4-8) (5-8-10) (10-8-11)
11010001	(5-4-0) (5-0-11) (5-11-10) (11-0-3)
11010010	(0-9-1) (8-10-4) (8-11-10) (10-5-4)
11010011	(10-4-11) (10-5-4) (11-4-3) (9-1-4) (3-4-1)
11010100	(2-1-5) (2-5-8) (2-8-11) (4-8-5)
11010101	(0-11-4) (0-3-11) (4-11-5) (2-1-11) (5-11-1)
11010110	(0-5-2) (0-9-5) (2-5-11) (4-8-5) (11-5-8)
11010111	(9-5-4) (2-3-11)
11011000	(2-10-5) (3-2-5) (3-5-4) (3-4-8)
11011001	(5-2-10) (5-4-2) (4-0-2)
11011010	(3-2-10) (3-10-5) (3-5-8) (4-8-5) (0-9-1)
11011011	(5-2-10) (5-4-2) (1-2-9) (9-2-4)
11011100	(8-5-4) (8-3-5) (3-1-5)
11011101	(0-5-4) (1-5-0)
11011110	(8-5-4) (8-3-5) (9-5-0) (0-5-3)
11011111	(9-5-4)
11100000	(4-7-11) (4-11-9) (9-11-10)
11100001	(0-3-8) (4-7-9) (9-7-11) (9-11-10)
11100010	(1-11-10) (1-4-11) (1-0-4) (7-11-4)
11100011	(3-4-1) (3-8-4) (1-4-10) (7-11-4) (10-4-11)
11100100	(4-7-11) (9-4-11) (9-11-2) (9-2-1)
11100101	(9-4-7) (9-7-11) (9-11-1) (2-1-11) (0-3-8)
11100110	(11-4-7) (11-2-4) (2-0-4)
11100111	(11-4-7) (11-2-4) (8-4-3) (3-4-2)

Continued on next page

Table A.1 – continued from previous page

Hexahedron Index	Triangles Edge Connectivity
11101000	(2-10-9) (2-9-7) (2-7-3) (7-9-4)
11101001	(9-7-10) (9-4-7) (10-7-2) (8-0-7) (2-7-0)
11101010	(3-10-7) (3-2-10) (7-10-4) (1-0-10) (4-10-0)
11101011	(1-2-10) (8-4-7)
11101100	(4-1-9) (4-7-1) (7-3-1)
11101101	(4-1-9) (4-7-1) (0-1-8) (8-1-7)
11101110	(4-3-0) (7-3-4)
11101111	(4-7-8)
11110000	(9-8-10) (10-8-11)
11110001	(3-9-0) (3-11-9) (11-10-9)
11110010	(0-10-1) (0-8-10) (8-11-10)
11110011	(3-10-1) (11-10-3)
11110100	(1-11-2) (1-9-11) (9-8-11)
11110101	(3-9-0) (3-11-9) (1-9-2) (2-9-11)
11110110	(0-11-2) (8-11-0)
11110111	(3-11-2)
11111000	(2-8-3) (2-10-8) (10-9-8)
11111001	(9-2-10) (0-2-9)
11111010	(2-8-3) (2-10-8) (0-8-1) (1-8-10)
11111011	(1-2-10)
11111100	(1-8-3) (9-8-1)
11111101	(0-1-9)
11111110	(0-8-3)
11111111	-

A.2 Prisms

Local Nodes	5	4	3	2	1	0
Index_2						

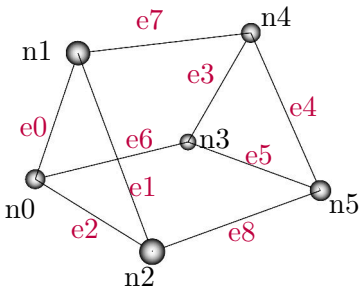


Figure A.2: *6–bit Prism index (left) and local edge and node numbering (right).*

Table A.2: *Marching Prisms Triangulation Table*

Prism Index	Triangles Edge Connectivity
000000	-
000001	(0-2-6)
000010	(0-7-1)
000011	(2-6-7) (1-2-7)
000100	(1-8-2)
000101	(0-1-8) (0-8-6)
000110	(0-7-2) (2-7-8)
000111	(6-7-8)
001000	(3-6-5)
001001	(0-2-3) (2-5-3)
001010	(0-6-1) (1-6-5) (1-5-7) (5-3-7)
001011	(1-2-5) (1-5-7) (5-3-7)
001100	(3-8-5) (1-8-3) (1-6-2) (3-6-1)
001101	(0-1-3) (1-8-3) (3-8-5)
001110	(0-6-2) (3-7-8) (3-8-5)
001111	(3-7-8) (3-8-5)
010000	(3-4-7)
010001	(0-2-7) (2-4-7) (2-6-3) (2-3-4)
010010	(0-3-1) (1-3-4)
010011	(1-2-4) (2-6-4) (6-3-4)
010100	(1-7-2) (2-7-3) (2-3-8) (3-4-8)
010101	(0-1-7) (6-3-8) (3-4-8)
010110	(3-4-8) (0-3-2) (2-3-8)
010111	(3-8-6) (3-4-8)
011000	(4-6-5) (4-7-6)
011001	(0-2-7) (2-4-7) (2-5-4)
011010	(0-6-1) (1-6-5) (1-5-4)
011011	(1-2-4) (2-5-4)
011100	(1-7-2) (2-7-6) (4-8-5)

Continued on next page

Table A.2 – continued from previous page

Prism Index	Triangles Edge Connectivity
011101	(0-1-7) (4-8-5)
011110	(0-6-2) (4-8-5)
011111	(4-8-5)
100000	(4-5-8)
100001	(0-4-6) (6-4-5) (0-2-8) (0-8-4)
100010	(1-0-8) (0-5-8) (0-7-5) (4-5-7)
100011	(1-2-6) (1-6-7) (4-5-8)
100100	(1-4-2) (2-4-5)
100101	(0-1-6) (1-5-6) (1-4-5)
100110	(0-7-2) (2-7-4) (2-4-5)
100111	(4-5-6) (4-6-7)
101000	(3-8-4) (3-6-8)
101001	(0-2-3) (2-8-3) (3-8-4)
101010	(0-7-1) (3-6-8) (3-8-4)
101011	(1-2-8) (3-7-4)
101100	(1-4-3) (1-3-6) (2-1-6)
101101	(0-1-3) (1-4-3)
101110	(0-6-2) (3-7-4)
101111	(3-7-4)
110000	(3-5-8) (3-8-7)
110001	(0-2-6) (3-5-8) (3-8-7)
110010	(0-3-1) (1-3-8) (3-5-8)
110011	(1-2-8) (3-5-6)
110100	(1-7-2) (2-7-3) (2-3-5)
110101	(3-5-6) (0-1-7)
110110	(0-3-2) (2-3-5)
110111	(3-5-6)
111000	(6-8-7)
111001	(0-2-7) (2-8-7)
111010	(0-8-1) (0-6-8)
111011	(1-2-8)
111100	(1-7-2) (2-7-6)
111101	(0-1-7)
111110	(0-6-2)
111111	-

A.3 Pyramids

Local Nodes	4	3	2	1	0
$Index_2$					

Figure A.3: *5–bit Pyramid index (left) and local edge and node numbering (right).*

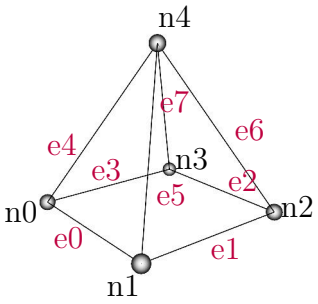


Table A.3: *Marching Pyramids Triangulation Table*

Pyramid Index	Triangles Edge Connectivity
00000	-
00001	(0-3-4)
00010	(0-5-1)
00011	(1-3-5) (3-4-5)
00100	(1-6-2)
00101	(0-4-1) (1-4-6) (2-4-3) (2-6-4)
00110	(0-5-2) (2-5-6)
00111	(3-4-2) (2-4-6) (4-5-6)
01000	(2-7-3)
01001	(0-2-4) (4-2-7)
01010	(0-5-3) (3-5-7) (1-2-5) (2-7-5)
01011	(1-2-5) (2-7-5) (4-5-7)
01100	(1-6-3) (3-6-7)
01101	(0-1-6) (0-6-4) (4-6-7)
01110	(0-5-3) (3-5-7) (5-6-7)
01111	(4-5-7) (5-6-7)
10000	(4-7-5) (5-7-6)
10001	(0-3-7) (0-7-5) (5-7-6)
10010	(0-4-1) (1-4-6) (4-7-6)
10011	(1-3-6) (3-7-6)
10100	(1-5-2) (2-5-7) (4-7-5)
10101	(0-1-5) (3-7-2)
10110	(0-4-2) (2-4-7)
10111	(2-3-7)
11000	(2-6-4) (3-2-4) (4-6-5)
11001	(0-2-5) (2-6-5)
11010	(0-4-3) (1-2-6)
11011	(1-2-6)
11100	(1-5-4) (1-4-3)

Continued on next page

Table A.3 – continued from previous page

Pyramid Index	Triangles Edge Connectivity
11101	(0-1-5)
11110	(0-4-3)
11111	-

A.4 Tetrahedra

Local Nodes	3	2	1	0
Index_2				

Figure A.4: *4–bit Tetrahedron index (left) and local edge and node numbering (right).*

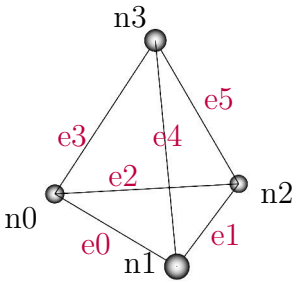


Table A.4: *Marching Tetrahedra Triangulation Table*

Tetrahedron Index	Triangles Edge Connectivity
0000	-
0001	(0-2-3)
0010	(0-4-1)
0011	(1-2-4) (2-3-4)
0100	(1-5-2)
0101	(0-1-3) (1-5-3)
0110	(0-4-2) (2-4-5)
0111	(3-4-5)
1000	(3-5-4)
1001	(0-2-4) (2-5-4)
1010	(0-3-5) (0-5-1)
1011	(1-2-5)
1100	(1-4-2) (2-4-3)
1101	(0-1-4)
1110	(0-3-2)
1111	-

A.5 Quadrilaterals

In the case of Quadrilateral for the reconstruction of the boundary of a domain if is part of the FSI, the original nodes of the boundary are also incorporated. For that reason a triangle is denoted as "(e1-n1-e2)" corresponds to a triangle generated by connected the generated nodes on edges e1 and e2 and the original node n1 of the element.

Local Nodes	3	2	1	0
Index_2				

Figure A.5: *4–bit Quadrilateral index (left) and local edge and node numbering (right).*

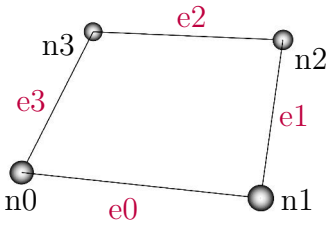


Table A.5: *Marching Quadrilaterals Triangulation Table*

Quadrilateral Index	Triangles Edge Connectivity
0000	-
0001	(n0-e0-e3)
0010	(e0-n1-e1)
0011	(n0-e1-e3) (n0-n1-e3)
0100	(e1-n2-e2)
0101	(e3-n0-e0) (e3-e0-e2) (e2-e0-e1) (e2-e1-n2)
0110	(e0-n1-n2) (e0-2-e2)
0111	(e3-n0-n1) (e2-e3-n1) (n1-n2-e2)
1000	(e2-n3-e3)
1001	(e2-n3-n0) (e2-n0-e0)
1010	(e2-n3-e3) (e2-e3-e0) (e2-e0-e1) (e1-e0-n1)
1011	(e2-n3-n0) (e2-n0-e1) (e1-n0-n1)
1100	(n3-e3-e1) (n2-n3-e1)
1101	(n3-n0-e0) (e1-n3-e0) (n2-n3-e1)
1110	(n2-n3-e3) (n2-e3-e0) (n2-e0-n1)
1111	(n0-n1-n3,) (n1-n2-n3)

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Ρευστών
Μονάδα Παράλληλης Υπολογιστικής Ρευστοδυναμικής
& Βελτιστοποίησης

Ανάπτυξη Λογισμικού Μετάβασης από Τοπολογία σε Σχήμα μέσω Αλγορίθμων Προέλασης

Διπλωματική Εργασία – Εκτενής Περίληψη στα Ελληνικά

Ευστράτιος Τραχανιάς

Επιβλέπων:
Κυριάκος Χ. Γιαννάκογλου, Καθηγητής ΕΜΠ

Αθήνα, 2024

Εισαγωγή

Η εξέλιξη των υπολογιστών και η πρόοδος των υπολογιστικών μεθόδων έχουν αλλάξει σημαντικά την προκαταρκτική μελέτη στη βιομηχανική σχεδιάση. Η ευρεία διαθεσιμότητα εργαλείων Υπολογιστικής Ρευστοδυναμικής (ΥΡ) επιτρέπουν στους μηχανικούς να προβλέψουν την απόδοση και να εντοπίσουν πιθανά προβλήματα πριν από την κατασκευή φυσικών πρωτοτύπων. Ο συνδυασμός της ΥΡ με μεθόδους βελτιστοποίησης όπως η Βελτιστοποίηση Μορφής (ΒεΜο) και η Βελτιστοποίηση Τοπολογίας (ΒεΤο) χρησιμοποιούνται συχνά για την ευρέση βέλτιστων γεωμετριών σε ρευστοδυναμικές/αεροδυναμικές μελέτες.

Η διπλωματική εργασία παρουσιάζει την ανάπτυξη λογισμικού *Μετάβασης από Τοπολογία σε Σχήμα (Topology–to–Shape Transition [TtoST])* για την επεξεργασία των αποτελεσμάτων ΒεΤο σε θέματα ρευστομηχανικής, με σκοπό την αναπαράσταση του τοιχώματος μεταξύ στερεού και ρευστού (Fluid–Solid Interface [FSI]) υπό τη μορφή επιφανειακού πλέγματος. Για το σκοπό αυτό, υιοθετείται ο αλγόριθμος Marching Cubes (MC) [22], ιδιαίτερα διάσημος στα γραφικά υπολογιστών. Η επέκταση της ιδέας του αλγορίθμου σε άλλα γεωμετρικά σχήματα όπως εξάεδρα, πρίσματα, πυραμίδες, τετράεδρα και τετράπλευρα δίνει τη δυνατότητα εφαρμογής του σε μη–δομημένα/υβριδικά πλέγματα. Η δομή της εργασίας περιλαμβάνει την βασική θεωρία της ΒεΜο και ΒεΤο, τη θεωρία του αλγορίθμου MC και την επέκταση της ιδέας σε άλλα γεωμετρικά σχήματα, την υλοποίηση της μεθοδολογίας σε C++, και τα αποτελέσματα από διδιάστατες και τριδιάστατες εφαρμογές, καταλήγοντας σε συμπεράσματα και προτάσεις για μελλοντική έρευνα.

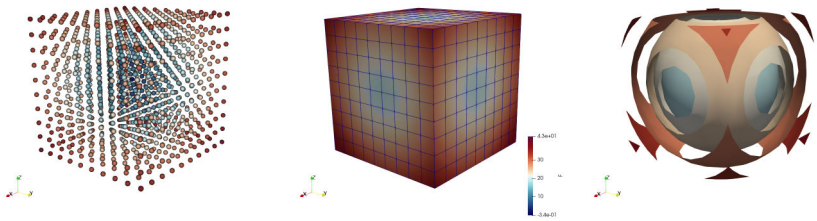
Βελτιστοποίηση Μορφής και Τοπολογίας

Τα περισσότερα προβλήματα βελτιστοποίησης στο πεδίο της ρευστομηχανικής μπορούν να κατηγοριοποιηθούν είτε ως ΒεΜο είτε ως ΒεΤο. Η ΒεΜο περιλαμβάνει τη χρήση μεταβλητών σχεδιασμού, που παραμετροποιούν την υπο εξέταση γεωμετρία για τη βελτίωση της απόδοσης της, ενώ η ΒεΤο παράγει γεωμετρίες ανεξάρτητα από ένα αρχικό σχήμα, εξερευνώντας μια ευρύτερη γκάμα δυνατοτήτων σχεδιασμού.

Στην ΒεΤο, ένα τεχνητό πεδίο πορώδους εισάγεται στις εξισώσεις ροής, και λειτουργεί ως μεταβλητή σχεδιαμού στους κόμβους ή στα κέντρα των κυψελών του υπολογιστικού χωρίου. Στόχος της ΒεΤο είναι ο υπολογισμός του βέλτιστου πεδίου του πορώδους που ελαχιστοποιεί τη συνάρτηση στόχο. Ο αριθμός των μεταβλητών σχεδιασμού του προβλήματος είναι ίσος με τον αριθμό των κόμβων ή των κελιών του υπολογιστικού χωρίου (συνεπώς πολύ μεγάλος). Έτσι, η χρήση της συζυγούς μεθόδου για τον υπολογισμό των παραγώγων ευαισθησίας αποτελεί την τέλεια επιλογή, αφού το υπολογιστικό κόστος της, είναι ανεξάρτητο του αριθμού των μεταβλητών σχεδιασμού. Η ΒεΤο έχει αποτελέσει βασικό θέμα για αρκετές δημοσιεύσεις της ερευνητικής ομάδας της Παράλληλης Υπολογιστικής Ρευστοδυναμικής & Βελτιστοποίησης του ΕΜΠ (ΜΠΥΡΒ/ΕΜΠ),[2, 3, 4, 5],οι οποίες αποτελούν και βασική πηγή αυτής της διπλωματικής εργασίας.

Ωστόσο, ένα μειονέκτημα της ΒεΤο είναι ο μη καθορισμός του τοιχώματος μεταξύ στερεού και ρευστού, όπου θα επιβληθούν οι οριακές συνθήκες, αλλά ο έμμεσος χειρισμός του από το πεδίο πορώδους. Το παραπάνω εισάγει ανακρίβειες στα αποτελέσματα και πολλές φορές όταν η επίλυση γίνεται με ανάλυση ΥΡ σε σωματόδετα πλέγματα στις προκύπτουσες γεωμετρίες, δεν είναι τα αναμενόμενα. Η εξαγωγή της γεωμετρίας από το βελτιστοποιημένο πορώδες της ΒεΤο είναι απαραίτητη, για την ενδεχόμενη κατασκευή της, την μετάβαση σε υψηλης πιστότητας ΥΡ με επιβεβλημένες οριακές συνθήκες στο τοίχωμα, ακόμα και την επακόλουθη ΒεΜο για περαιτέρω βελτίωση. Η διπλωματική εργασία επικεντρώνεται στην εξαγωγή της διεπιφάνειας στερεού–ρευστού απο το πορώδες που προκύπτει από την ΒεΤο, υπο τη μορφή επιφανειακού πλέγματος, μέσω Αλγορίθμων Προέλασης, όπως θα παρουσιαστεί στη συνέχεια.

Αλγόριθμοι Προέλασης



(α) *(β)* *(γ)*

Σχήμα 1: *Σφαιρικό πεδίο επεκτεινόμενο προς τα έξω σε κυβικό πλέγμα. (α) Τιμές πεδίου στους κόμβους του πλέγματος, (β) το δομημένο κυβικό πλέγμα, (γ) δείγματα ισο-επιφανειών.*

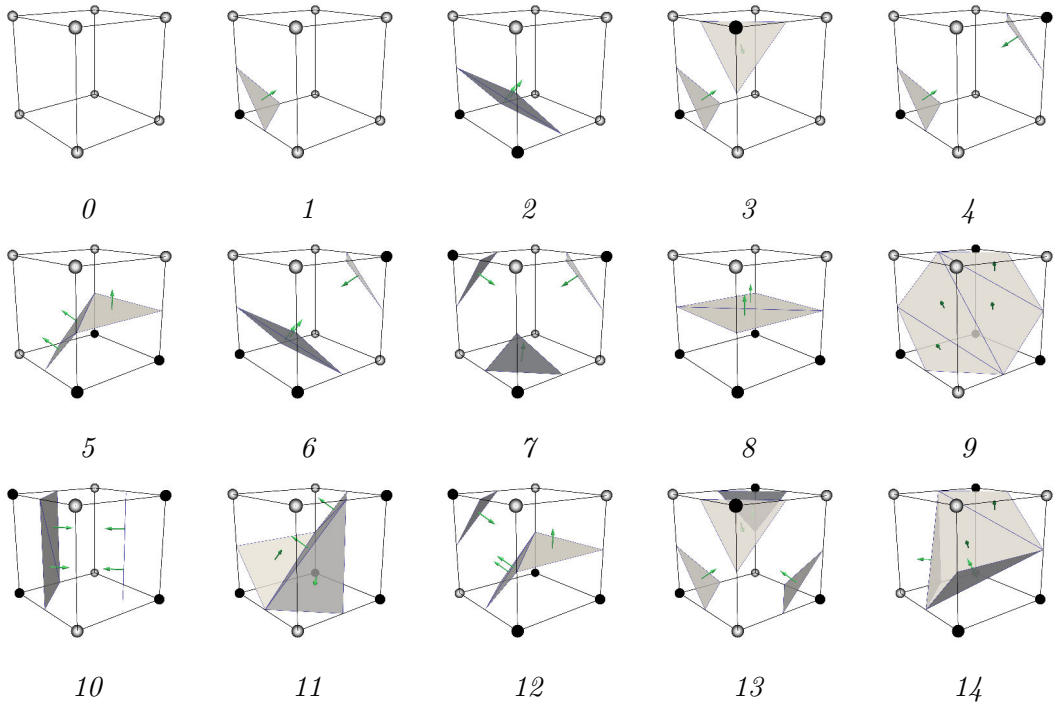
Ο αλγόριθμος Marching Cubes

Ο αλγόριθμος Marching Cubes (MC) εισήχθει το 1987 [22], και εφαρμόζεται σε τριδιάστατα δομημένα κυβικά πλέγματα με πραγαμτικές τιμές πεδίου στους κόμβους του πλέγματος. Ο χρήστης καθορίζει μια τίμη κατωφλιού, που αναφέρεται ως *ισοτιμή*, και ο αλγόριθμος εξάγει την τριδιάστατη επιφάνεια που της αντιστοιχεί, καλούμενη *ισοεπιφάνεια*. Ένα δομημένο κυβικό πλέγματα με τιμές πεδίου στους κόμβους και δείγματα ισοεπιφάνειών φαίνονται στο Σχ. 1. Ο αλγόριθμος επεξεργάζεται κάθε κυψέλη–κύβο του πλέγματος και καθορίζει μια διαμόρφωση τριγώνων που αναπαριστά τον τρόπο με τον οποίο η ισοεπιφάνεια διέρχεται από αυτόν. Κατά την επεξεργασία, η τιμή του κάθε κόμβου–κορυφή του εκάστοτε κύβου συγκρίνεται με την ισοτιμή, και κάθε κόμβος χαρακτηρίζεται ως 'εσωτερικός–μαύρος' ή 'εξωτερικός–λευκός' της ισοεπιφάνειας. Η κατάσταση των κόμβων και κατ' επέκταση της κυψέλης–κύβου κωδικοποιείται σε ένας δείκτη (8-bit) ο οποίος αντισοιχείται σε αποθηκευμένες λίστες μοτίβων–διαμορφώσεων τριγώνων (βλέπε Παράρτημα Α). Με τη χρήση συμμετριών ο αριθμός των μοναδικών μοτίβων– διαμορφώσεων μειώνεται σημαντικά. Ο αρχικός αλγόριθμος MC κάνει χρήση της περιστροφικής και της συμπληρωματικής συμμετρίας καταλήγωντας σε 15 μοναδικά

μοτίβα διαμορφώσεων, τα οποία παρατίθενται στο Σχ. 2. Η θέση του σημείου τομής $\vec{r}_{\text{is}}$ της ισοεπιφάνειας με ισοτιμή α με τις ακμές της κυψέλης–κύβου του πλέγματος υπολογίζονται με γραμμική παρεμβολή σύμφωνα με τη σχέση 1:

$$\vec{r}_{\text{is}} = \vec{r}_{V1} + \rho(\vec{r}_{V2} - \vec{r}_{V1}) \,, \qquad \rho = \frac{\alpha - f_1}{f_2 - f_1} \tag{1}$$

όπου $\vec{r}_{V1}$, f_1 και $\vec{r}_{V2}$, f_2 η θέση και η τιμή πεδίου των κορυφών 1 και 2 αντίστοιχα.

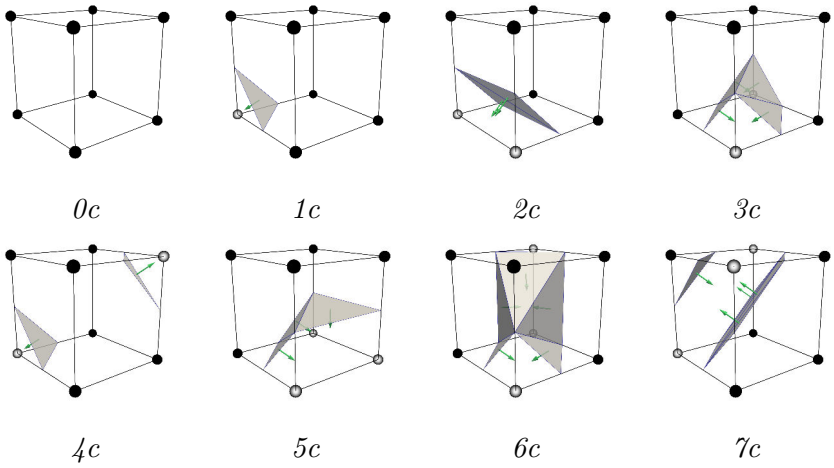


Σχήμα 2: *Οι 15 διαμορφώσεις του αρχικού αλγορίθμου Marching Cubes (όμοια με [22]). Κόμβοι με άσπρο χρώμα βρίσκονται εκτός, ενώ κόμβοι με μαύρο βρίσκονται εντός της ισοεπιφάνειας υπό εξέταση. Παραγόμενα τρίγωνα με γκρι και τα κάθετα σε αυτά διανύσματα φαίνονται ως πράσινα βέλη. Εως και 4 τρίγωνα μπορούν να παραχθούν για κάθε κελί (κύβο) του αρχικού αλγορίθμου.*

Επίλυση αμφισημιών

Μετά την εμφάνιση του αλγορίθμου MC, εκτεταμένη βιβλιογραφία [41, 42, 43, 44, 45, 46] έχει εκδοθεί σχετικά με τα διαφορετικά μοτίβα που κάποιες διαμορφώσεις μπορούν να έχουν, αναφερόμενα ως αμφισημίες. Οι αμφισημίες μπορούν να χωριστούν σε αμφισημίες *έδρας* και σε *εσωτερικές* αμφισημίες. Ως προς τις αμφισημίες στις έδρες των κύβων αυτές προκύπτουν στις περιπτώσεις όπου δύο διαγώνιοι κόμβοι της έδρας βρίσκονται μέσα στην ισοεπιφάνεια, και οι άλλοι δύο εκτός. Αποδείχθηκε ότι οι αμφισημίες έδρας οφείλονται στην χρήση της συμπληρωματικής συμμετρίας [44]. Για την άμεση επίλυση των αμφισημιών έδρας εισάγονται οι συμπληρωματικές διαμορφώσεις των περιπτώσεων που εμφανίζουν αμφισημία, δημιουργώντας τον διευρυμένο αλγόριθμο MC,

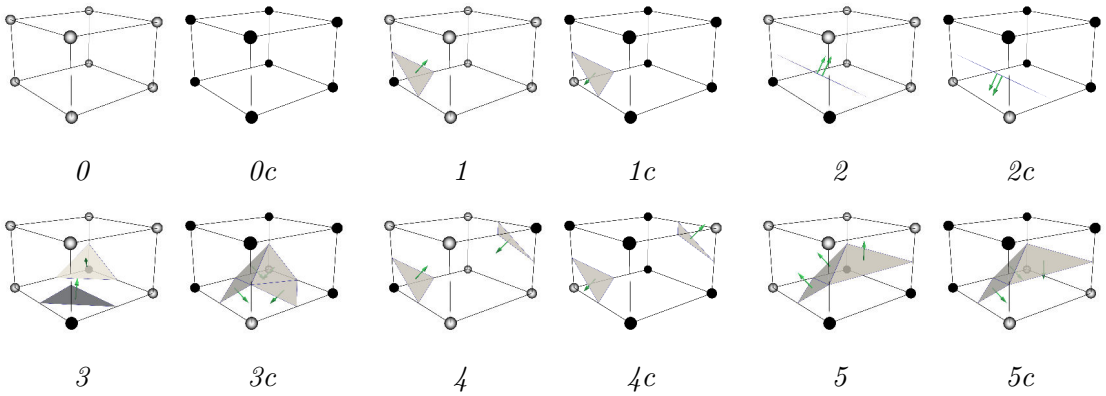
οι οποίες φαίνονται στο Σχ. 3. Οι εσωτερικές αμφισημίες αφορούν στην άγνοια της συμπεριφοράς της ισοεπιφάνειας εντός της κυψέλης–κύβου. Οι εσωτερικές αμφισημίες δεν αποτελούν αντικείμενο μελέτης της διπλωματικής εργασίας.



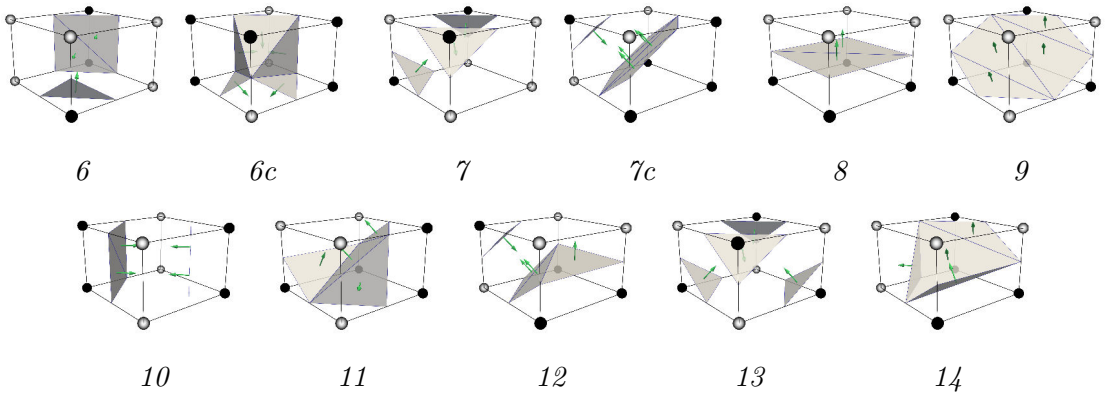
Σχήμα 3: *Συμπληρωματικές διαμορφώσεις των αρχικών περιπτώσεων 0, 1, 2, 3, 4, 5, 6, 7 για την ευθεία επίλυση των αμφισημιών.*

Επέκταση σε μη–δομημένα πλέγματα

Πολύ συχνά σε μελέτες ΥΡ το υπολογιστικό χωρίο αποτελείται και από αλλα είδη κυψελών πέραν των κύβων. Στην περίπτωση αυτή μιλάμε για ένα μη–δομημένο/υβριδικό πλέγμα το οποίο μπορεί να περιλαμβάνει εξάεδρα, πρίσματα, πυραμίδες & τετράεδρα. Στη διπλωματική εργασία, η ιδεά του αλγορίθμου MC επεκτείνεται στην περίπτωση των άλλων γεωμετρικών στοιχείων, δημιουργώντας τους αντίστοιχους γενικευμένους Αλγορίθμους Προέλασης. Ως προς τα εξάεδρα, τα μοτίβα τριγωνοποίησης είναι κοινά με αυτά του διευρυμένου αλγορίθμου MC για τους κύβους, και παρουσιάζονται συνολικά στο Σχ. 4.

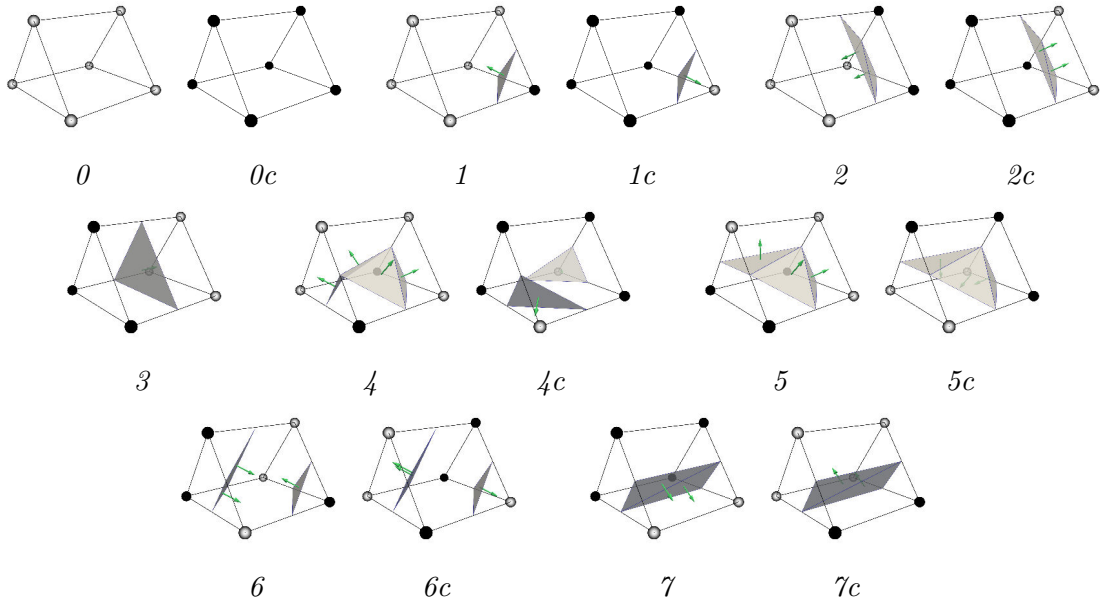


Σχήμα 4: *Συνέχεια στην επόμενη σελίδα*



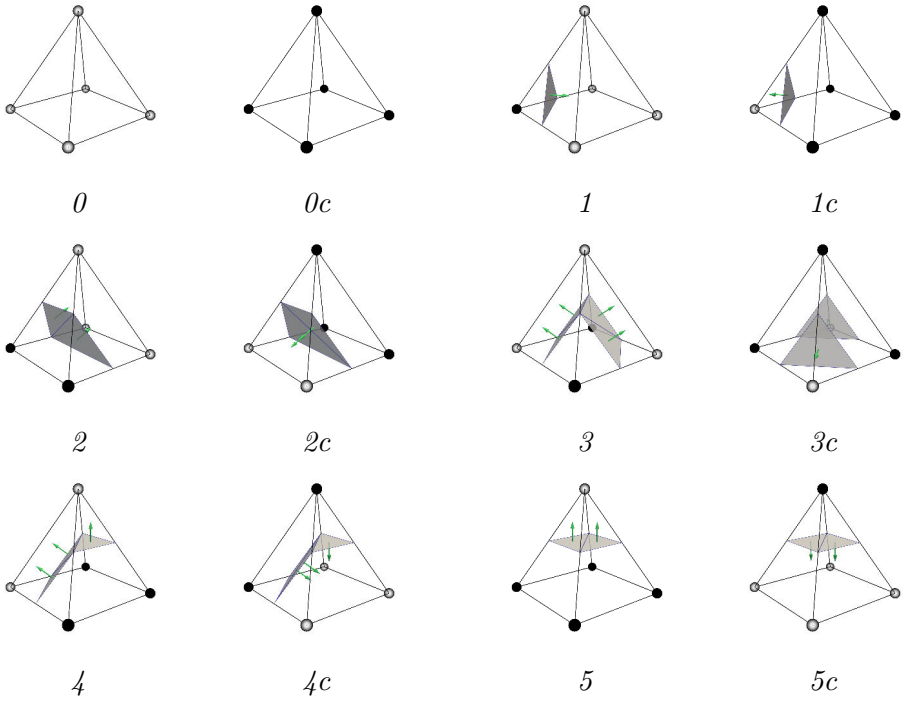
Σχήμα 4: *Οι 23 διαμορφώσεις των Εξαέδρων. Τα τρίγωνα που προκύπτουν φαίνοντα με γκρι χρώμα και τα κάθετα σε αυτά διανύσματα με πράσινο βέλος. Εως και 5 τρίγωνα μπορούν να προκύψουν από τον διερυμένο αλγόριθμο για τα εξάεδρα.*

Για τα πρίσματα τα μοτίβα τριγωνοποίησης φαίνονται στο Σχ. 5, με εως και 4 τρίγωνα να προκύπτουν για κάθε κυψέλη–πρίσμα.

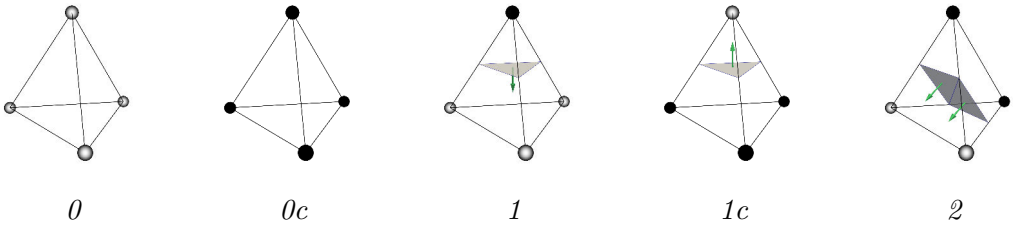


Σχήμα 5: *Οι 15 διαμορφώσεις για τα πρίσματα. Τα τρίγωνα που προκύπτουν φαίνοντα με γκρι χρώμα και τα κάθετα σε αυτά διανύσματα με πράσινο βέλος.*

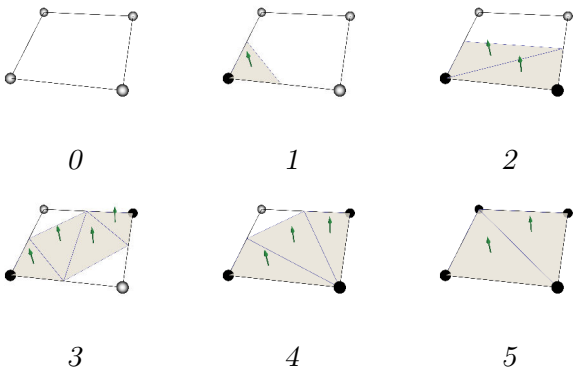
Αντίστοιχα, για τις πυραμίδες φαίνονται στο Σχ. 6, με εως και 4 τρίγωνα να προκύπτουν για κάθε κυψέλη–πυραμίδα. Για τα τετράεδρα στο Σχ. 7, με εως και 2 τρίγωνα να προκύπτουν. Για το τετραπλευρικό όριο του υπολογιστικού χωρίου στο Σχ. 8, με εως και 4 προκύπτοντα τρίγωνα.



Σχήμα 6: *Οι 12 διαμορφώσεις για τις πυραμίδες.*



Σχήμα 7: *Οι 5 διαμορφώσεις για τα τετράεδρα.*

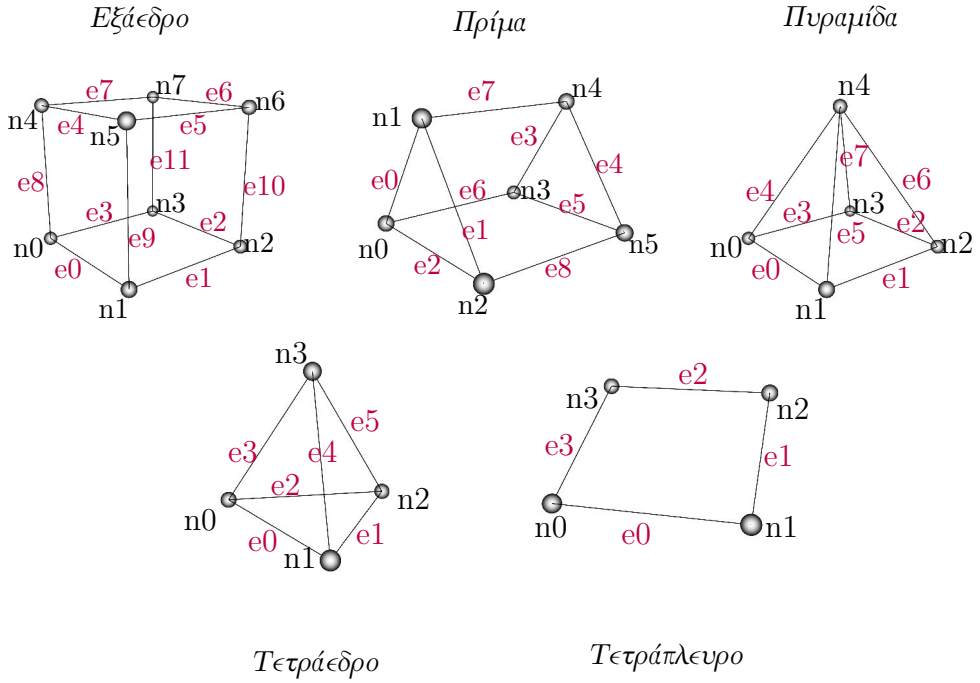


Σχήμα 8: *Οι 6 διαμορφώσεις για τα Τετράπλευρα. Οριακοί κόμβοι του υπολογιστικού χωρίου που βρίσκονται μέσα από την ισοεπιφάνεια με μαύρο χρώμα, εκτός αυτής με λευκό χρώμα. Προκύπτοντα τρίγωνα με γκρί με τα κάθετα σε αυτά διανύσμαατα (πράσινο βέλος).*

Υλοποίηση αλγορίθμου

Στη συνέχεια παρουσιάζεται η υλοποίηση της προτεινόμενης μεθόδου. Πρώτο βήμα του αλγορίθμου αποτελεί η εισαγωγή του υπολογιστικού χωρίου. Η γεωμετρία και η συνεκτικότητα των κόμβων του χωρίου μπορεί να φορτωθεί είτε από την οικεία μορφή αρχείου εισόδου είτε από το περιβάλλον του λογισμικού OpenFOAM©. Επόμενο βήμα είναι η ανάγνωση του πεδίο πορώδους της λύσης BeTo. Στην περίπτωση που το πεδίο πορώδους είναι στα κέντρα των κύψελων εκτελείται σταθμισμένη παρεμβολή στους κόμβους με βάση τους όγκους των κυψελών. Κατόπιν, δημιουργείται η απαραίτητη δομή δεδομένων για την μοναδική αρίθμηση όλων των ακμών του υπολογιστικού χωρίου. Κάθε στοιχείο του πλέγματος επεξεργάζεται, στη συνέχεια, με τον κατάλληλο Αλγόριθμο Προέλασης, ανάλογα με τον τύπο του. Το τελικό βήμα της διαδικασίας είναι η παραγωγή του αρχείου εξόδου που περιέχει τη συλλογή των τριγώνων που σχηματίζουν την αναπαράσταση του επιφανειακού πλέγματος της ισοεπιφάνειας. Η διαδικασία συνοψίζεται σε ένα διάγραμμα ροής όπως φαίνεται στο Σχ. 4.1 της αγγλικής έκδοσης.

Πλεονέκτημα του αλγορίθμου είναι ότι κάθε σημείο τομής ακμής–ισοεπιφάνειας $\vec{r}_{\text{is}}$ (σχέση 1) χρειάζεται να υπολογιστεί μόνο μία φορά. Το σημείο τομής μπορεί να επαναχρησιμοποιηθεί κατά την επεξεργασία των επόμενων στοιχείων που μοιράζονται την ίδια ακμή. Η παρατήρηση αυτή υπαγορεύει την ανάγκη για μια *Δομή Δεδομένων* που χαρτογραφεί όλες τις ακμές του υπολογιστικού πλέγματος και κρατά πληροφορίες για εκείνες που τέμνονται. Ως εκ τούτου, μετά την αρχική φόρτωση του υπολογιστικού πλέγματος, δημιουργείται η Δομή Δεδομένων. Η δημιουργία της δομής δεδομένων βασίζεται στην τοπική αρίθμηση των ακμών των στοιχείων όπως φαίνεται στο Σχ. 9.



Σχήμα 9: *Τοπική αρίθμηση κόμβων και ακμών για τα γεωμετρικά στοιχεία.*

Η τοπική αρίθμηση των ακμών ενός στοιχείου αποδίδεται στον χάρτη ακμών για το πρώτο στοιχείο που υφίσταται επεξεργασία. Για κάθε επόμενο στοιχείο λαμβάνεται υπόψη αρχικά η τοπική αρίθμηση συν το τρέχον αναγνωριστικό ακμής, το οποίο παρακολουθείται μέσω ενός μετρητή. Εάν μια ακμή έχει ήδη αριθμηθεί, τότε θα διατηρήσει την αρχική της αρίθμηση. Οι ακμές των στοιχείων αναγνωρίζονται από τα ζεύγη των αναγνωριστικών κόμβων που συνδέουν, φροντίζοντας πάντα ο μικρότερος να είναι πρώτος. Το ζεύγος των αναγνωριστικών κόμβων χρησιμοποιείται ως κλειδί σε έναν πίνακα κατακερματισμού για να προσπελάσει τον χάρτη ακμών που περιέχει τα αναγνωριστικών ακμών. Η παραπάνω διαδικασία διασφαλίζει ότι σε κάθε ακμή αποδίδεται ένα μοναδικό αναγνωριστικό.

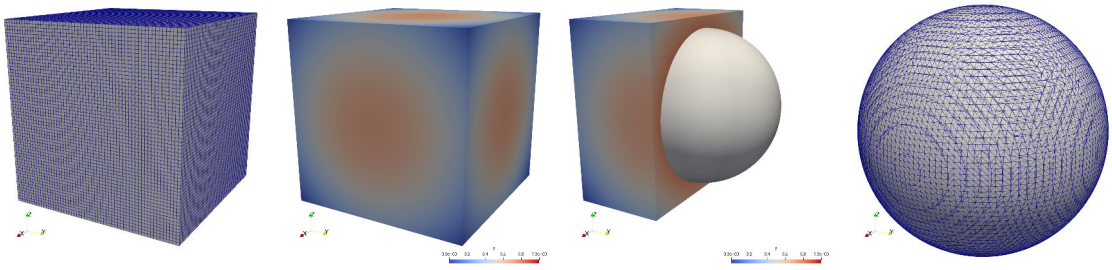
Μετά τη δημιουργία της δομής δεδομένων, πραγματοποιείται η αρχικοποίηση τριών πινάκων με σκοπό την παρακολούθηση των ακμών στις οποίες δημιουργείται κόμβος–κορυφή (του επιφανειακού πλέγματος της ισοεπιφάνειας) καθώς και την οργανωμένη αποθήκευση των νέων κόμβων–κορυφών που έχουν παρεμβληθεί σε αυτές, για μεταγενέστερη ανάκτηση. Αναλυτικότερα, ο αλγόριθμος αρχικοποιεί:

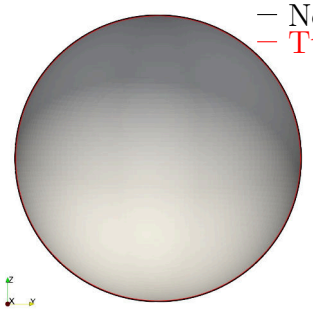
- Έναν πίνακα αληθείας (boolean), `edgesBool`, για την παρακολούθηση των επεξεργασμένων ακμών.
- Έναν πίνακα, `edgesCounter`, ο οποίος αποθηκεύει τα αναγνωριστικά των νεών κόμβων–κορυφών που υπολογίζονται με γραμμική παρεμβολή μεταξύ των κόμβων μιας ακμής.
- Έναν πίνακα, `Isonodes`, ο οποίος αποθηκεύει τις συντεταγμένες αυτών των νεων κόμβων–κορυφών.

Το αρχείο εξόδου που παράγεται περιέχει τη συλλογή των τριγώνων που αποτελούν το επιφανειακό πλέγμα της ισοεπιφάνειας. Το αρχείο εξόδου μπορεί να λάβει τη μορφή ενός αρχείου STL (STereoLithography), το πιο δημοφιλές αρχείο εισόδου για τους (εμπορικούς) πλεγματοποιητές, οπότε μπορεί να εισάγεται απευθείας σε έναν πλεγματοποιητή. Επίσης, μπορεί να εξάγεται σε μορφή αρχείου VTK (Visualization Toolkit) για οπτικοποίηση στο λογισμικό ParaView.

Εφαρμογές

Το λογισμικό επαληθεύεται σε μια σειρά σεναρίων δοκιμής σε δείγματα δομημένων και υβριδικών πλευγμάτων. Αρχικά ο αλγόριθμος δοκιμάζεται στο σενάριο εξαγωγής της γεωμετρίας μιας σφαίρας σε ένα δομημένο πυκνό πλέγμα (Σχ. 10), και σε ένα υβριδικό πλέγμα λιγότερο πυκνό πλέγμα (Σχ. 11) και ελέγχεται η απόκλιση της προκύπτουσας ακτίνας από την ονομαστική. Κατόπιν εφαρμόζεται στην περίπτωση πεδίου δύο επικαλυπτόμενων σφαιρών ίδιας ακτίνας με μικρή απόσταση μεταξύ τους. Στην περίπτωση αυτή ελέγχεται η συμπεριφορά του αλγορίθμου ανάλογα με την σχετική θέση διαχωρισμού των σφαιρών. Όταν αυτή γίνεται σε διαφορετικά κελιά ο αλγόριθμος παράγει δύο ξεχωριστές οντότητες ενώ όταν γίνεται εντός κοινών κελιών ο αλγόριθμος τις ενώνει, όπως φαίνεται στο Σχ. 12.

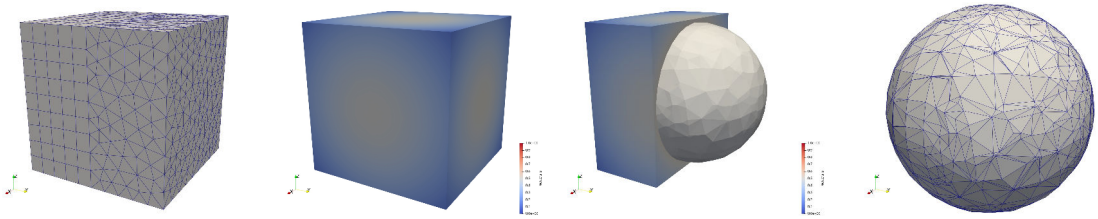


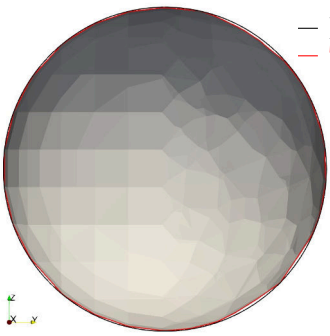


Πίνακας 1: *Απόκλιση της ακτίνας από την ονομαστική*

Απόκλιση	TtoST - Nominal
Min	-8.05454×10^{-5}
Max	-1.44176×10^{-9}
Mean	-2.72957×10^{-5}
σ	2.17697×10^{-5}

Σχήμα 10: *Εφαρμογή της μεθόδου σε δομημένο κυβικό πλέγμα (πάνω αριστερά) με ενα πεδίο πορώδους β στους κόμβους (πάνω μέση), σε μορφή σφαίρας που βαίνει μειούμενο οσο απομακρύνεται από το κέντρο. Η προκύπτουσα σφαίρα από την TtoST (πάνω δεξιά) και η απόκλιση της ακτίνας της από την ονομαστική (κάτω αριστερά & Πίνακας 1).*

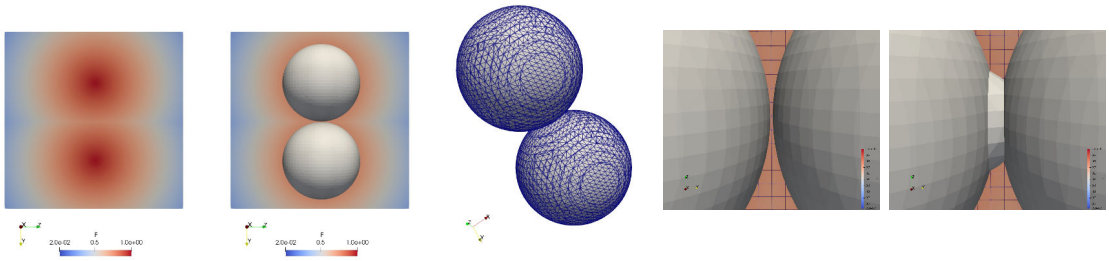




Πίνακας 2: *Απόκλιση της ακτίνας από την ονομαστική*

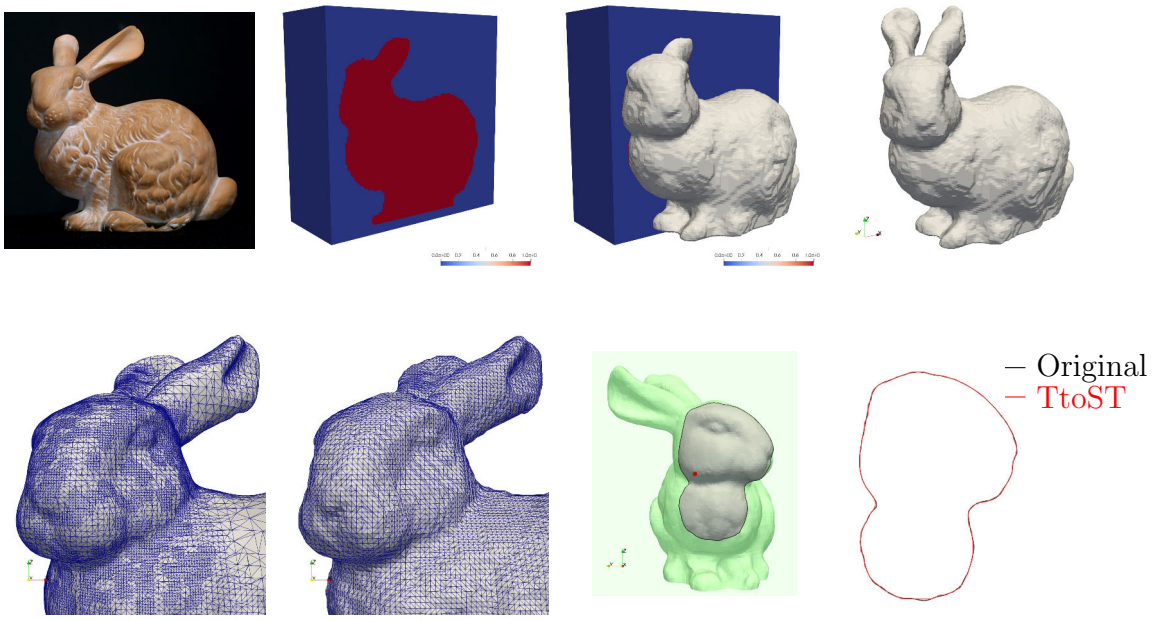
Απόκλιση	TtoST - Nominal
Min	-8.2492×10^{-3}
Max	-4.7228×10^{-7}
Mean	-1.3942×10^{-3}
σ	1.3625×10^{-3}

Σχήμα 11: *Εφαρμογή της μεθόδου σε υβριδικό πλέγμα (πάνω αριστερά) με ενα πεδίο πορώδους β στους κόμβους (πάνω μέση), σε μορφή σφαίρας που βαίνει μειούμενο οσο απομακρύνεται από το κέντρο. Η προκύπτουσα σφαίρα από την TtoST (πάνω δεξιά) και η απόκλιση της ακτίνας της από την ονομαστική (κάτω αριστερά & Πίνακας 2).*



Σχήμα 12: *Εφαρμογή της μεθόδου σε δομημένο κυβικό πλέγμα (αριστερά) με πεδίο μορφής επικαλυπτόμενων σφαιρών με τιμή μειούμενη προς τα έξω. Προκύπτουσες σφαίρες (μέση). Όταν η θέση διαχωρισμού των σφαιρών είναι σε διαφορετικά κελιά ο αλγόριθμος παράγει δύο ξεχωριστές οντότητες ενώ όταν είναι εντός κοινών κελιών ο αλγόριθμος τις ενώνει (δεξιά).*

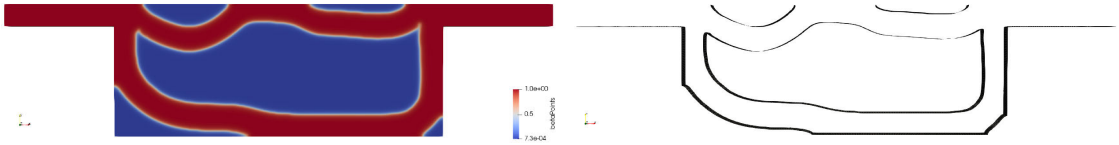
Στη συνέχεια η μέθοδος χρησιμοποιείται για την εξαγωγή τριδιάστατου μοντέλου κουνελιού [26]. Η γεωμετρία εισάγεται σε ένα υπολογιστικό χωρίο και δημιουργείται το πεδίο πορώδους. Η εφαρμογή της μεθόδου αποδίδει το μοντέλο με μεγάλη λεπτομέρεια και συγκρίνεται με το αρχικό, όπως φαίνεται στο Σχ. 13.



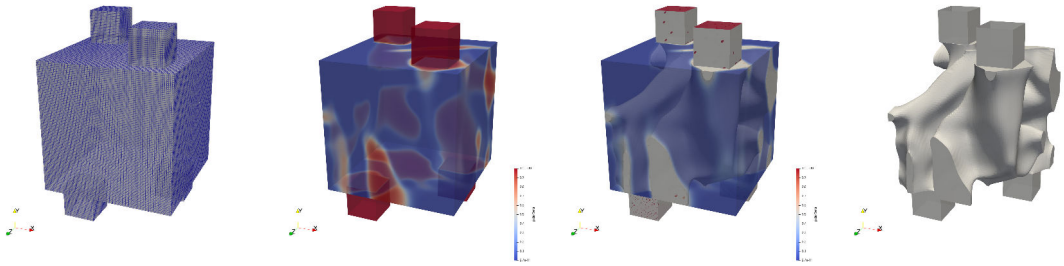
Σχήμα 13: *Εφαρμογή της μεθόδου για την εξαγωγή μοντέλου κουνελιού (πάνω αριστερά). Πορώδες υπολογιστικού χωρίου (πάνω μέση) και προκύπτον μοντέλο (πάνω δεξιά). Σύγκριση λεπτομέρειας αρχικού μοντέλου (κάτω αριστερά) με το προκύπτον (δεξία του προηγούμενου) καθώς και τομής αυτών σε μετωπιαίο επίπεδο (κάτω δεξία).*

Ακόμη, η μέθοδος εφαρμόζεται στην εξαγωγή του τοιχώματος στερεού–ρευστού (FSI) από πορώδη που έχουν προκύψει από BeTo για τον σχεδιασμό εναλλακτών θερμότητας. Στο Σχ. 14 παρατίθεται η περίπτωση διδιάστατου εναλλάκτη δύο ρευστών μιας

εισόδου και μιας εξόδου. Στο Σχ. 15, αντίστοιχα, παρατίθεται η περίπτωση τρισδιάστατου εναλλάκτη δύο ρευστών μιας εισόδου και μιας εξόδου. Σε αυτές τις περιπτώσεις παρατηρείται και η συμμετοχή του ορίου του υπολογιστικού χωρίου στο προκύπτον επιφανειακό πλέγμα.

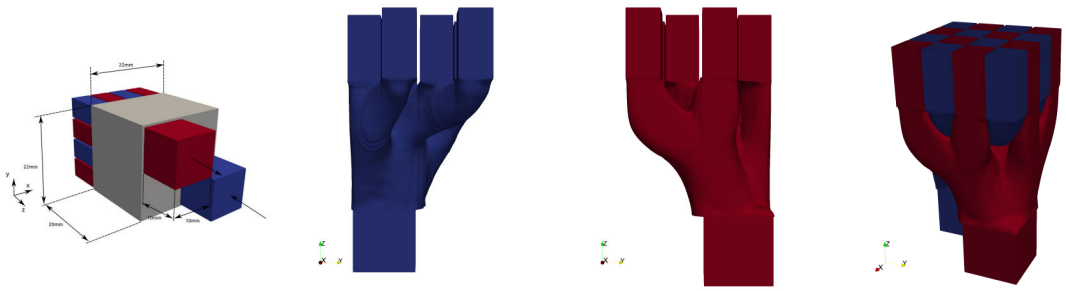


Σχήμα 14: *ΒεΤο διδιάστατου εναλλάκτη θερμότητας μιας εισόδου και μιας εξόδου. Υπολογιστικό χωρίο και πεδίο πορώδους (αριστερά), προκύπτον τοίχωμα FSI από την εφαρμογή της μεθόδου.*



Σχήμα 15: *ΒεΤο τριδιάστατου εναλλάκτη θερμότητας δύο ρευστών μιας εισόδου και μιας εξόδου για κάθε ρευστό. Υπολογιστικό χωρίο (αριστερά), πεδίο πορώδους (μέση) και προκύπτων τοίχωμα FSI από την εφαρμογή της μεθόδου.*

Τέλος η μέθοδος εφαρμόζεται για την εξαγωγή του τοιχώματος στερού–ρευστού (FSI) από πορώδες ΒεΤο για το σχεδιαμό εναλλάκτη δύο ρευστών μιας εισόδου και οκτώ εξόδων για κάθε ρευστό, που φαίνεται στο Σχ. 16, όπως προέρχεται από δημοσιευμένη ερευνητική εργασία του Ν. Γαλανού και συνεργατών [5].



Σχήμα 16: *ΒεΤο τριδιάστατου εναλλάκτη θερμότητας δύο ρευστών μιας εισόδου και οκτώ εξόδων για κάθε ρευστό. Υπολογιστικό χωρίο (αριστερά), προκύπτων τοίχωμα FSI πολλαπλής ψυχρού (μέση μπλε) και θερμού (μέση κόκκινο) ρευστού από την εφαρμογή της μεθόδου. Ολόκληρος ο εναλλάκτης (δεξιά).*

Bibliography – Βιβλιογραφία

[1] Kontoleontos, Evgenia A., Papoutsis-Kiachagias, Evangelos M., Zymaris, Alexandros S., Papadimitriou, Dimitrios I., and Giannakoglou, Kyriakos C.: *Adjoint-based constrained topology optimization for viscous flows, including heat transfer*. Engineering Optimization, 45(8):941–961, 2013. https://doi.org/10.1080/0305215X.2012.717074, 10.1080/0305215X.2012.717074.

[2] Papoutsis-Kiachagias, Evangelos M. and Giannakoglou, Kyriakos C.: *Continuous Adjoint Methods for Turbulent Flows, Applied to Shape and Topology Optimization: Industrial Applications*. Archives of Computational Methods in Engineering, 23:255–299, December 2014. 10.1007/s11831-014-9141-9.

[3] Koch, James Robert Lee, Papoutsis-Kiachagias, Evangelos M., and Giannakoglou, Kyriakos C.: *Transition from adjoint level set topology to shape optimization for 2D fluid mechanics*. Computers & Fluids, 150:123–138, 2017, ISSN 0045-7930. https://www.sciencedirect.com/science/article/pii/S0045793017301196, https://doi.org/10.1016/j.compfluid.2017.04.001.

[4] Vrionis, Panagiotis Yiannis, Samouchos, Konstantinos D., and Giannakoglou, Kyriakos C.: *Topology optimization in fluid mechanics using continuous adjoint and the cut-cell method*. Computers & Mathematics with Applications, 97:286–297, 2021, ISSN 0898-1221. https://www.sciencedirect.com/science/article/pii/S0898122121002406, https://doi.org/10.1016/j.camwa.2021.06.002.

[5] Galanos, Nikolaos, Papoutsis-Kiachagias, Evangelos M., Giannakoglou, Kyriakos C., and Yoshiteru Kondo, Koichi Tanimoto: *Synergistic Use of Adjoint-Based Topology and Shape Optimization for the Design of Bi-Fluid Heat Exchangers*. Struct. Multidiscip. Optim., 65(9), sep 2022, ISSN 1615-147X. https://doi.org/10.1007/s00158-022-03330-w, 10.1007/s00158-022-03330-w.

[6] Kontoleontos, Evgenia A.: *Designing Thermo-Fluid Systems using Gradient-based Optimization Methods and Evolutionary Algorithms*. PhD thesis, Lab. of Thermal Turbomachines, N.T.U.A., Athens, 2012.

[7] Papoutsis-Kiachagias, Evangelos M.: *Adjoint methods for turbulent flows, applied to shape and topology optimization and robust design*. PhD thesis, Lab. of Thermal Turbomachines, N.T.U.A., Athens, 2013.

[8] Karpouzas, Georgios K.: *A hybrid method for shape and topology optimization in fluid mechanics*. PhD thesis, Lab. of Thermal Turbomachines, N.T.U.A., Athens, 2019.

[9] Vrionis, Panagiotis Yiannis: *Shape and Topology Optimization using the cut-cell Method and its Continuous Adjoint for Single- and Two-phase Turbulent Flows, in a Multiprocessor Environment*. PhD thesis, Lab. of Thermal Turbomachines, N.T.U.A., Athens, 2022.

[10] Samouchos, Konstantinos D.: *The Cut-Cell Method for the Prediction of 2D/3D Flows in Complex Geometries and the Adjoint-Based Shape Optimization*. PhD thesis, Lab. of Thermal Turbomachines, N.T.U.A., Athens, 2022.

[11] Davis, L.: *Handbook of Genetic Algorithms*. VNR Computer Library VNR Computer Library. Van Nostrand Reinhold, 1991, ISBN 9780442001735. https://books.google.gr/books?id=Kl7vAAAAMAAJ.

[12] Thévenin, D. and Janiga, G.: *Optimization and Computational Fluid Dynamics*. Optimization and Computational Fluid Dynamics. Springer Berlin Heidelberg, 2008, ISBN 9783540721536. https://books.google.gr/books?id=fhV6CD25XF8C.

[13] Michalewicz, Zbigniew: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer Berlin Heidelberg, 3rd edition, 1996, ISBN 978-3-662-03315-9. https://doi.org/10.1007/978-3-662-03315-9.

[14] Arora, Jasbir S.: *Introduction to Optimum Design*. Academic Press, 2017, ISBN 978-0-12-800806-5. https://doi.org/10.1016/C2013-0-15344-5.

[15] Jameson, Antony: *Aerodynamic design via control theory*. Journal of Scientific Computing, 3(3):233–260, 1988, ISSN 1573-7691. https://doi.org/10.1007/BF01061285, 10.1007/BF01061285.

[16] Pironneau, Olivier: *Optimal Shape Design for Elliptic Systems*. Scientific Computation. Springer Berlin Heidelberg, 1st edition, 1984, ISBN 978-3-642-87724-7. https://doi.org/10.1007/978-3-642-87722-3.

[17] Papoutsis-Kiachagias, Evangelos M., Asouti, Varvara G., Giannakoglou, Kyriakos C., Gkagkas, Konstantinos, Shimokawa, S., and Itakura, E.: *Multi-point aerodynamic shape optimization of cars based on continuous adjoint*. Structural and Multidisciplinary Optimization, 59(2):675–694, Feb 2019, ISSN 1615-1488. https://doi.org/10.1007/s00158-018-2091-3, 10.1007/s00158-018-2091-3.

[18] Farhikhteh, M. Erfan, Papoutsis-Kiachagias, Evangelos M., and Giannakoglou, Kyriakos C.: *Aerodynamic shape optimization of wind turbine rotor blades*

using the continuous adjoint method. Optimization and Engineering, Nov 2023, ISSN 1573-2924. https://doi.org/10.1007/s11081-023-09868-y, 10.1007/s11081-023-09868-y.

[19] Alexandersen, Joe and Andreasen, Casper Schousboe: *A Review of Topology Optimisation for Fluid-Based Problems*. Fluids, 5(1), 2020, ISSN 2311-5521. https://www.mdpi.com/2311-5521/5/1/29, 10.3390/fluids5010029.

[20] Bendsøe, Martin Philip and Kikuchi, Noboru: *Generating optimal topologies in structural design using a homogenization method*. Computer Methods in Applied Mechanics and Engineering, 71(2):197–224, 1988, ISSN 0045-7825. https://www.sciencedirect.com/science/article/pii/0045782588900862, https://doi.org/10.1016/0045-7825(88)90086-2.

[21] Kreissl, Sebastian, Pingen, Georg, and Maute, Kurt: *Topology optimization for unsteady flow*. International Journal for Numerical Methods in Engineering, 87(13):1229–1253, 2011. https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.3151, https://doi.org/10.1002/nme.3151.

[22] Lorensen, William E. and Cline, Harvey E.: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. SIGGRAPH Comput. Graph., 21(4):163–169, aug 1987, ISSN 0097-8930. https://doi.org/10.1145/37402.37422, 10.1145/37402.37422.

[23] Bourke, Paul: *Polygonising a scalar field*, May 1994. https://paulbourke.net/geometry/polygonise/.

[24] Trompoukis, Xenofon S., Tsiakas, Konstantinos T., Asouti, Varvara G., and Giannakoglou, Kyriakos C.: *Continuous adjoint-based shape optimization of a turbomachinery stage using a 3d volumetric parameterization*. International Journal for Numerical Methods in Fluids, 95(7):1054–1075, 2023. https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.5187, https://doi.org/10.1002/fld.5187.

[25] Asouti, Varvara G., Trompoukis, Xenofon S., Kampolis, Ioannis C., and Giannakoglou, Kyriakos C.: *Unsteady cfd computations using vertex-centered finite volumes for unstructured grids on graphics processing units*. International Journal for Numerical Methods in Fluids, 67(2):232–246, 2011. https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2352, https://doi.org/10.1002/fld.2352.

[26] Stanford University, Computer Graphics Laboratory: *The Stanford 3D Scanning Repository*. https://graphics.stanford.edu/data/3Dscanrep/.

[27] Jameson, Antony and Martinelli, Luigi: *Aerodynamic shape optimization techniques based on control theory*, pages 151–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, ISBN 978-3-540-44976-8. https://doi.org/10.1007/BFb0103920.

[28] Chen, Song, Lyu, Zhoujie, Kenway, Gaetan K. W., and Martins, Joaquim R. R. A.: *Aerodynamic shape optimization of common research model wing–body–tail configuration*. Journal of Aircraft, 53(1):276–293, 2016. https://doi.org/10.2514/1.C033328, 10.2514/1.C033328.

[29] He, Zhao, Xiong, Xiaohui, Yang, Bo, and Li, Haihong: *Aerodynamic optimisation of a high-speed train head shape using an advanced hybrid surrogate-based nonlinear model representation method*. Optimization and Engineering, 23(1):59–84, Mar 2022, ISSN 1573-2924. https://doi.org/10.1007/s11081-020-09554-3, 10.1007/s11081-020-09554-3.

[30] Amstutz, Samuel and Andrä, Heiko: *A new algorithm for topology optimization using a level-set method*. Journal of Computational Physics, 216(2):573–588, 2006, ISSN 0021-9991. https://www.sciencedirect.com/science/article/pii/S0021999105005656, https://doi.org/10.1016/j.jcp.2005.12.015.

[31] Aage, Niels, Poulsen, Thomas, Gersborg, Allan, and Sigmund, Ole: *Topology optimization of large scale stokes flow problems*. Structural and Multidisciplinary Optimization, 35:175–180, February 2008. 10.1007/s00158-007-0128-0.

[32] Borrvall, Thomas and Petersson, Joakim: *Topology optimization of fluids in Stokes flow*. International Journal for Numerical Methods in Fluids, 41(1):77–107, 2003. https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.426, https://doi.org/10.1002/fld.426.

[33] Gersborg, Allan, Sigmund, Ole, and Haber, Robert: *Topology optimization of channel flow problems*. Structural and Multidisciplinary Optimization, 30:181–192, January 2005. 10.1007/s00158-004-0508-7.

[34] Othmer, Carsten: *A continuous adjoint formulation for the computation of topological and surface sensitivities of ducted flows*. International Journal for Numerical Methods in Fluids, 58(8):861–877, 2008. https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.1770, https://doi.org/10.1002/fld.1770.

[35] Deng, Yongbo, Zhang, Ping, Liu, Yongshun, Wu, Yihui, and Liu, Zhenyu: *Optimization of unsteady incompressible Navier–Stokes flows using variational level set method*. International Journal for Numerical Methods in Fluids, 71(12):1475–1493, 2013. https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.3721, https://doi.org/10.1002/fld.3721.

[36] Vrionis, Panagiotis Yiannis, Samouchos, Konstantinos D., and Giannakoglou, Kyriakos C.: *The continuous adjoint cut-cell method for shape optimization in cavitating flows*. Computers & Fluids, 224:104974, 2021, ISSN 0045-7930. https://www.sciencedirect.com/science/article/pii/S0045793021001419, https://doi.org/10.1016/j.compfluid.2021.104974.

[37] Schroeder, William J. and Martin, Kenneth M.: *1 - overview of visualization*. In Hansen, Charles D. and Johnson, Chris R. (editors):

Visualization Handbook, pages 3–35. Butterworth-Heinemann, Burlington, 2005, ISBN 978-0-12-387582-2. https://www.sciencedirect.com/science/article/pii/B9780123875822500034.

[38] Newman, Timothy S. and Yi, Hong: *A survey of the marching cubes algorithm.* Computers & Graphics, 30(5):854–879, 2006, ISSN 0097-8493. https://www.sciencedirect.com/science/article/pii/S0097849306001336, https://doi.org/10.1016/j.cag.2006.07.021.

[39] Rajon, Didier A. and Bolch, Wesley E.: *Marching cube algorithm: review and trilinear interpolation adaptation for image-based dosimetric models.* Computerized Medical Imaging and Graphics, 27(5):411–435, 2003, ISSN 0895-6111. https://www.sciencedirect.com/science/article/pii/S0895611103000326, https://doi.org/10.1016/S0895-6111(03)00032-6.

[40] Roberts, Jonathan C. and Hill, Steve: *Piecewise linear hypersurfaces using the marching cubes algorithm.* In Erbacher, Robert F., Chen, Philip C., and Wittenbrink, Craig M. (editors): *Visual Data Exploration and Analysis VI*, volume 3643, pages 170 – 181. International Society for Optics and Photonics, SPIE, 1999. https://doi.org/10.1117/12.342833.

[41] Dürst, Martin J: *Additional reference to "marching cubes"(letters).* Computer Graphics, 22(2):72–73, 1988.

[42] Nielson, Gregory M. and Hamann, Bernd: *The asymptotic decider: resolving the ambiguity in marching cubes.* In *Proceeding Visualization '91*, pages 83–91, 1991. https://doi.org/10.1109/VISUAL.1991.175782.

[43] Chernyaev, Evgueni V.: *Marching Cubes 33: Construction of topologically correct isosurfaces.* November 1995.

[44] Nielson, Gregory M., Huang, Adam, and Sylvester, Steve: *Approximating normals for marching cubes applied to locally supported isosurfaces.* In *Proceedings of the Conference on Visualization '02*, VIS '02, page 459–466, USA, 2002. IEEE Computer Society, ISBN 0780374983.

[45] Montani, Claudio, Scateni, Riccardo, and Scopigno, Roberto: *A modified look-up table for implicit disambiguation of marching cubes.* The Visual Computer, 10:353–355, June 1994. 10.1007/BF01900830.

[46] Natarajan, Balas K.: *On generating topologically consistent isosurfaces from uniform samples.* Visual Compututer, 11:52–62, January 1994. https://doi.org/10.1007/BF01900699.

[47] Gallagher, R. S. and Nagtegaal, J. C.: *An Efficient 3-D Visualization Technique for Finite Element Models and Other Coarse Volumes.* In *Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '89, page 185–194, New York, NY, USA, 1989. Association for

Computing Machinery, ISBN 0897913124. https://doi.org/10.1145/74333.74352.