



FINITE ELEMENT MESH MULTIPLICATION WITH BOUNDARY SMOOTHING

Rômulo Montalvão Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Civil, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Civil.

Orientadores: Alvaro Luiz Gayoso de Azeredo
Coutinho
Renato Nascimento Elias

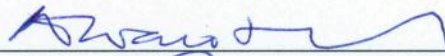
Rio de Janeiro
Fevereiro de 2019

FINITE ELEMENT MESH MULTIPLICATION WITH BOUNDARY SMOOTHING

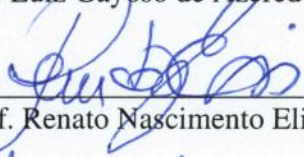
Rômulo Montalvão Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL.

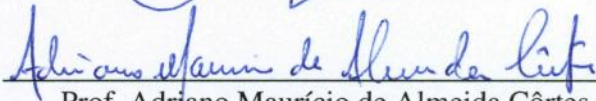
Examinada por:



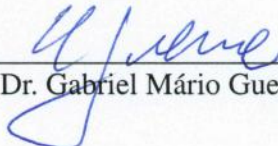
Prof. Alvaro Luiz Gayoso de Azeredo Coutinho, D.Sc.



Prof. Renato Nascimento Elias, D.Sc.



Prof. Adriano Maurício de Almeida Côrtes, D.Sc.



Dr. Gabriel Mário Guerra Bernadá, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2019

Silva, Rômulo Montalvão

Finite Element Mesh Multiplication with Boundary Smoothing/Rômulo Montalvão Silva. – Rio de Janeiro: UFRJ/COPPE, 2019.

XI, 59 p.: il.; 29, 7cm.

Orientadores: Alvaro Luiz Gayoso de Azeredo Coutinho
Renato Nascimento Elias

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Civil, 2019.

Referências Bibliográficas: p. 51 – 59.

1. Finite Element Method. 2. Boundary Smoothing.
 3. Mesh Multiplication. 4. High Performance Computing.
- I. Coutinho, Alvaro Luiz Gayoso de Azeredo *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Civil. III. Título.

Agradecimentos

Aos meus orientadores Alvaro Coutinho e Renato Elias, por toda dedicação, orientação, paciência e conversas empolgantes ao longo desta pesquisa, ajudando-me a chegar tão longe, tornando essa jornada muito mais divertida que o esperado.

Aos meus pais, Fátima e Vanivaldo, pelo amor e apoio, constantes e incondicionais.

À minha namorada, Patricia Figueiredo, por todo amor, apoio, carinho, alegria e atenção, mesmo nos momentos mais difíceis.

Aos meus amigos de longa data, em especial Luiz Eduardo, José Ewerton e José Neto. Mesmo à distância vocês me proporcionaram momentos divertidos durante essa jornada.

Aos amigos que este mestrado me trouxe, em especial Gabriel Freguglia, Lucas Teotônio e Fellipe Araujo.

Aos amigos e colegas do Núcleo Avançado de Computação de Alto Desempenho - NACAD, por terem me proporcionado um ambiente agradável durante a realização deste trabalho.

Aos membros da banca avaliadora por todas as contribuições a este trabalho.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico CNPq pelo financiamento dessa pesquisa.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

MULTIPLICAÇÃO DE MALHAS DE ELEMENTOS FINITOS COM SUAVIZAÇÃO DE SUPERFÍCIE

Rômulo Montalvão Silva

Fevereiro/2019

Orientadores: Alvaro Luiz Gayoso de Azeredo Coutinho
Renato Nascimento Elias

Programa: Engenharia Civil

Este trabalho apresenta um procedimento para melhoria da representação da superfície de malhas de elementos finitos quando refinadas por um processo de subdivisão. Inicialmente, a malha inicial é submetida a um refinamento uniforme por meio da subdivisão dos elementos, mantendo a representação inicial do contorno da geometria. Em seguida, os nós do contorno da geometria são filtrados para serem reposicionados utilizando um modelo geométrico de referência. Com o intuito de evitar buscas excessivas durante o processo de reposicionamento, é utilizada uma técnica de subdivisão espacial. Para o reposicionamento dos nós restantes (os nós internos), é utilizado um esquema simples, geralmente aplicado à problemas de interfaces móveis. Para ilustração dos resultados desse procedimento, são apresentados os estágios da geometria ao longo do processo, bem como a convergência volumétrica do sólido para cada nível de refinamento. Esse procedimento visa ajudar na geração de grandes malhas para simulações de larga escala em tempo de execução, evitando o procedimento tradicional de geração de malhas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

FINITE ELEMENT MESH MULTIPLICATION WITH BOUNDARY SMOOTHING

Rômulo Montalvão Silva

February/2019

Advisors: Alvaro Luiz Gayoso de Azeredo Coutinho

Renato Nascimento Elias

Department: Civil Engineering

A procedure to improve the boundary geometry of refined finite element meshes is presented. Initially, the coarse mesh is submitted to an uniform refinement by splitting the elements, but maintaining the coarse boundary. Then, the boundary nodes of the refined mesh are filtered. To avoid search overheads, a spatial subdivision technique is applied to the set of reference nodes. To repositionate the remaining nodes i.e., the internal nodes, a simple scheme widely used for moving boundary problems is applied. The capability of this procedure is illustrated showing the stages of the geometry during the process, as well as the mesh volumetric convergence for each level of refinement. This procedure aims to help generating meshes for large-scale problems at runtime instead of generating those meshes using traditional mesh generators.

Contents

List of Figures	ix
List of Tables	xi
1 Introduction	1
1.1 Challenges of High Performance Computing	1
1.2 Generating Unstructured Meshes for Large-Scale Simulations	4
1.2.1 Adaptive Mesh Refinement	4
1.2.2 Uniform Mesh Refinement	5
1.2.3 Communication-Avoiding Parallel Mesh Multiplication	6
1.3 Thesis Proposal and Organization	6
2 EdgeCFD: A Parallel Multiphysics Implicit Solver	8
2.1 Presentation	8
2.2 Threaded parallelism	10
2.3 Distributed Memory Parallelism	10
3 Mesh Multiplication Method	14
3.1 Communication-Free Parallel Mesh Multiplication	15
3.1.1 Communication Map	15
3.1.2 Assigning Processes to New Communication Nodes	17
4 Boundary Smoothing	20
4.1 Spatial Data Structures	21
4.1.1 Brute Force Search	21
4.1.2 BINs	21
4.1.3 kD-trees	22
4.2 Estimating the Normal Vectors at the Nodes	24
4.3 Surface Correction	25
5 Internal Nodes Repositioning	27
5.1 Setting the problem	28

5.1.1	Strong form of the problem	28
5.1.2	Weak form of the problem	28
5.2	Modified re-positioning problem	29
6	Mesh Quality Measurement	31
6.1	Mesh Quality Methods Review	31
7	Results	34
7.1	Performance Evaluation of the Communication-Free Parallel Mesh Mul- tiplication	34
7.2	Boundary Approximation	36
7.2.1	Case 01	37
7.2.2	Case 02	42
7.2.3	Case 03	46
8	Conclusions	49
	Bibliography	51

List of Figures

1.1	Roofline scheme	3
1.2	Tetrahedral split	5
2.1	Tetrahedra element matrix being disassembled into edges contributions . .	10
2.2	Master and Slave classification according to partition IDs	12
2.3	Mesh partition and GID generation	13
3.1	Regular refinement (8-subtetrahedron subdivision).	14
3.2	Pairing functions	18
3.3	Problem to assigning processes to new communication nodes (left). New partition assigned from nodes A and B (right). $A \cap B$ returns wrong partition assignment, suggesting this information should not be inherited from edges	18
3.4	Ghost Elements for partitions 0 and 1	19
4.1	BINs in two-dimensions. The black circles are the points to be searched. .	22
4.2	Distorted sphere geometry caused by a elevated number of BINs (left) and the real or target geometry (right).	23
4.3	kD-tree representation	23
4.4	Projection scheme to determine the new node's position	26
5.1	Variation of τ^e with the element's volume V^e	30
7.1	Unstructured base mesh for YF-17	35
7.2	Unstructured mesh for YF-17 at the second level of refinement	36
7.3	Mesh multiplication hotspots for 512 cores and 4 refinement levels	36
7.4	Elements Quality for each refinement level.	38
7.5	Coarse or Base mesh	39
7.6	Base and Target geometry	39
7.7	Unit normal vectors of the surface	40
7.8	Distance Field Magnitude for the 2nd Mesh Level	40
7.9	Distance Field Magnitude for the 3rd Mesh Level	41

7.10	Distance Field Magnitude for the 4th Mesh Level	41
7.11	Elements Quality for each refinement level.	42
7.12	Dragon Geometry - Base Mesh.	43
7.13	Dragon Geometry - Target Mesh.	43
7.14	Dragon's Head - Level 02 without Boundary Smoothing.	44
7.15	Dragon's Head - Level 02 with Boundary Smoothing.	45
7.16	Dragon Geometry - Distance Field.	45
7.17	Riser Geometry - Overall Mesh.	46
7.18	Strake Geometry - Target Mesh.	47
7.19	Strake Geometry - Distance Field.	47
7.20	Elements Quality for each refinement level.	48

List of Tables

3.1	Indexes combinations that provide the maximum paired values for each var size	17
6.1	A list of tetrahedron shape measures used in literature [1]	31
7.1	Time spent by refinement level (512 cores)	34
7.2	Mesh multiplication strong scalability.	35
7.3	Element quality histogram	37
7.4	Element quality degradation for one level refinement of the reference tetrahedron	37
7.5	Mesh details - Sphere Mesh	38
7.6	Volume Convergence and algorithm performance- Sphere Mesh	42
7.7	Mesh details - Dragon Mesh	44
7.8	Volume convergence and algorithm performance - Dragon Mesh	46
7.9	Mesh details - Riser Mesh	47
7.10	Volume Convergence and algorithm performance - Riser Mesh	48

Chapter 1

Introduction

The need to solve challenging problems in science and engineering often results in using tools powered by the three pillars of empirical sciences: Theoretical, experimental and computational. Keeping in mind the difficulties of doing experiments and developing new theoretical models to solve those problems, the computational sciences aim to support the other two pillars. To achieve high levels of physical fidelity and accuracy, the use of large-scale computing resources is indispensable. There are many applications where the use of these tools have become essential over the last years [2]:

- Large scale experiments in physics.
- wind-tunnel for aeronautics and automotive applications.
- water-tunnel for naval engineering applications.
- efficiency of mobile communications in telecom engineering analyses.

The solution of those problems usually relies in numerically solving PDEs, often implying in large nonlinear systems of equations. But, when the problem comes to the large scale, the generation of high fidelity models becomes a burden, specially the generation of unstructured boundary filled meshes.

The remaining sections show the current challenges of the use of High Performance Computing to run science and engineering problems (section 1.1), and do a short introduction about the generation of large unstructured meshes (section 1.2). Section 1.3 shows the proposal and organization of this work.

1.1 Challenges of High Performance Computing

Computational power has grown over the last years. Now, scientists are aiming to achieve the *Exascale Computing Era*, allowing supercomputers to get the computing capability of doing $10^{18} FLOP/s$. These machines can help us to solve more bigger and

bigger science and engineering problems. Recently, for a very specific application of deep learning in climate analytics, Kurth *et al.* [3] achieved a peak and sustained throughput of 1.13 EFlop/s and 999.0 PFlop/s respectively during the training stage of a network running in half-precision on Summit supercomputer. Notice that the half-precision floating point arithmetic differs from the industry standard which usually works with single and double precision standards. Furthermore, there are at least four major challenges difficulting us reaching to the Exascale computing [4]:

1. **Energy and Power Challenge:** maybe the most important of the four, which lies in the difficulty of develop mature technologies and combine them.
2. **Memory and Storage Challenge:** refers to the lack of available technology to store data, and to access it at desirable rates. Furthermore, some applications may have its performance bounded by memory access.
3. **Concurrency and Locality Challenge:** with the limitation of clock rates and the end of increasing single thread performance, the only mechanism to increase the performance of our applications consists in parallel programming. More and more studies try to find efficient parallel programming models to the current available hardware.
4. **Resiliency Challenge:** related to the ability of a system to continue operating even in the presence of faults or performance fluctuations.

New hardware technologies and architectures must be developed and optimized to reach the Exascale. But it is also necessary to develop new programming models for these new architectures, and look for new algorithms to extract more performance from the hardware and achieve more energetic efficiency as power became a first-order design constraint for large-scale parallel computing [5].

The more the running time of an application is increased, the more we increase the energy consumption. This way, it is obviously necessary to reduce the running time of scientific applications.

The two main cost sources of the algorithms can be summarized as:

- **Arithmetic:** quantity of floating points operations per second, $FLOPs$, and the time per each $FLOP$ operation, T_{FLOP} .
- **Communication:** the movement of data between levels of a memory hierarchy (*Moved Data / Bandwidth*) or between processors over a network (*Messages \times Latency*).

Furthermore, the total running time, RT , can be expressed as [6]:

$$RT = FLOPs \times T_{FLOP} + \frac{\text{Moved Data}}{\text{Bandwidth}} + \text{Messages} \times \text{Latency} \quad (1.1)$$

where the red part is associated to communication and the remaining part is linked to arithmetic operations.

To reduce the running time associated to the communication process, it turns attractive to use communication-avoiding algorithms. These algorithms can help us to achieve larger speedups for tasks like matrix multiplication, tensor contractions, N-body direct simulations, combustion simulation, Tall skinny QR, Unstructured Mesh Simulations and more [6].

It is expected that a computer must spend most of its computation time by doing arithmetic operations instead of changing information between processes or memory levels. Measuring only the running time is not so efficient on disclosing where or how to optimize the code. An attempt to understand the performance limitations is estimating how many data the code is sending/receiving and the number of floating points operations for a time interval. In other words, it is necessary to try to relate processor performance to memory traffic as proposed by William *et al.* [7]. This relation is called **Arithmetic Intensity**.

Figure 1.1 shows a model capable to represent together memory and floating-point performance called *Roofline model* [7]. To build such graph it is necessary to access the specifications of memory and processor, which can be found through benchmarks or looking at the hardware specifications. Some code profiling tools, like IntelTMVtune, are capable to show hotspots like entire functions or loops whose arithmetic intensity is high limited by communication.

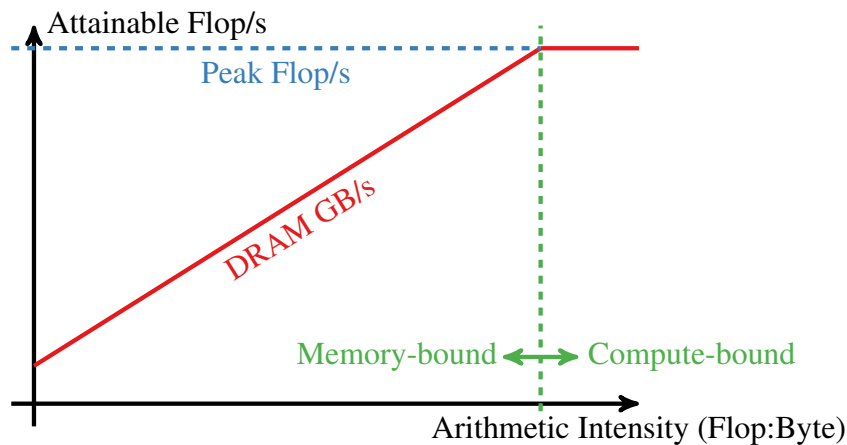


Figure 1.1: Roofline scheme

1.2 Generating Unstructured Meshes for Large-Scale Simulations

Computational scientists and engineers have now the ability to run simulations on unstructured meshes with millions or even billions of elements, as for example [8–11]. However, building such large meshes is still a cumbersome task even in today’s largest computers and parallel mesh generation represents an open and active research topic [12].

A possible solution to generate those big meshes is to refine a initial coarse mesh whose price to be generated is meaningless, and then refine the mesh using one of the well known mesh refinement methods like the Adaptive Mesh Refinement and the Uniform Mesh Refinement.

1.2.1 Adaptive Mesh Refinement

During adaptive mesh refinement, a base mesh is refined according to some metric which guides in what regions elements must be refined or coarsened [13]. In a parallel environment, this process usually leads to unbalanced mesh partitions. Elements and nodes must be exchanged among processes to recover a nearly uniform mesh distribution per processor. This operation relies on message passing, often at a high computational cost.

Mesri *et al* [14], for instance, proposes to keep parallel interfaces unchanged, during local re-meshing, and leave interface refinements for a later phase to reduce communication costs. Another common issue in parallel adaptive mesh refinement is keeping track of mesh entities shared among processes. These entities are responsible for synchronizing the solution for the whole mesh, making it consistent, and are also used during the solution phase in point-to-point (usually non-blocking) message passing [15].

Bauman and Stogner in [16] present a review of parallel adaptive software libraries. In particular they discuss MOOSE [17], FEniCS [18], deal.II [19] and libMesh [20]. In their paper, they propose GRINS, a framework built on top of libMesh to support multi-physics applications. In particular, libMesh does mesh partitioning through interfaces to several packages, including Hilbert space-filling curves and graph-based algorithms such as Metis and ParMetis [21]. GRINS and MOOSE also share the same mesh partitioning and adaptive strategies of libMesh. Currently libMesh scales to hundreds of thousands of cores [16]. Bangerth *et al.* in [22] extended the open source deal.II library functionality, offering the ability to solve finite element problems on fully adaptive meshes with billions of cells and several billion unknowns. *p4est* [23] library is used to efficiently generate and partition hierarchically refined meshes on several thousands of cores. It also provides information on the distributed nature of the mesh through a well-defined set of queries and executes directives to coarsen, refine, and re-partition the mesh. *P4est* is based on avoid-

ing global communication wherever possible in favor of gather/scatter and point-to-point operations.

1.2.2 Uniform Mesh Refinement

A different approach for quickly building large-scale parallel meshes is based on uniform element subdivision, a technique usually called mesh multiplication [10, 24, 25]. In this method, each element of the original coarse mesh is subdivided into smaller elements following a common pattern for the entire mesh. Fig. 1.2 shows how this process applies to a single tetrahedron being divided into eight smaller ones. By applying the same subdivision for all mesh elements, a new and finer mesh, eight times larger is created. The process can be applied hierarchically creating different refinement levels and could also be extended to other elements [26]. The benefit of such approach is that element creation is known a priori and follows a constant pattern.

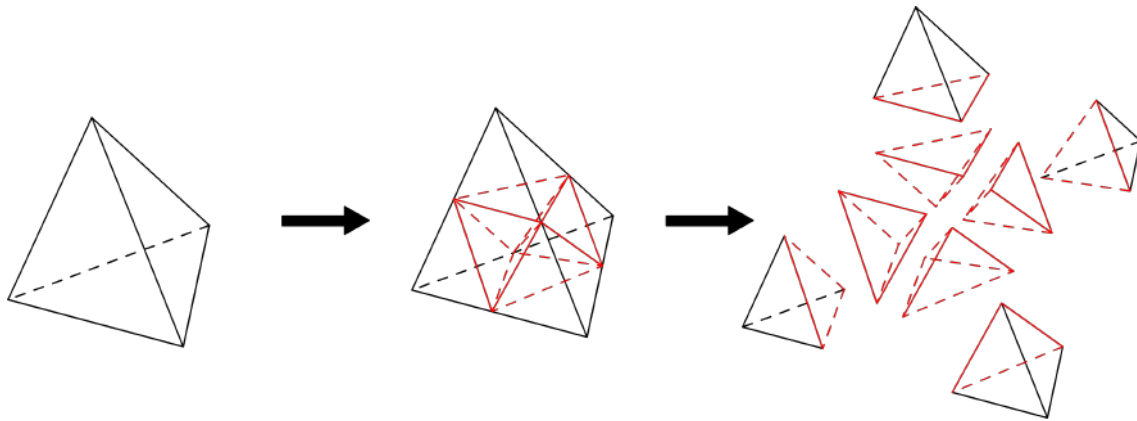


Figure 1.2: Tetrahedral split

Therefore, distributed parallel processes may apply the same algorithm independently. However, as the interface among processes is also subdivided, new shared mesh entities (nodes, edges, and element faces for instance) are created and must be accounted for in the refinement process. It is well known that the main computational cost from parallel adaptive mesh refinement comes from keeping this shared entities consistent [18–20]. These operations are usually performed with the help of a dual or nodal graph, storing the neighborhood structure of each parallel process and mesh partitions [27], and must be updated as new nodes and elements are created due to the refinement process. Moreover, in the mesh multiplication method, the finer mesh preserves the original element distribution and, consequently, the refinement pattern of the original coarser mesh.

Mesh multiplication methods found in the literature still relies on communication when the parallel interface is modified due to element refinements. In [10], the array holding nodes, that must be communicated during the solution phase, is updated to take into account new nodes created at the parallel interface. In [25], a linked list holding in-

formation about parallel edges, local and global indexes is built and communicated using specialized routines developed by the authors. Kabelikova *et al* in [9], also reports the need for communicating data across parallel interfaces to re-index global nodes.

1.2.3 Communication-Avoiding Parallel Mesh Multiplication

To overcome the performance bottleneck due to communication during the generation of big meshes, a fast tetrahedral mesh multiplication technique is implemented inside in the multiphysics finite element code, EdgeCFD, described in Chapter 2. Although general, it can be used as a standalone procedure or incorporated into an unstructured grid multiphysics solver [28]. The method is based on a uniform tetrahedral mesh refinement scheme applying recursively edge bisection (also called Bey's refinement [29]). The method is very fast and does not require any communication among processes. This is achieved by applying a particular pairing function to create unique indexes for globally shared mesh entities. The method is tested and produces unstructured meshes with billions of elements very quickly as we shall see.

1.3 Thesis Proposal and Organization

This work proposes an improvement of the mesh multiplication procedure of EdgeCFD, which at first is not capable to improve the boundary of the geometry even after the refinement. To improve the geometry representation and consequently more accurate results, the new nodes inserted during the refinement are repositioned by using a reference geometry. The secondary objectives and the structure of the remaining chapters are listed below:

- Chapter 2 describes the multiphysics solver developed at the High Performance Computing Center - NACAD-COPPE/UFRJ, in which the present contribution is implemented. Furthermore, details about its features (Section 2.1) and parallelism (Sections 2.2 and 2.3) are discussed.
- Chapter 3 shows some details about the mesh multiplication (MM) method including the introduction on how to multiply a mesh with a communication-free scheme.
- Chapter 4 presents a simple methodology to do the boundary smoothing of a given mesh using a reference geometry, and shows some data structures to do an efficient spatial search and a brief review on how to estimate the normal vectors at nodes lying over a surface mesh.

- Chapter 5 sets the problem of repositioning mesh nodes using a simple mesh diffusion scheme, showing how to formulate the problem to solve over the same mesh used to do the simulations and how to get boundary conditions from the boundary smoothing problem.
- Chapter 6 shows a review of methods to measure mesh quality, and shows the evaluation of one particular metric in the meshes produced by MM with and without boundary smoothing.
- Chapter 7 presents the results generated by applying the MM procedure with and without the boundary smoothing, and performs evaluations of mesh quality for all generated meshes.
- Finally, Chapter 8 presents the conclusions for this work, as well as some suggestions for future ones.

Chapter 2

EdgeCFD: A Parallel Multiphysics Implicit Solver

2.1 Presentation

The uniform mesh refinement process presented in this work is implemented in EdgeCFD, a multiphysics simulation tool capable to run problems such as [30–32]:

- Incompressible, Compressible and Free Surface flows.
- Transient Advection-Diffusion of multiple scalar fields.
- Fluid-Object Interaction (FOI) problems using an Arbitrary Lagrangian-Eulerian (ALE) formulation [33].
- Stabilized and Residual-Based Variational Multiscale (RBVMS) formulations for the Navier-Stokes equations (incompressible and compressible).
- SUPG formulation with discontinuity-capturing for scalar Advection-Diffusion Transport [31] and [32].
- Treatment of Turbulence by Smagorinsky’s model (static and/or dynamic) or by RBVMS as described in [34].
- Simulation of Particle-Laden flows [35].

The code is entirely written in *Fortran 90* and is built from scratch supporting hybrid parallelism. It consists of an outer time integration loop of staggered systems of equations for solving each physics: incompressible and compressible flows, mesh movement for fluid-object interaction and transport of an arbitrary number of scalar fields (also used by Volume-of-Fluid and Level Set solvers).

Time integration is a predictor-multicorrector algorithm with adaptive time stepping by a Proportional-Integral-Derivative (PID) controller (further details available in [36]). Within the flow solution loop, the multi-correction steps correspond to the Inexact-Newton method with backtracking as described in [30].

All solvers are fully implicit and require the solution of linear(ized) system of equations. For this purpose, the Generalized Minimal Residual Method (GMRES) is used for the flow and transport solvers and Preconditioned Conjugate Gradients (PCG) for mesh movement on FOI problems. Furthermore, a nodal block-diagonal and diagonal preconditioner are used respectively for flow and transport (and PCG).

Most of the computational effort spent in the solution phase is devoted to sparse matrix-vector products. To compute such operations more efficiently, it is applied an edge-based data structure as detailed in [30].

In threaded memory parallelism, EdgeCFD main loops are blocked to remove memory dependency and are parallelized by *OpenMP* directives. Distributed parallelism is reached by applying Metis or Parmetis [21], a graph partitioner, to split the original mesh into “ p ” parts, where “ p ” is usually the number of processors (cores). Shared information about nodes and edges are duplicated and synchronized by Message Passing Interface (MPI) calls. No ghost/halo information is required in EdgeCFD.

More details about the threaded and distributed memory parallelism will be given later on this section. All mesh entities (nodes, edges, elements) are reordered to improve data locality as in [37]. The mesh refinement method, proposed here, takes full advantage from the data structure implemented in EdgeCFD as new elements are successively obtained by edges bisection.

Most of the computational costs involved in our implicit solvers come from (a) formation and (b) solution of linear(ized) system of equations of the form $\mathbf{Ax} = \mathbf{b}$ where \mathbf{A} comes from the assembling of edge matrices contributions, T^e ; \mathbf{x} the vector of unknowns and \mathbf{b} the right-hand side vector, usually the residual. During the formation phase, element matrices are computed and disassembled into edge contributions before being stored as edge matrices, T^e , as shown in Fig. 2.1. During the same phase, residua are also computed and stored for the whole mesh. In phase (b), the major computational costs are due to sparse matrix–vector products, executed edge-by-edge [30]. Performance comparisons among the usual data structures in FEM for sparse matrix–vector products are given in [38] and the comparison clearly favors edge–based data structures in the case of linear tetrahedra.

For transient and nonlinear problems, several linear(ized) systems of equations must be solved until the tolerance determined by the Inexact Newton method is reached. This process is repeated for each time step. In summary, computational costs from phases (a) and (b) are directly related to refinement in space and time and the nonlinear character of the problem being solved. In order to scale up the software and keep portability, it is

chosen to follow what became a standard for modern discrete solvers and employed both threaded and distributed parallelism ([15]) as described in the following sub-sections.

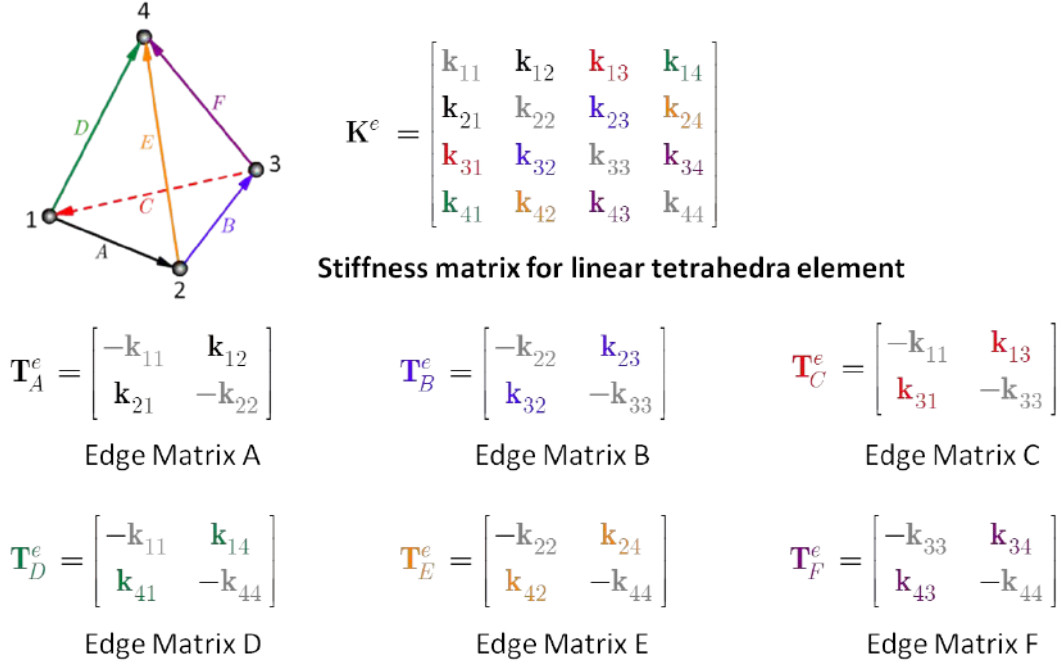


Figure 2.1: Tetrahedra element matrix being disassembled into edges contributions

2.2 Threaded parallelism

Threaded parallelism in EdgeCFD is implemented by removing memory dependency from major arrays shared by the solvers. This is done by applying a greedy algorithm to the graph of element incidences, splitting them into blocks of disjoint entities. This process is also known as *mesh coloring* [39–41] and used earlier for vectorization. Edge’s information are extracted after the mesh coloring process and is naturally safe regarding memory dependency as well. Therefore, loops involving element matrices, residua and edge-based matrix–vector computations are *blocked* and can be executed in multiple threads, using also vector instructions, if available. The outer loop iterates over each block/color while the inner loop is shared by threads in parallel.

Moreover, the granularity of inner and outer loops are controlled by the number of entities in each block and is a parameter supplied to the greedy algorithm.

2.3 Distributed Memory Parallelism

It is well known that the size of the linear system of equations, $\mathbf{Ax} = \mathbf{b}$, obtained in the finite element method is directly associated to the refinement level of the discretization.

Therefore, a natural way of thinking in distributed memory parallelism is to break the original problem into smaller parts and associate each part to a processor (or core). This can be accomplished by using a graph partitioner library, such as Metis, which, given an input graph $G = (V, E)$, with n vertices V and m edges E , will partition G into p subsets V_1, V_2, \dots, V_p such that $|V_i| \approx n/p$ as described in [21].

The number of subsets p is usually associated to the number of processors (or cores) that will treat the problem. Moreover, the computational effort is related to the number of vertices for each subset and, for uniform parallel computers, is desired to be nearly equal, to produce a balanced workload. Note that, although partitioned, the original problem can be reassembled since $G = \bigcup G_p$. From the partitioned problem, it is needed to guarantee the equivalence with the original and global system of equations, $\mathbf{Ax} = \mathbf{b}$. This is accomplished by exchanging information among processors about common data shared through a "parallel interface" – formed by faces, edges and nodes that are assigned to more than one processor as illustrated, in 2D, by Fig2.2(a) where a mesh of quadrilateral elements is partitioned in four processes and nodes 1-10 and edges drawn in blue are shared forming the parallel interface. This task is done by calling Message Passing Interface (MPI) library routines, to deal with communication between processors.

The way this data exchange is performed is crucial for software scalability in current parallel systems. MPI provides communication models to cover different needs, from collective to point-to-point (p2p) routines, regarding the number of processors involved in the communication process; blocking, non-blocking, synchronous and asynchronous referring to how data exchange is performed.

In EdgeCFD, data synchronization is required every time a global result from matrix-vector and dot products ($L2$ norms) are computed as well as the formation of residua vector. Matrix-vector and residua computations relies on p2p non-blocking operations over nodes lying on the parallel interface. It is important to clarify how these operations are carried out, as the mesh refinement will affect the parallel interface and how processes exchange information as well.

EdgeCFD follows a similar scheme as the one described in Karanam *et al* [42], for non-blocking p2p communication, as illustrated in Fig. 2.2(a) and (b) where a 2D mesh of quadrilaterals is partitioned in four parts that will be assigned to a processor. Partitions are classified as "master" or "slaves" according to each one *id* number such that, if $i > j$ P_i will be master of P_j . Thus, P2 is classified as master of P1 regarding nodes 5 and 10 while it is a slave of P3 regarding node 7 and slave of P4, as well, for nodes 1, 4 and 9. This master-slave relationship is created for each parallel node lying on the parallel (or communication) interface defining the "communication map", and summarized by the neighborhood graph in Figure 2.2b. Filled and hollow circles in Figure 2.2a define, respectively, master and slave node images while arrows indicate the direction slave-master that the communication will take place.

Therefore, taking for instance the residua assembly for the mesh shown in Figure 2.2a, a synchronization process will be necessary after all local computations are done. The processors will send their slave nodes information to be combined by the processors classified as masters and the results will be given back to the slaves when the process finishes. A similar procedure is needed to assemble the results of the edge-by-edge sparse matrix-vector product.

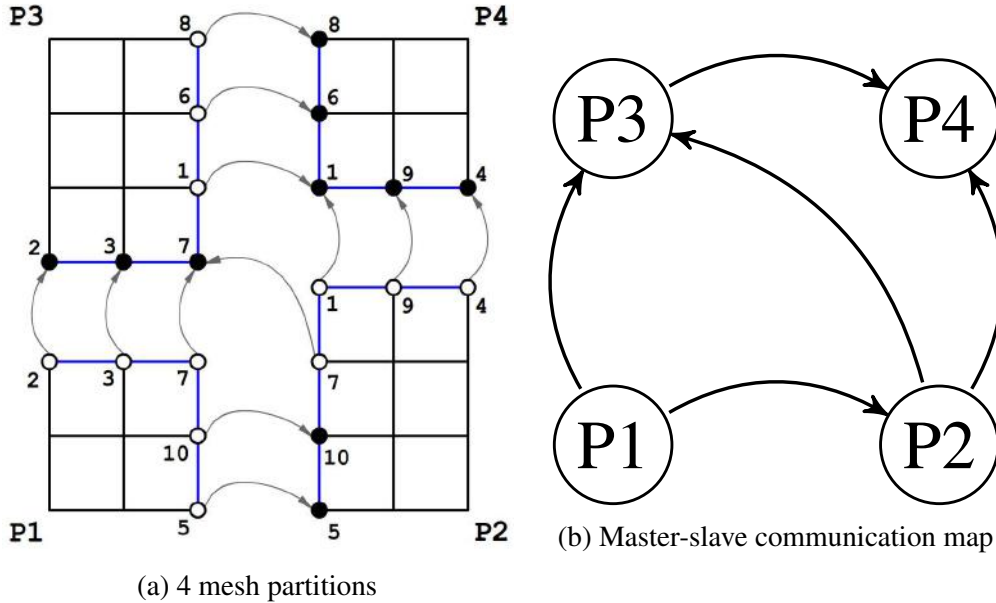


Figure 2.2: Master and Slave classification according to partition IDs

Note that the communication map must take into account any change in the parallel interface, so as the synchronization process preserves its consistency. It is also important to discuss other aspects regarding EdgeCFD's implementation before tackling the proposed mesh multiplication method. After partitioning, all mesh entities (elements, nodes, and edges) are locally renumbered and, eventually, reordered as shown in Fig. 2.3 where 6 tetrahedra are broken into 3 parts. The original global *ids* (*gid*) will be replaced by a new local numbering scheme. Moreover, the new local *ids* (*lid*) do not follow any specific rule to preserve the correspondence with the global numbering scheme given by the original mesh.

In fact, EdgeCFD can apply independently any ordering scheme to the partitions without losing global consistency. Global *ids* are preserved only for "parallel nodes" (nodes lying in the parallel interface) and are important in our mesh multiplication scheme to insert new nodes in the communication map.

The communication map in EdgeCFD is implemented as a derived data type as shown in code 2.1. Each process will store the list of *SendTo* (masters) and *GetFrom* (slaves) entities to be used during communication. To succeed, the messages sent, from one process to another must match not only in *MessageSize* but also in how data is stored in the communication buffer. In other words, taking Fig.2.2a as an example, when P1 sends

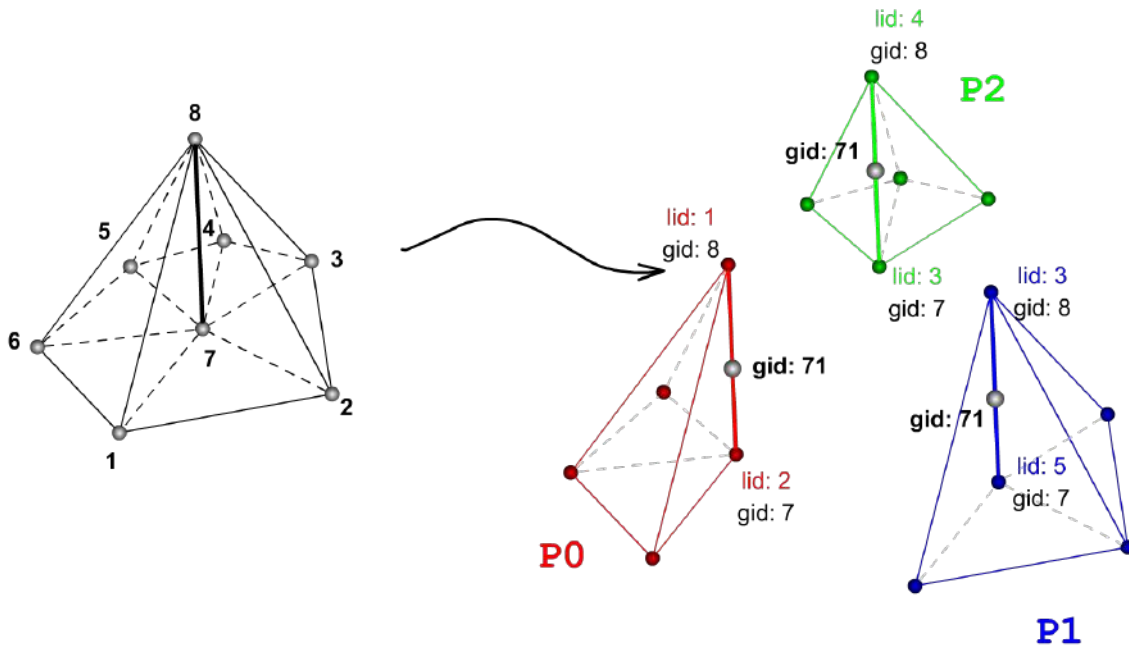


Figure 2.3: Mesh partition and GID generation

nodes 3, 4 and 5 to P2, P2 must be expecting that the data it is receiving corresponds to local nodes 4, 2 and 3, in this sequence. Therefore, EdgeCFD relies on global *ids* of parallel nodes to sort the list of data that must be sent to (or gotten from) a process when the communication map is built.

Listing 2.1: Communication map definition

```

type TMessageInfo
  integer :: NumberOfMessages           ! Number of messages (number of neighbors)
  integer :: NumberOfSharedEntities    ! Number of shared entities
  integer, allocatable :: MessageSize(:) ! Number of nodes to be sent/got per neighbor
  integer, allocatable :: Proc(:)      ! Neighbors ID
  integer, allocatable :: SharedID(:)  ! Shared entities ID
end type TMessageInfo

type(TMessageInfo), pointer :: SendNodeTo ! Data that must be sent
type(TMessageInfo), pointer :: GetNodeFrom ! Data that must be gotten

```


Chapter 3

Mesh Multiplication Method

The Mesh Multiplication method or simply MM consists on a straightforward uniform mesh refinement, which can be achieved by doing the subdivision of all elements [43]. Starting from an initial coarse mesh, the elements subdivision can be done progressively until the desired refinement level is reached. This kind of refinement is also called SUB_8 or Bey's refinement. Each triangular face of a tetrahedron T (Fig. 3.1) is refined into four sub-triangles by connecting the midpoints of the edges resulting in 8 new tetrahedra [44].

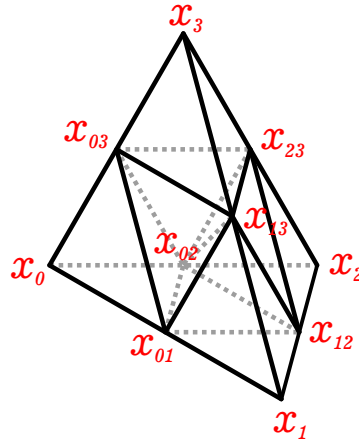


Figure 3.1: Regular refinement (8-subtetrahedron subdivision).

The nodal map for each element after the subdivision is given by

$$\mathbf{T} = \begin{cases} \mathbf{T}_1(x_0, x_{01}, x_{02}, x_{03}) \\ \mathbf{T}_2(x_{01}, x_1, x_{12}, x_{13}) \\ \mathbf{T}_3(x_{02}, x_{12}, x_2, x_{23}) \\ \mathbf{T}_4(x_{03}, x_{13}, x_{23}, x_3) \\ \mathbf{T}_5(x_{01}, x_{13}, x_{03}, x_{02}) \\ \mathbf{T}_6(x_{01}, x_{12}, x_{13}, x_{02}) \\ \mathbf{T}_7(x_{23}, x_{02}, x_{12}, x_{13}) \\ \mathbf{T}_8(x_{23}, x_{03}, x_{02}, x_{13}) \end{cases} \quad (3.1)$$

and the indices i and j are restricted to be $x_{ij} = (x_i + x_j)/2, i < j$.

This simple, but powerful method allows to decrease the bottleneck of generating bigger meshes for large-scale simulations [9].

Some authors have applied this scheme successfully, but using a more old-fashioned approach for the communication scheme [8, 10, 25, 45].

- Houzeaux *et al* (2013) [10] used the parallel mesh multiplication method inside a Navier-Stokes solver, in order to avoid serial mesh generators.
- To run problems with billions of elements, Wang *et al* (2017) [8], used schemes like parallel mesh generation, parallel surface recovery, parallel boundary updating, and parallel mesh multiplication.
- Casoni *et al* (2015) [45] used mesh multiplication to generate a mesh with 250 million tetrahedral elements on 8000 cores.
- Yilmaz & Aliabadi (2013) [25] needed to correct the boundary surface by using a master fine mesh in the CaMEL mesh format (standard format for the CaMEL flow solver). The master fine mesh was generated by the same geometry and the same generator of the coarse mesh.

3.1 Communication-Free Parallel Mesh Multiplication

3.1.1 Communication Map

Based on how the communication map is built, now, it is necessary to find a way to keep it updated and consistent while creating new nodes, due to the refinement process. Moreover, it is desirable that the proposed method avoid communication. Indeed, the proposed method does not require any communication at all. The problem is illustrated in Fig. 2.3 where new nodes are created after bisection of the parallel edge shared by three partitions.

New *lids* are freely created by each process, as previously explained, however, as these nodes are the same in the global problem, a unique *gid* must be computed by the different processes. Therefore, we must find a way to create new global *ids* that are unique across shared partitions, ideally, without having to exchange information among processors. To create these new *gids*, EdgeCFD supports the application of *pairing functions*. By definition, a pairing function is a bijection $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ where f is strictly monotone in each argument such that: $\forall i, j \in \mathbb{N} \Rightarrow f(i, j) < f(i + 1, j)$ and $f(i, j) < f(i, j + 1)$ are true [46].

Pairing bijection functions have the property to map two natural numbers onto a single one. Its inverse, f^{-1} , is called *unpairing functions*. They were made popular after Cantor's work in the second half of the 19th century on set theory [47] and are used in the theory of recursive functions, multidimensional dynamic arrays, network mapping, indexing and proximity search using space-filling curves [48–50], computational systems, among many other applications. Pairing bijection functions may be constructed to fulfill specific requirements. In Rosenberg [46] and Tarau [47] the authors discuss different methods for creating these functions. However, consider the classical Cantor pairing function, defined by: $f(i, j) = (i + j)(i + j + 1)/2 + i$, which assigns consecutive numbers to points along diagonals in the plane, as shown in Figure 3.2a, this function can be applied for building new global *ids* during the refinement process. For this purpose, global *ids* for nodes i and j of parallel edges, are paired to create a new global *id*. Note that, since Cantor's function is a bijection, the inverse operation is possible and nodes i and j could be recovered at any time by solving:

$$\begin{aligned}
 k &= f(i, j) \\
 w &= \left\lfloor \frac{\sqrt{8k + 1} - 1}{2} \right\rfloor \\
 t &= \frac{w^2 + w}{2} \\
 j &= k - t \\
 i &= w - j
 \end{aligned} \tag{3.2}$$

Although Cantor's pairing function supplies a straightforward way for building new global *ids*, it must be discussed its limitations when used in a computer program. Note that, taking two 16-bits integer numbers, ranging from 0 to $2^{16} - 1$, there will be $2^{16} \times 2^{16} - 1$ possible combinations and, from the Pigeonhole principle, an output of size $2^{16} \times (2^{16} - 1)$ which equals to $2^{32} \times 2^{16}$ will be necessary. In other words, a successful pairing function from two 16-bits integer would require, at least, a 32-bits integer as output ¹. Therefore, for practical purposes in large-scale simulations, a good pairing function should be capable of mapping the largest index values possible within the range of the output number representation. Now, let's check this assumption for Cantor's pairing function. For this purpose, consider the pair $\langle 65535; 65535 \rangle$, of the largest unsigned 16-bit integers. It returns 8589803520 when paired by Cantor's function, which is a number greater than the largest unsigned 32-bits integer (2^{32}) and, consequently, limits its use for our purposes. To overcome this limitation, the Szudzik's or "Elegant" pairing

¹this discussion originally initiated at <http://stackoverflow.com/questions/919612/mapping-two-integers-to-one-in-a-unique-and-deterministic-way>

(Figure 3.2b) bijection presented in [51] was adopted and is expressed by

$$f(i, j) = \begin{cases} j^2 + i & \text{if } i < j \\ i^2 + i + j & \text{if } i \geq j \end{cases}. \quad (3.3)$$

The Unpair function for the Elegant scheme is given by

$$\begin{aligned} k &= f(i, j) \\ s_1 &= \lfloor \sqrt{k} \rfloor \\ s_2 &= s_1^2 \\ i &= \begin{cases} s_1 & \text{if } k - s_2 \geq s_1 \\ k - s_2 & \text{if } k - s_2 < s_1 \end{cases} \\ j &= \begin{cases} k - s_2 - s_1 & \text{if } k - s_2 \geq s_1 \\ s_1 & \text{if } k - s_2 < s_1 \end{cases} \end{aligned} \quad (3.4)$$

which for the pair of the largest 16-bit integers it returns $f(65535, 65535) = 4294967295$ that is less than the value generated by using Cantor's pairing for the same indexes. In addition, Table 3.1 shows the indexes that provide the maximum unsigned 32 and 16-bit integers using both, Cantor's and the Elegant pairing. Notice that for the 32-bit unsigned int the Elegant Unpairing returns equal maximum indexes which correspond to the maximum admissible value for a 16-bit unsigned int (the same relation remains between 16-bit and 8-bit unsigned int).

Table 3.1: Indexes combinations that provide the maximum paired values for each var size

Type	Value range	Cantor's Pairing		Elegant Pairing	
		i_{max}	j_{max}	i_{max}	j_{max}
unsigned int (16-bit)	65535	167	194	255	255
unsigned int (32-bit)	4294967295	55607	37074	65535	65535

3.1.2 Assigning Processes to New Communication Nodes

The following question expresses another common issue that may appear in mesh multiplication methods: *how to correctly assign parallel partitions to nodes created at the communication interface?* This question is illustrated in Fig 3.3, where a new node is created after bisection of edge AB . It is shown that nodes A and B are shared by

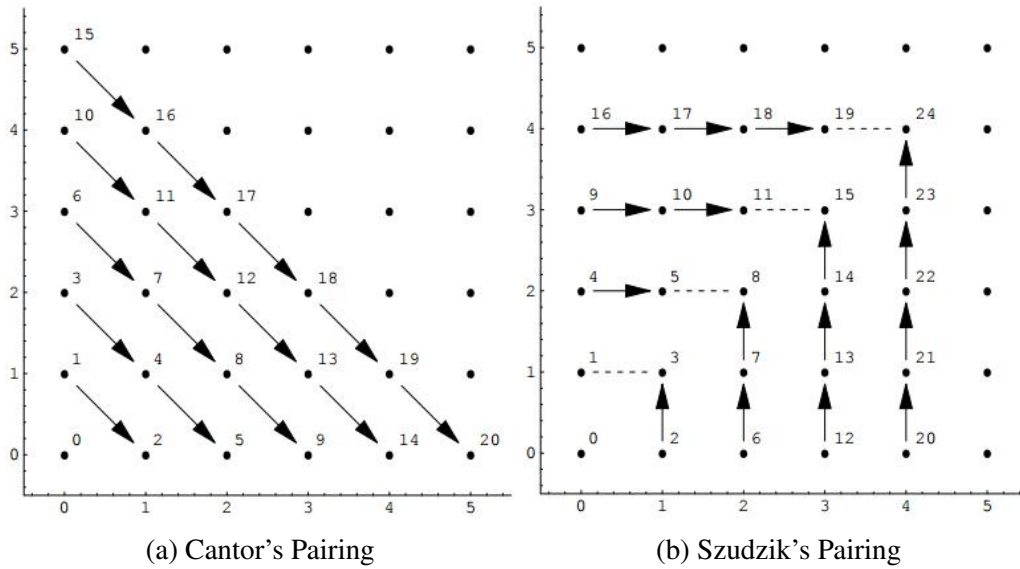


Figure 3.2: Pairing functions

partitions 0 (blue), 1 (red) and 2 (green), however, the new created node will only be part of partitions 1 and 2. We can see that this information can not be obtained from the intersection of the sets of processes shared by nodes i and j of an edge. Moreover, the sets of processes shared by edges would also lead to erroneous results since, in 3D, edges can be shared by an unknown number of elements. In other words, this information should not be built from mesh entities that have a varying number of sharing processes such as nodes and edges. Therefore, the processes are assigned to new communication nodes with the help of ghost (or halo) elements.

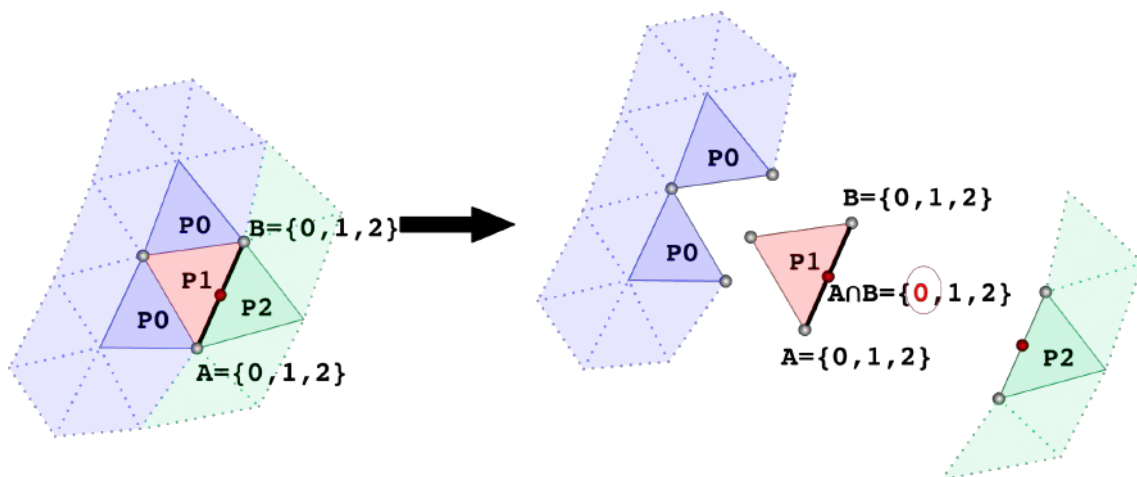


Figure 3.3: Problem to assigning processes to new communication nodes (left). New partition assigned from nodes A and B (right). $A \cap B$ returns wrong partition assignment, suggesting this information should not be inherited from edges

A ghost element for a partition i is defined by the set of elements from other partitions that share information with i and is illustrated in Fig. 3.4. Note that, for the problem of assigning processes to new parallel nodes, only elements sharing at least one edge with

partition i are required. This information is held by a data structure which is updated during the refinement process. Thus, ghost elements must also be multiplied following the same method described for regular elements.

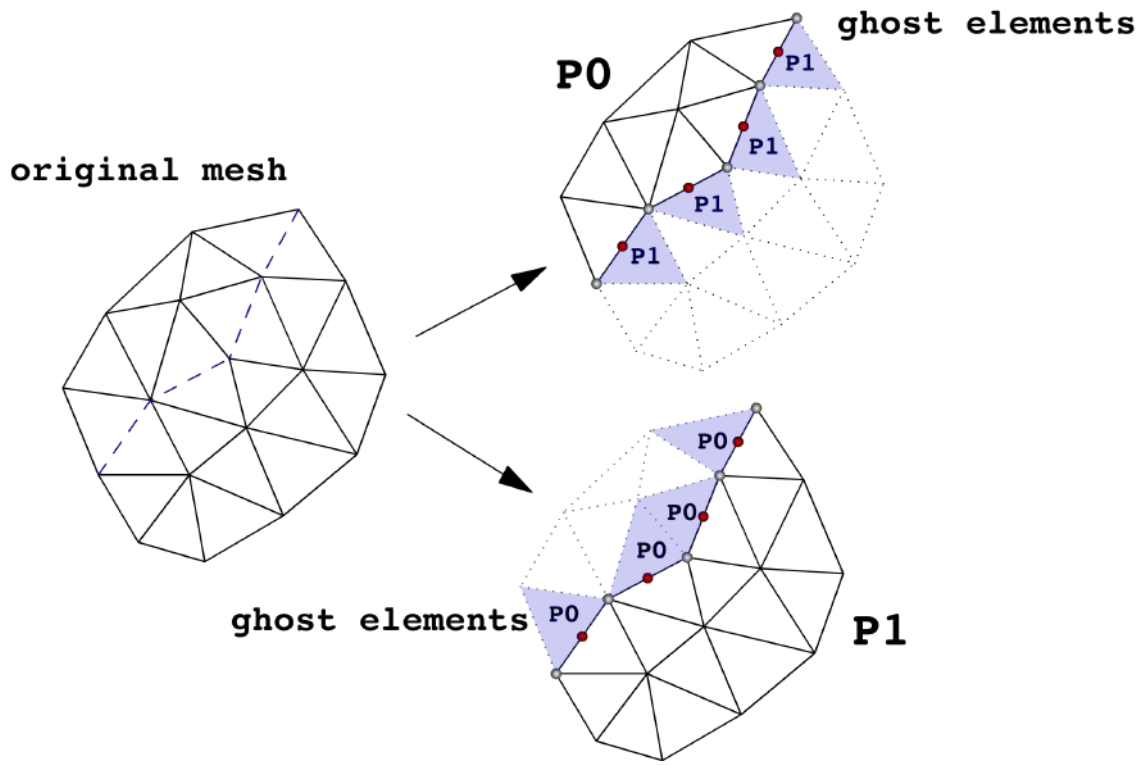


Figure 3.4: Ghost Elements for partitions 0 and 1

However, the new nodes are inserted in the edge midpoints. Consequently as we refine the mesh the geometry is not improved since at first the new nodes are not moved in order to do a better approximation of the geometry. A way of impose the geometry in the new mesh is discussed in Chapter 4 below.

Chapter 4

Boundary Smoothing

Many classical CFD problems involve the simulation of fluid flow over an object (fluid flow around a sphere, a car, aircraft and others). Since MM does not modify the geometry boundary, we need to apply a scheme in order to repositionate the new nodes created over the geometry boundary to make them more closer to the real geometry (here called *target geometry*). To do so, it is necessary to use a reliable representation of the real geometry, which consists in a discrete surface if we are running a 3D problem, and is much less expensive to generate than a complete 3D volumetric mesh. We consider that the target geometry is given in VTK file format [52].

The steps to repositionate the nodes is presented bellow:

1. **Extraction of boundary information:** It is necessary to extract the boundary faces and its respective nodes. To do so, we must use a built-in tool of EdgeCFD called `FaceTools` which is capable to handle any information associated with elements' faces.
2. **Looking for nearest nodes:** The process of finding the position of the new created nodes described in section 4.3 relies in finding the closest node that lies in the reference domain (it must be done for each new boundary node created by MM). Section 4.1 justifies why we need to use spatial data structures to do that task, and shows two alternatives to create a spatial data structure.
3. **Estimating normal vectors at the nodes:** Section 4.2 indicates how to estimate the normal vectors at the boundary nodes that are needed to do the procedure in section 4.3. This task also depends on `FaceTools`.
4. **Setting the nodal displacement for boundary nodes:** Section 4.3 shows how to use the nearest VTK node information and the normal vector at the boundary nodes to set the nodal displacement as presented in Figure 4.4.

4.1 Spatial Data Structures

Many computational procedures are dependent of a fundamental intrinsic operation called *searching*. This operation consists in retrieving some desirable information from a large amount of data. Usually, the data is partitioned into records, in a way that each record has its own key, which is used during the *searching* process. The goal is to find data matching with a given key [53].

Considering particularly point data (which is the case of the data used in this work), it is possible to represent it in a variety of ways, whose influence comes from the type of operations that we need to perform with the data [54].

The procedure presented in section 4.3 is extremely dependent of searching operations which incite the research about searching and k-Nearest Neighbor (kNN) methods. The following sections present three kNN methods and their brief descriptions.

4.1.1 Brute Force Search

It consists on a search for the nearest neighbor in a whole set of nodes including the most distant ones, which implies in a high complexity algorithm with a high overhead.

Given a set $\mathcal{C} \in \mathbb{R}^D$ containing n_p points, if we try to perform a search for the nearest neighbor of each node inside this set, the number of distance computations would be of order $\mathcal{O}(n_p^2)$, and the number of operations to compute each distance depends on the dimension D of the set.

The quadratic complexity of using brute force algorithms to do nearest neighbor search motivates us to use alternatives based on first building a data structure to organize the set and then, perform a much faster search for nearest neighbors like kD-trees, BINs [55–57].

4.1.2 BINs

The search for closest points does not need to be performed for points in distant regions on the domain, and Figure 4.1 shows one simple way to reduced the search overhead: a subdivision of the spatial domain using BINs [58] to classify the data inside a regular mesh of bricks with $n_{\text{sub}x} \times n_{\text{sub}y} \times n_{\text{sub}z}$ subdivisions.

The size of the bins must be

$$\begin{aligned}\Delta x &= (x_{\text{max}} - x_{\text{min}})/n_{\text{sub}x} \\ \Delta y &= (y_{\text{max}} - y_{\text{min}})/n_{\text{sub}y} \\ \Delta z &= (z_{\text{max}} - z_{\text{min}})/n_{\text{sub}z},\end{aligned}\tag{4.1}$$

where $x_{\text{min}/\text{max}}$, $y_{\text{min}/\text{max}}$ and $z_{\text{min}/\text{max}}$ denote the limits of the spatial domain.

To determine the bin into which each point falls we must compute

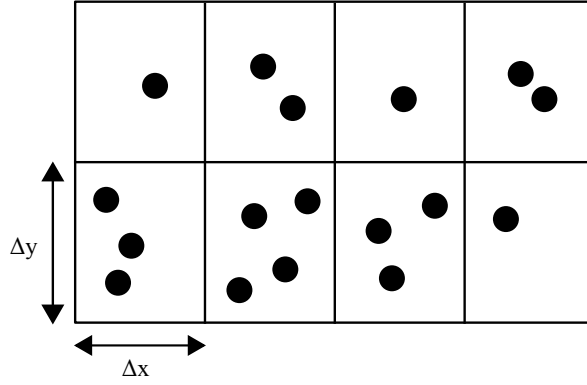


Figure 4.1: BINs in two-dimensions. The black circles are the points to be searched.

$$\begin{aligned}
 \text{isubx} &= (x_i - x_{\min})/\Delta x \\
 \text{isuby} &= (y_i - y_{\min})/\Delta y \\
 \text{isubz} &= (z_i - z_{\min})/\Delta z,
 \end{aligned} \tag{4.2}$$

and the global ID of the bin in which the point falls is given by

$$\text{ibin} = 1 + \text{isubx} + \text{nsubx} * \text{isuby} + \text{nsubx} * \text{nsuby} * \text{isubz} \tag{4.3}$$

The data produced after the domain subdivision is stored into two arrays: `lbin1(1:npoint)` and `lbin2(1:nbins+1)`, where `npoint` and `nbins` are the number of points and BINs, respectively. The `lbin1` array stores the point's *ids*, which are already ordered according to the bin into each point falls, and the `lbin2` array indicates the BINs for each interval of points inside the `lbin1` array.

The more we subdivide our spatial domain into more BINs, the more efficient are the searches. The problem about using BINs in cases that we have multiple domains (Γ and Γ_{ref} , in our case) and doing the searches only inside the BINs containing the point, is that there is the possibility to find no reference points in some BINs that we have points to be placed in their position over the real geometry, which sometimes implies in getting totally distorted geometries (Figure 4.2).

4.1.3 kD-trees

A *tree* data structure associate each node point to a number of nodes (Figure 4.3). So, unlike the linked lists the kD-tree is considered a non-linear data structure, which allows us to represent the hierarchy of a data set in a graph form [59].

The kD-tree can be used to describe space decomposition methods that proceed by recursive decomposition across a single spatial dimension containing the data until some condition is met such as that the resulting blocks contain no more than b objects (e.g.,

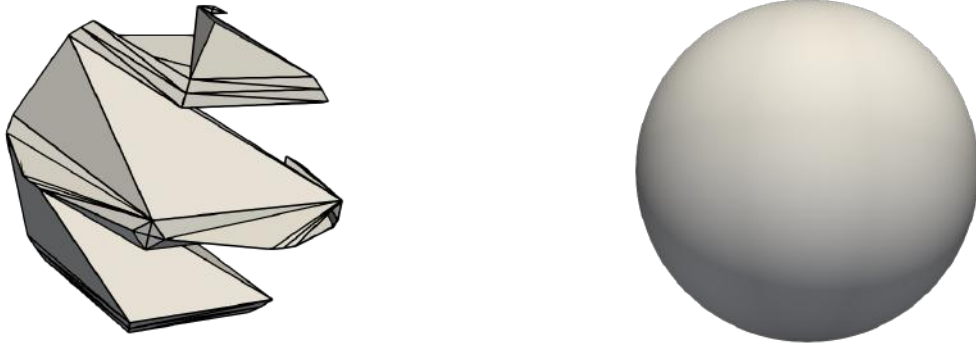


Figure 4.2: Distorted sphere geometry caused by a elevated number of BINs (left) and the real or target geometry (right).

points, line segments, etc.) or that the blocks are homogeneous. The kD-tree is usually a data structure for points which cycle through the dimensions as it decomposes the underlying space [60].

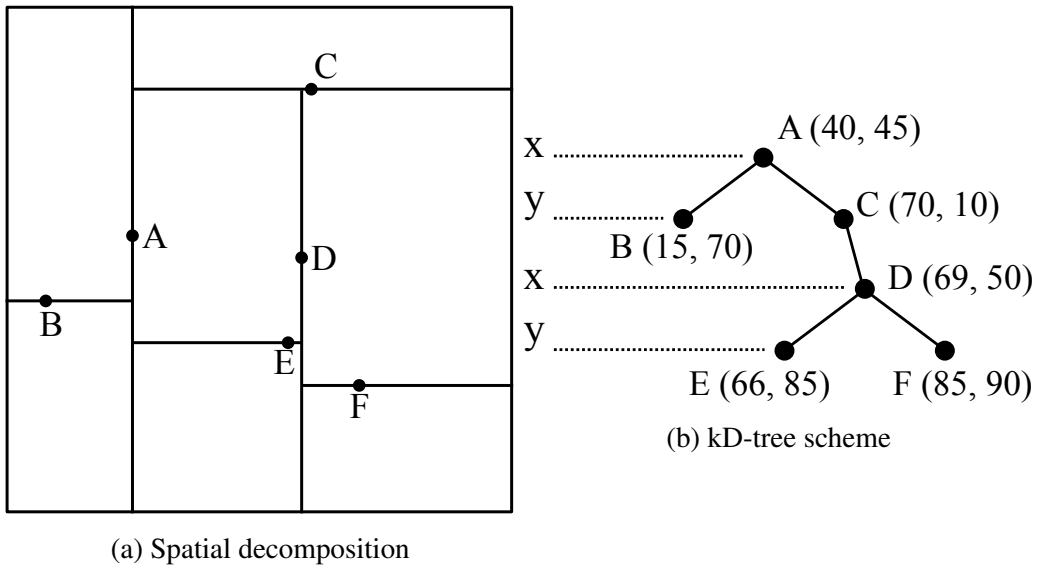


Figure 4.3: kD-tree representation

Given a point $\mathbf{x} \in \mathbb{R}^D$ that lies in $\mathcal{C} \in \mathbb{R}^D$, the kD-tree building and searching process basically consists on sorting the data base and performing a recursive binary bisection, narrowing the data base to get close to the possible neighbors of \mathbf{x} until it is possible to find the true nearest neighbors. It will usually take an average of $\mathcal{O}(\log n_p)$ bisections to locate the nearest neighbor, which is less expensive than doing an exhaustive brute force search.

The KD TREE2 package used in this work was implemented and described by Kennel (2004) [55]. It consists on a Fortran 95 module, and a set of C++ classes implementing the construction and searching routines to use the kD-trees. Some details about the building and searching process of this package are summarized bellow.

Building:

- The package assumes that the input data set is fixed and can not be changed, and builds the kD-tree structure for the whole data set.
- The *root* node corresponds to the whole data set.
- Aiming for performance, the build procedure can create a copy of the data set and permute its order to store the data contiguously in memory and achieve more CPU cache efficiency.

Searching:

- There are two search modes where the first one consists on searching for a fixed number of neighbors (find all k nearest neighbor of a given point), and the second one consists on looking for all neighbors in a given radius r (find all points \mathbf{p}_i in which $d(\mathbf{p}_i, \mathbf{x}) \leq r$).

4.2 Estimating the Normal Vectors at the Nodes

There some approaches to compute the normal vector at a boundary node of a given mesh. In general, all the methods to do that involve the computation of the normal of the faces surrounding the node, and weighting each face's normal to get the node's normal vector. The most common weighting approaches are [61, 62]:

- weighted with the surface area of each triangle,
- weighted with the inverse of the surface area,
- weighted with the angle made by the two edges connected in the node under consideration.

The use of those standard techniques to approximate the normal vectors of a node present some issues in problems like mesh generation as reported by Pirzadeh [63]. But to avoid an optimization problem [61, 62] to find the normal at a given node, a simple approach can be acceptable.

In general, the procedure of estimating the normal at a given boundary node can be described in a few ways:

1. Extract the faces of the whole mesh.
2. Select the faces of the interest region.
3. Search for nodes over the region of interest.
4. Perform an inverse mapping of the connectivity array in order to find the faces surrounding each node.
5. Compute information about the faces surrounding the nodes (usually face's normal and area).

4.3 Surface Correction

Making use of a tool called `FaceTools` already implemented in `EdgeCFD`, we are capable to get information about some geometric characteristics of a mesh, including the set of points to be re-positioned and its own normal vectors. We also need a target geometry $\Gamma_{ref} \subset \mathbb{R}^{n_{sd}}$ to guide the new nodes to the real position.

For each node over the boundary of the refined mesh, we need to perform a fast search operation to find the nearest neighbor that lies in Γ_{ref} , and compute the projection of the distance vector between these two points in the normal direction of the node of the refined geometry

$$\Delta u_{\Gamma} = \mathbf{d} \cdot \mathbf{n}, \quad (4.4)$$

where Δu_{Γ} represents the coordinate correction for a node, \mathbf{d} is the distance between the two nodes, and \mathbf{n} is the unit normal vector.

The position of a boundary node can be updated computing

$$\mathbf{x}_i = \mathbf{x}_i + \Delta u_{\Gamma} \mathbf{n} \quad (4.5)$$

There is also the possibility to apply the same scheme recursively. However, depending on the way that it is done, it could imply in computing the normal vectors for each node every time we update the nodes' position. But, even a recursive scheme will not be able to correct geometries with a too bad initial representation (which makes impossible to achieve the target geometry in regions with high curvature). Apart from this, one can conclude that it is not so effective to apply the surface correction recursively keeping in mind that there will be so much computational effort computing normal vectors for each time we apply the procedure.

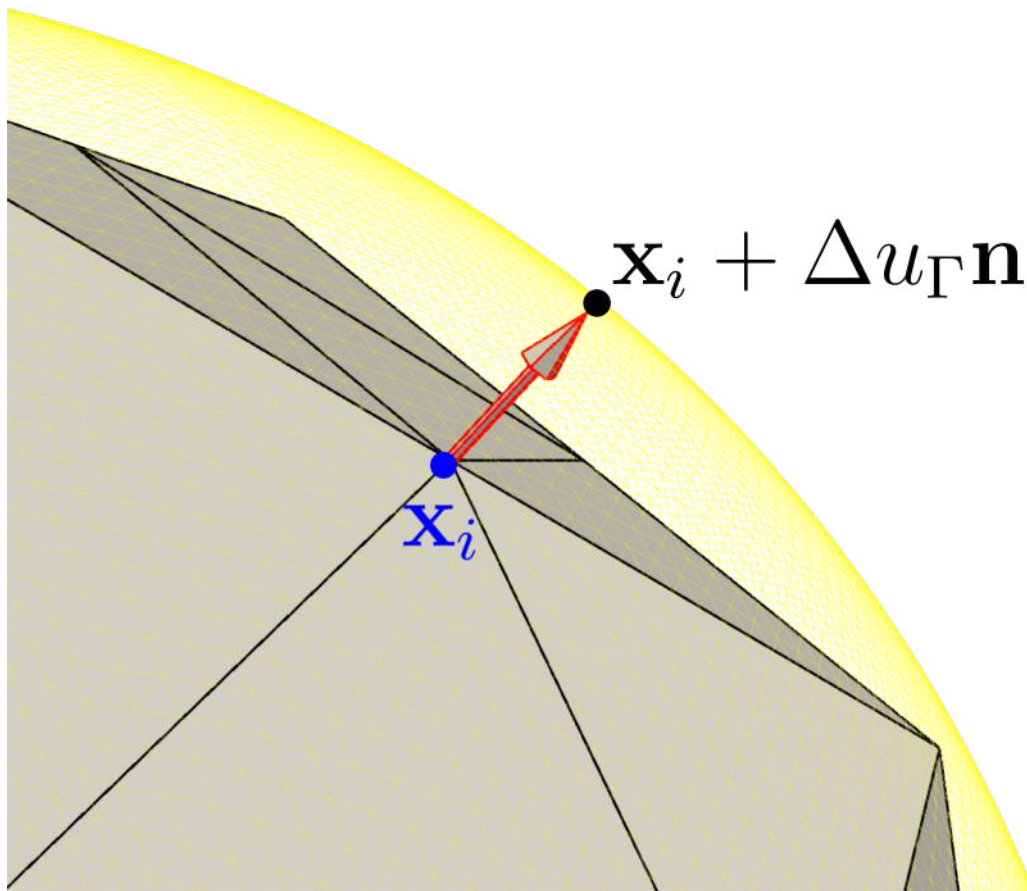


Figure 4.4: Projection scheme to determine the new node's position

Chapter 5

Internal Nodes Repositioning

There are a huge variety of discrete problems that usually require the adoption of a scheme to modify the mesh during the simulation. The main ones are compressible and incompressible flow problems with free surfaces, two-liquid interfaces, moving mechanical components, and fluid-object and fluid-particle interactions [64]. It is also important to keep in mind that the shape of the spatial domain changes with time, which turns the mesh update an expensive computation.

There are some ways to update the mesh during the simulation time. The first one, which is not the most efficient is the remeshing approach. This method try to call an automatic mesh generator during the simulation, which causes at least three disadvantages:

- The error associated to the projection of the old mesh solution to the new mesh.
- The cost of calling an automatic mesh generator at each time step to generate a new mesh from scratch.
- The difficulties to make the mesh generator work correctly while running the simulation in parallel machines.

Those disadvantages have motivated the use of mesh moving techniques built on top of elasticity [65] and diffusion models [66], which consist on solving a PDE over the same mesh to find the new position of each node. The boundary conditions come from the displacements of the moving interfaces according to the physics of the problem. Considering the boundary smoothing problem, those boundary conditions come from approaches like the one presented in chapter 4.

5.1 Setting the problem

5.1.1 Strong form of the problem

Given a mesh whose the internal nodes need to be repositioned, we can set the problem as: Let $\bar{\Omega} = \Omega \cup \Gamma$ represent the union between our domain (Ω) and boundary (Γ), and note that $\bar{\Omega} \subset \mathbb{R}^{n_{sd}}$, where $n_{sd} \geq 2$ represents the number of spatial dimensions. Let's also assume that our boundary Γ admits the decomposition bellow [66]:

$$\Gamma = \Gamma_m \cup \Gamma_f \quad (5.1)$$

and

$$\Gamma_m \cap \Gamma_f = \emptyset. \quad (5.2)$$

Here, Γ_f represents the fixed boundary (the 6 external faces of the fluid domain if we are solving a 3D fluid flow problem), while Γ_m represents the moving boundary (usually the surface nodes of the body immersed in the fluid domain).

So, given the prescribed mesh displacement \mathbf{g} at Γ_m , find the mesh displacement field $\mathbf{u} : \Omega \rightarrow \mathbb{R}^{n_{sd}}$, such that

$$\Delta \mathbf{u} = \mathbf{0} \text{ in } \Omega \quad (5.3)$$

$$\mathbf{u} = \mathbf{g} \text{ in } \Gamma_m \quad (5.4)$$

$$\mathbf{u} = \mathbf{0} \text{ in } \Gamma_f \quad (5.5)$$

Equation (5.3) is the governing equation, while Eqs. (5.4) and (5.5) are the moving and fixed boundary conditions. The linear system of equations resulting from the problem exposed here is solved by using the Preconditioned Conjugate Gradients (PCG) method with diagonal preconditioning.

5.1.2 Weak form of the problem

Applying the Galerkin's method to the weak formulation of the problem Eqs. (5.3)-(5.5), we reach to the following finite dimensional variational problem:

Find $\mathbf{u}^h \in \mathcal{S}^h$ such that

$$a(\mathbf{w}^h, \mathbf{u}^h) = \mathbf{0}, \quad (5.6)$$

for all $\mathbf{w}^h \in \mathcal{V}^h$, where

$$a(\mathbf{w}^h, \mathbf{u}^h) = \int_{\Omega} \nabla \mathbf{u}^h : \nabla \mathbf{w}^h d\Omega = 0 \quad (5.7)$$

and \mathcal{S}^h and \mathcal{V}^h are finite dimensional subspaces satisfying

$$\mathcal{S}^h \subset \mathcal{S} = \{\mathbf{u} \in H^1(\Omega) | \mathbf{u} = \mathbf{g} \text{ in } \Gamma_m \text{ and } \mathbf{u} = \mathbf{0} \text{ in } \Gamma_f\}, \quad (5.8)$$

$$\mathcal{V}^h \subset \mathcal{V} = \{\mathbf{w} \in H^1(\Omega) | \mathbf{w} = \mathbf{0} \text{ in } \Gamma\} = H_0^1(\Omega). \quad (5.9)$$

5.2 Modified re-positioning problem

Usually, some mesh regions are more refined than others due to the variations on the solution's gradient. It is reasonable to require the larger elements to absorb the great part of the distortion of the mesh, while the small elements are supposed to absorb less amounts of distortion. To do so, we can build our weighting functions to take into account the characteristics (the element's volume in this case) of each element and do a better distortion distribution [66, 67].

Playing with the aspect ratio of the elements could usually deteriorate the element's quality. The presence of highly refined mesh regions indicates that we want more accuracy of the results for those particular regions which are usually next to the moving boundary Γ_m . This way, it is possible to use the approximation proposed by Masud and Hughes [68], which for 3D elements leads to

$$\tau^e = \frac{1 - V_{min}/V_{max}}{V^e/V_{max}}, \quad (5.10)$$

where V^e is the volume of the current element, and V_{min} and V_{max} are the volumes of the smaller and bigger elements in a given mesh, respectively.

The new variational problem stated on Eq. (5.6) can be re-written as [67]

$$a(\mathbf{w}^h, \mathbf{u}^h) + \sum_{e=1}^{n_{el}} \tau^e a(\mathbf{w}^h, \mathbf{u}^h)_{\Omega^e} = \mathbf{0} \quad (5.11)$$

where n_{el} is the number of elements in the mesh.

If we try to look to the previous equation with a element-wise perspective, we could note that the new included term τ^e corresponds to an artificial stiffness or artificial diffusion, and by re-arranging the terms of Eq. (5.11) and writing it for the domain Ω^e we come up with

$$\sum_{e=1}^{n_{el}} (1 + \tau^e) a(\mathbf{w}^h, \mathbf{u}^h)_{\Omega^e} = 0 \quad (5.12)$$

Figure 5.1 shows the behavior of τ^e with the variation of V^e , for a mesh with $V_{min} = 0.1$ and $V_{max} = 1.0$. Notice that, as expected, the larger elements are less stiff than the smaller ones, and will absorb most part of the distortion caused by the mesh motion.

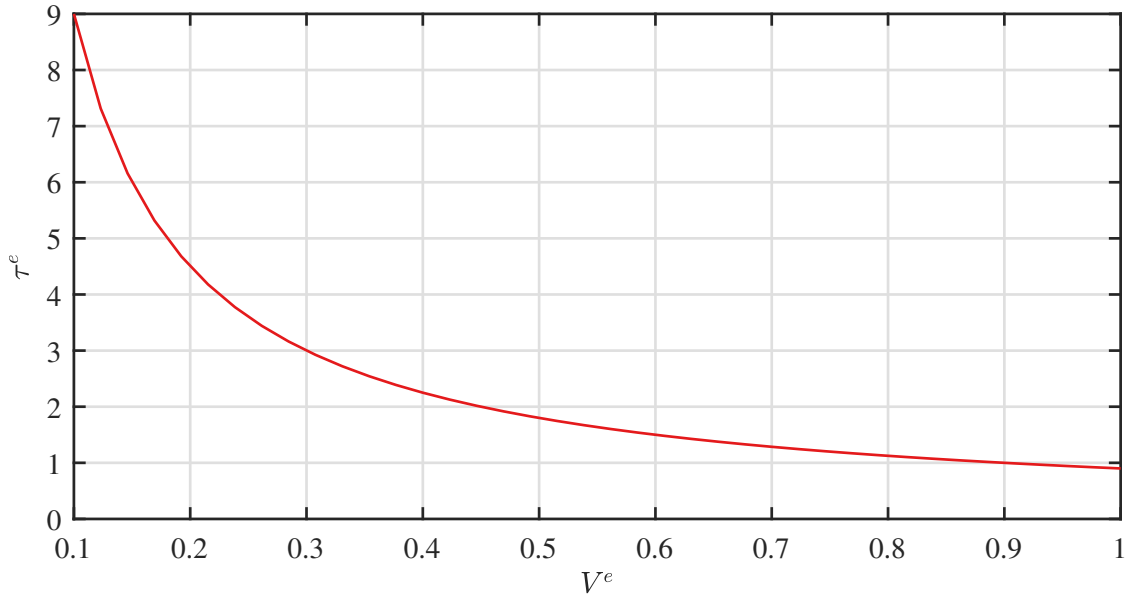


Figure 5.1: Variation of τ^e with the element's volume V^e

Fluid Mechanics problems are usually discretized by using a box to represent the fluid domain with a object immersed inside the fluid domain. Clearly, the boundary conditions to solve the variational problem will be given for the external fixed faces of the box (Γ_f) and at the body's boundary (Γ_m), where the second one is provided by the boundary smoothing scheme presented in Section 4.3.

Chapter 6

Mesh Quality Measurement

6.1 Mesh Quality Methods Review

Some mesh improvement techniques like *edge swapping*, *smoothing* and *optimization* lie in the use of quality metrics. In addition, quality constraints can also be imposed prior to the mesh creation processes, and those quality metrics can be used to do a *quality control* before passing it to a client or doing your own simulations [69], involving complex problems with irregular geometries, multi-material domains and variations of spatial scales. In addition, simulations involving moving boundaries are specially affected by cells quality resulting in loss of computational effectiveness, requiring shorter time-steps and more iterations.

Unfortunately, MM also affects the elements quality. The more the mesh is multiplied, the more it loses quality, since that for each element, the new sub-tetrahedron are worse than the original one. Table 6.1 shows a list of methods published until the 90's to measure the quality of tetrahedron elements. All those methods were compared by Parthasarathy *et al* in [1], showing the advantage of methods using the root mean square of the edge lengths and volume, whose computational effort is less expensive to do the assessment of element quality.

Table 6.1: A list of tetrahedron shape measures used in literature [1]

Aspect ratio measure	Value for a equilateral tetrahedron	Used in
$\beta = \frac{CR}{LR}$	$\beta^* = 3.0$	[70]
$\sigma = \frac{S_{max}}{LR}$	$\sigma^* = 4.898979$	[71]
$\omega = \frac{LR}{CR}$	$\omega^* = 0.612507$	[71]
$\tau = \frac{S_{max}}{S_{min}}$	$\tau^* = 1.0$	[71]
$\kappa = \frac{V^4}{[\sum_{i=1}^4 SA^2]^3}$	$\kappa^* = 4.58457E - 04$	[72]
$\alpha = \frac{S_{avg}^3}{V}$	$\alpha^* = 8.479670$	[73]
$\gamma = \frac{S_{rms}^3}{V}$	$\gamma^* = 8.479670$	[74]

In Table 6.1, CR is the radius of the circumscribed sphere, IR the radius of the inscribed sphere, S_i the length of any edge i , $S_{max} = \max(S_i)$ ($i = 1, \dots, 6$), $S_{min} = \min(S_i)$ ($i = 1, \dots, 6$), SA the surface area of a triangular facet, $S_{avg} = \text{average}(S_i)$ ($i = 1, \dots, 6$), $S_{rms} = \text{root mean square}(S_i)$ ($i = 1, \dots, 6$), V the volume of the tetrahedron, and

$$V = \frac{1}{6} \begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix}, IR = \frac{4V}{[\sum_{i=1}^6 SA]}, S_{rms} = \sqrt{\frac{1}{6} \sum_{i=1}^6 S_i^2} \quad (6.1)$$

There are also other approaches that are capable to compute elements' quality for general shapes as presented by Knupp (2001) [69]. Another approach to measure the quality of general shape elements is presented by Branets & Carey [75]. Their approach give a local measurement of cell quality with an associated scalar, and it is based on mapping a reference cell and its Jacobian.

They consider a Laplace system which is a simple elliptic problem and a powerful tool for generating meshes using PDE-based methods [76]. The main characteristics of elliptic methods for generating meshes are:

1. The ability to preserve grid orthogonality near to the boundary nodes.
2. The grid smoothness over the entire domain which is inherent from elliptic problems.
3. The discontinuities over the boundary are not far propagated over the entire domain.
4. The coordinate system comes from a solution of a PDEs' system whose computational cost to solve is greater than other methods of mesh generation.

One simple Laplace system can be defined as

$$\Delta x^i = 0 \quad i = 1, \dots, n_{sd} \quad (6.2)$$

which can be more conveniently represented in a non-convex physical domain

$$\Delta \xi^i = 0 \quad i = 1, \dots, n_{sd} \quad (6.3)$$

Both systems represent the Euler-Lagrange equations that minimize the integral

$$\mathcal{I} = \int_{\hat{\Omega}} \sum_{i=1}^{n_{sd}} \|\nabla \xi^i\|^2 d\hat{\Omega} \quad (6.4)$$

where $\|\nabla \xi^i\|$ is the grid point density along a line for a variation of ξ^i ($\xi^1 = \xi, \xi^2 = \eta, \dots$).

It is also possible to represent Equation (6.4) in a compact way

$$\int_{\hat{\Omega}} \frac{\text{tr}(\mathbf{S}^T \mathbf{S})}{\det \mathbf{S}} d\hat{\Omega} \quad (6.5)$$

where tr denotes the trace and \mathbf{S} is the so called Jacobian matrix that maps one frame from onto the other.

Branets & Carey [75] follow the idea of using metric-tensor invariants or functions of the Jacobian matrix to describe grid quality, and define the *local distortion measure*

$$\beta(\mathbf{S}) = \frac{\left[\frac{1}{n_{sd}} \text{tr}(\mathbf{S}^T \mathbf{S}) \right]^{n_{sd}/2}}{\det \mathbf{S}} \quad (6.6)$$

and then, it is possible to define the *local mesh quality* as $Q_0 = \beta^{-1}$. Branets & Carey [75] show that $\beta(\mathbf{S})$ is related to cell angles (the angles between the edges of a cell), and $\beta(\mathbf{S}) = 1$ enforces more uniform cell angles which implies in well-shaped cells.

Considering the simplex element in 3D, the tetrahedron, starting from Equation (6.6) with a Jacobian matrix of a tetrahedron it leads to Equation (6.7), which allows us to estimate a cell quality by computing

$$Q_0 = \frac{72\sqrt{3}V}{\left(\sum_{i=1}^6 l_i^2\right)^{3/2}}. \quad (6.7)$$

Chapter 7

Results

7.1 Performance Evaluation of the Communication-Free Parallel Mesh Multiplication

This section evaluate the performance of the parallel mesh multiplication without communication. Consider the YF-17 aircraft model [77] shown in Fig. 7.1. The original mesh, (level 1) or "base mesh", formed by 528, 915 elements, 639, 846 edges, and 97, 104 nodes is refined up to 2.2 billions of elements in 512 cores on Stampede (a supercomputer located at Texas Advanced Computing Center - TACC in the University of Texas at Austin). It was necessary 5 minutes (wall time) to perform four multiplication levels and reach 2.2 billion elements as shown in Table 7.1, where $NElem$ is the number of elements (tetrahedra), $NNodes$ the number of nodes (vertices) and $NEdges$ the number of edges. For good visualization, it is showed in Figure 7.2 just the 2nd mesh level of a total 5.

For the sake of simplicity, this example discards any improvements on the geometry as the mesh is refined. The important issue in this scalability study is the index generation for the sub-domain boundaries and not the insertion of new points and faces to improve the object shape as the mesh is refined.

Table 7.1: Time spent by refinement level (512 cores)

Levels	NElem	NNodes	NEdges	Time (sec)
1	528,915	97,104	719,744	-
2	4,231,320	3,579,392	5,370,398	2.99
3	33,850,560	8,949,760	41,470,464	7.07
4	270,804,480	50,420,224	325,909,504	21.44
5	2,166,535,840	376,329,728	2,584,090,624	275.23

A strong scalability evaluation is performed considering 128, 256 and 512 cores and the results are listed in Table 7.2. As one may note, a superlinear speedup is achieved when using two and four times more processors for the same computational effort – to

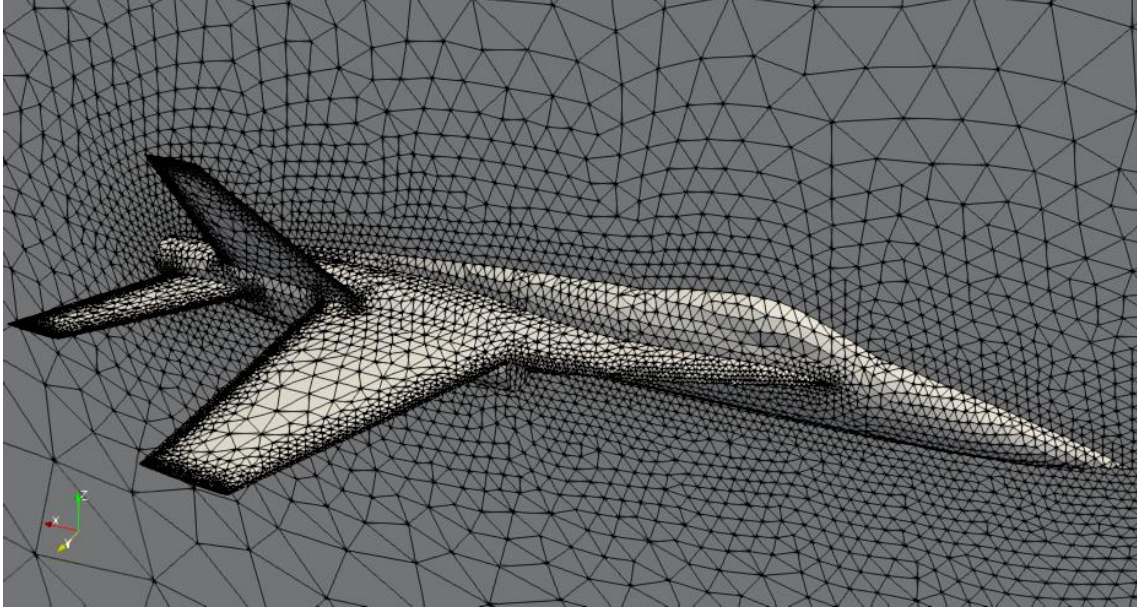


Figure 7.1: Unstructured base mesh for YF-17

perform four parallel mesh multiplication sweeps and reach 2.2 billion elements. It turns out that this may be explained by (a) cache effects (b) unnecessary communication. Thus, as the processors do not require communication, if the mesh is distributed in a larger number of cores, the partitions will be smaller and more feasible to fit in lower and faster memory cache levels.

Table 7.2: Mesh multiplication strong scalability.

Cores	NElem/Core	NNodes/Core	NEdges/Core	time (sec)	Speedup
128	16,925,280	2,940,076	20,188,208	1237.74	1.00
256	8,462,640	1,470,038	10,094,104	567.72	2.18
512	4,231,320	735,019	5,047,052	275.23	4.50

Figure 7.3 shows the evaluation of the hot spots for the proposed method when running on 512 cores. For this purpose, the mesh multiplication scheme is instrumented by TAU (Tuning Analysis Utility) tool [78]. In this figure, routines are listed by inclusive time. The top consuming time routines are `GetGlobalID` and `AddNode`, both from `ParallelNodesModule`, a Fortran90 module responsible to manage the list of parallel nodes, storing the corresponding local *ids* of nodes and neighboring information. `AddNode` is called every time a new *lid* and corresponding *gid* is created. The new node is inserted in a linked list sorted by *gid*, which justifies the time consumption due to linked list traversal. `GetGlobalID` is called when, in the final stage of mesh multiplication, a new communication map is created. The kernel of this routine consists in traverse the list of global *ids* shared by each neighboring processor. Note that no MPI calls exist within

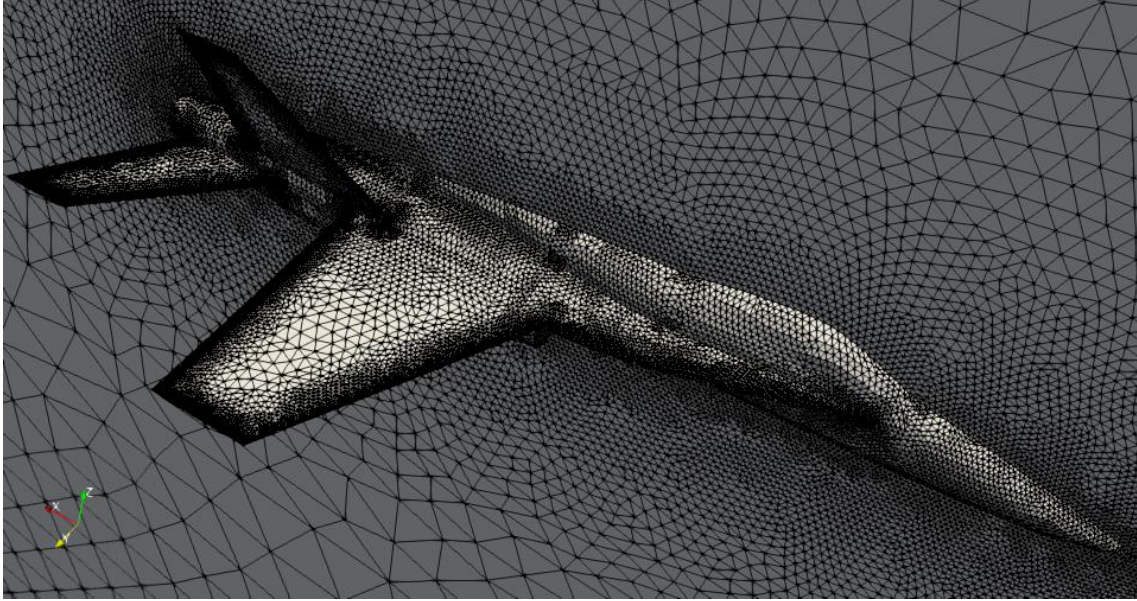


Figure 7.2: Unstructured mesh for YF-17 at the second level of refinement

MeshSubdivision module, what confirms that the proposed method is completely parallel and communication-free.

Name	Exclusive LINU...	Inclusive...	Incl...	Excl...	Excl...	Incl...	Calls	Child ...
MAIN [(main.F90) {94,1}-{195,11}]	0,009	354,802	6.625...	579,215	2.491...	97.89...	1	16
MESHMULTIPLICATIONDRIVER [(MeshMultiplication.F90) {25,1}-{24	0,056	317,91	5.897...	1.198...	1.010...	72.33...	1	23
MESHSUBDIVISION:MS_CREATEMESHSUBDIVISION [(MeshM...	0,002	215,526	5.894...	1.179...	1.554...	72.33...	1	17
MESHSUBDIVISION:MULTIPLYCOMMUNICATIONMAPNEW]	3,377	208,761	5.863...	23.931...	13.304...	60.02...	4	393.358...
PARALLELNODESMODULE:PN_GETGLOBALID [(ParallelNo...	94,678	94,678	2.065...	367.82...	2.065...	367.8...	83,581,2...	0
PARALLELNODESMODULE:PN_ADDNODE [(ParallelNo...	69,149	69,149	3.000...	183.66...	3.000...	183.6...	41,790,6...	0
FACETOOLS:FT_ISAPARALLELFACE [(FaceTools.F90) {	0,519	18,469	18.66...	1.344...	703.08...	18.86...	31,555,9...	31,555,9...
PARALLELNODESMODULE:PN_POINTTOLOCALID [(Pa...	7,236	7,236	572.8...	363.66...	572.85...	363.6...	81,189,34	0
LINKEDLISTARRAY::GETLISTITEM [(LinkedListArrayMod...	6,291	6,291	80.58...	187.43...	80.580...	187.4...	41,790,6...	0
PARALLELNODESMODULE:PN_ADDSHAREDPROCTO	2,916	5,591	50.48...	6.043...	34.818...	10.44...	41,790,6...	124.173...
FACETOOLS:FT_GETEDGESFORFACE [(FaceTools.F90	0,78	2,069	2.103...	3.324...	688.79...	4.973...	26,970,5...	80,911,5...
MESHSUBDIVISION:ASSEMBLYNEWMAP [(MeshMultipli...	0,037	0,679	27.18...	232.72...	41.262...	259.2...	8	5,525,062
LINKEDLISTARRAY::SORTEDINSERTION [(LinkedListArr...	0,675	0,675	26.13...	200.65...	26.138...	200.6...	42,565,1...	0
FACETOOLS:FT_EXTRACTFACES [(FaceTools.F90) {18;	0,441	0,441	5.944...	550,529	5.944...	651,6...	4	12

Figure 7.3: Mesh multiplication hotspots for 512 cores and 4 refinement levels

Table 7.3 shows the element quality histogram computed using the metric given by Equation 6.7. It may be observed that most of the elements have $0.4 < Q_0 \leq 0.8$ and around 10% of them have low quality and 20% high quality. The element quality degradation between two levels of refinement with respect to an arbitrary reference tetrahedron is shown in Table 7.4. We may see that the element quality deteriorates more than 25% in the worst case.

7.2 Boundary Approximation

All the following examples were executed on Lobo Carneiro Supercomputer which is a resource provided by the HPC Center at COPPE/Federal University of Rio de Janeiro. Due to the current capabilities of *FaceTools*, the algorithm needed to be executed in serial

Table 7.3: Element quality histogram

Level	$0.4 < Q_0$	$0.4 < Q_0 \leq 0.8$	$Q_0 > 0.8$	NElem
1	5,497	251,392	272,026	528,915
2	158,974	2,572,928	1,499,418	4,231,320
3	1,649,466	21,871,860	10,329,234	33,850,560
4	21,286,830	178,223,742	71,293,908	270,804,480
5	234,455,754	1,433,010,008	498,970,078	2,166,435,840

Table 7.4: Element quality degradation for one level refinement of the reference tetrahedron

Level/Tet	Q_0	Relative Q_0
1/1	0.7698	
2/1	0.7698	0%
2/2	0.7698	0%
2/3	0.7698	0%
2/4	0.7698	0%
2/5	0.6572	-15%
2/6	0.5697	-26%
2/7	0.5697	-26%
2/8	0.6572	-15%

mode. Some experiments of this code showed a tolerance of $10E - 2$ for the linear solver to be enough for the simple problem of repositioning the internal nodes. The meshes presented bellow were generated using GMSH [79] and/or ANSYS [80].

7.2.1 Case 01

The example consists on approaching the boundary of a trivial geometry, the sphere, whose volume can be exactly computed and used to test the volume convergence of increasingly refined meshes using the approach proposed in chapter 4. In this example the volume is compared for 4 mesh levels (the base mesh and three more refinement levels).

Base and Target Geometry

Figure 7.5 shows the usual mesh for a problem of fluid flow around a sphere, while Figure 7.6 shows the base or coarse geometry (Figure 7.6a) which is inside the fluid domain, and the target or reference geometry which consists on a surface mesh (Figure 7.6b).

The technique presented in chapter 4 is applied to reposition the new nodes to approach the true geometry.

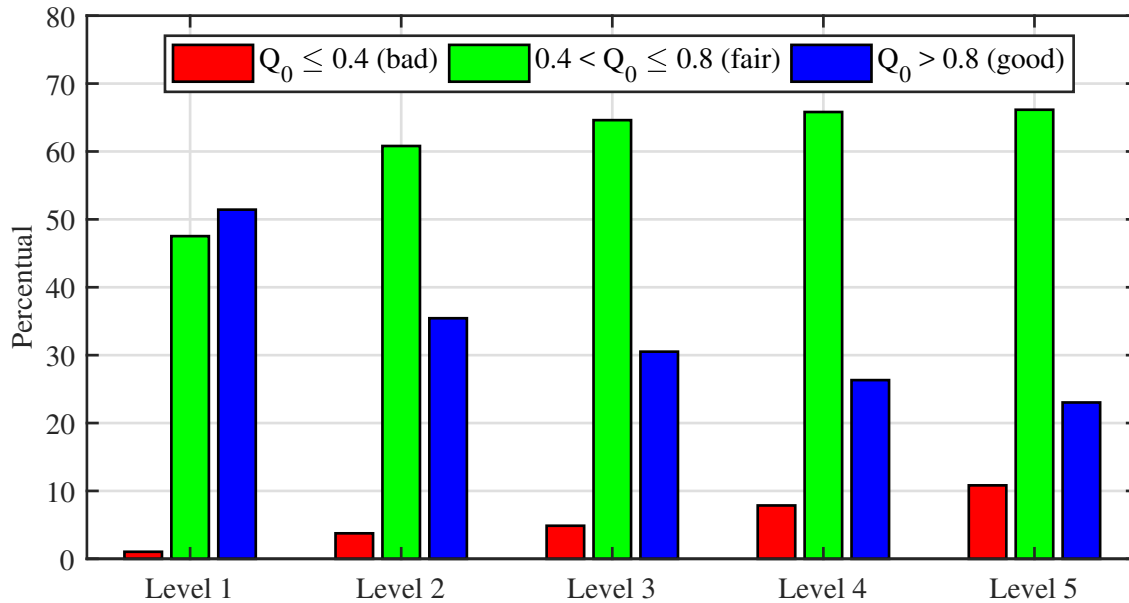


Figure 7.4: Elements Quality for each refinement level.

Mesh Details

Table 7.5 shows the number of equations that comes from the problem of reposition the internal nodes for each mesh level and their respective number of elements. Notice that, as expected, the number of elements grow with a factor of 8.

Table 7.5: Mesh details - Sphere Mesh

Mesh Level	N_{eq}	N_{el}	N_{Γ}
Reference	-	-	65538
Level 01	-	23841	18
Level 02	161886	190728	66
Level 03	1414632	1525824	258
Level 04	11770140	12206592	1026

Normal vectors to the sphere surface

We first estimate the normal vector for each node that lies over the surface (Figure 7.7) aiming to project the distance vectors from each node over the base mesh to the nearest neighbor on the refined mesh.

Distance Field for each Refinement Level

Once we determined the base mesh as Mesh Level 1, here we start to show the distance field for each level starting at level 2. Figures 7.8, 7.9 and 7.10 show the distance field (the distance that the points were moved in order to approach the geometry) for some refinement levels.

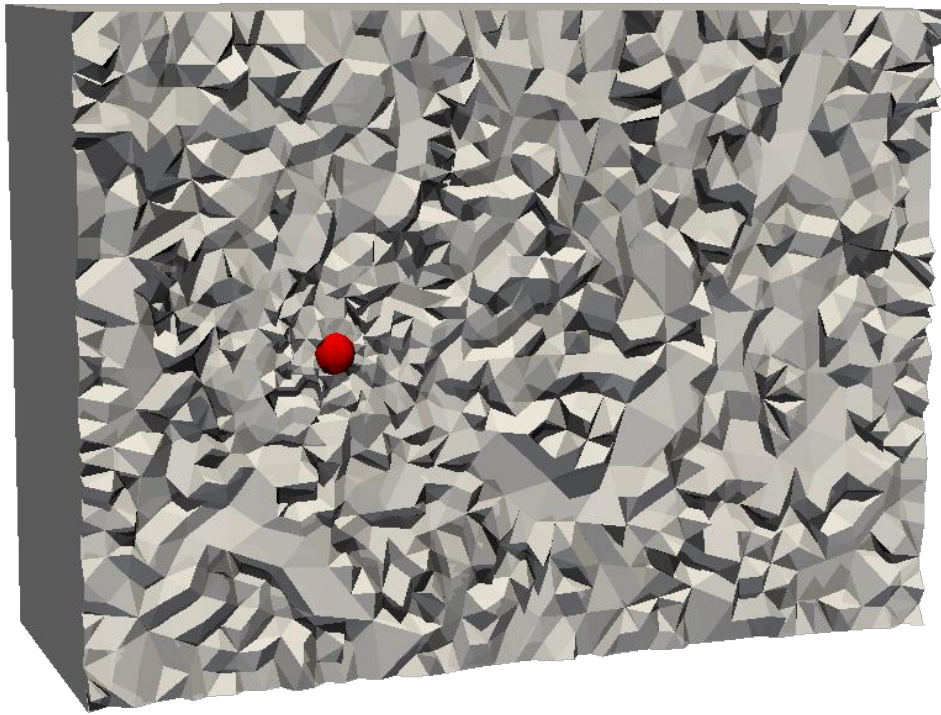


Figure 7.5: Coarse or Base mesh

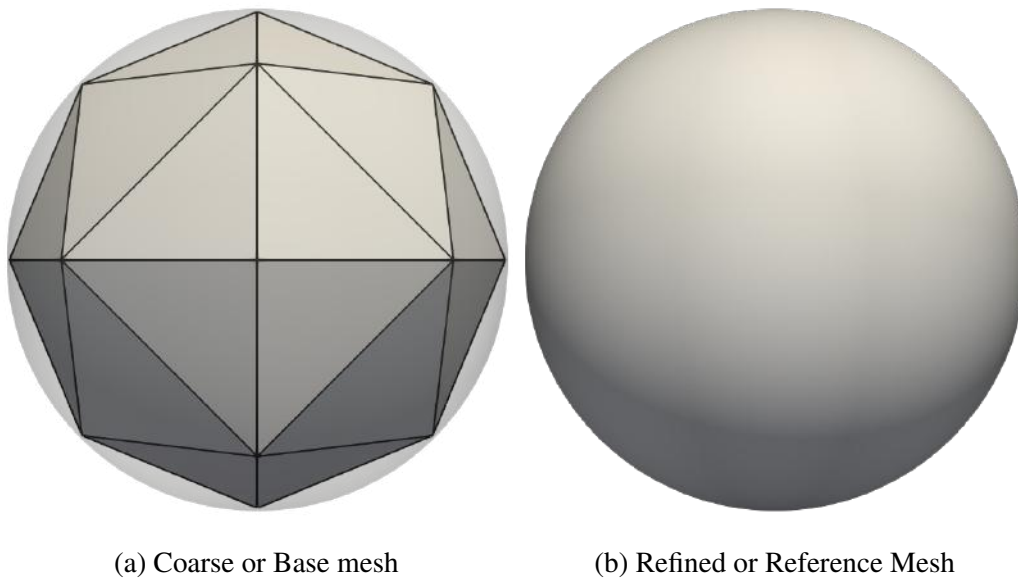


Figure 7.6: Base and Target geometry

Volume Convergence and Algorithm Performance

Table 7.6 shows the volumetric convergence of the geometry, presenting the relative error related to the real volume (the volume of the reference geometry). It is valid to highlight that without the repositioning, all levels would have the same error presented in

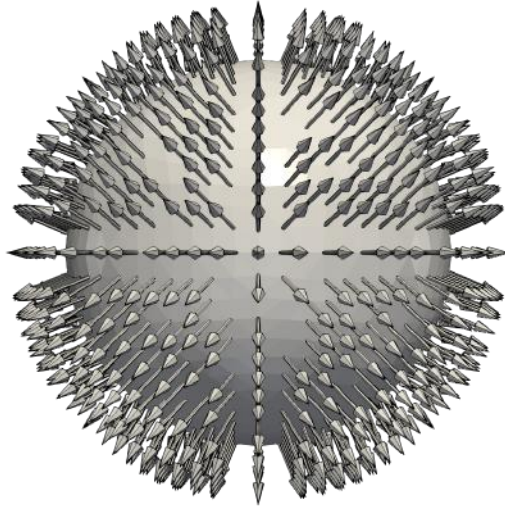


Figure 7.7: Unit normal vectors of the surface

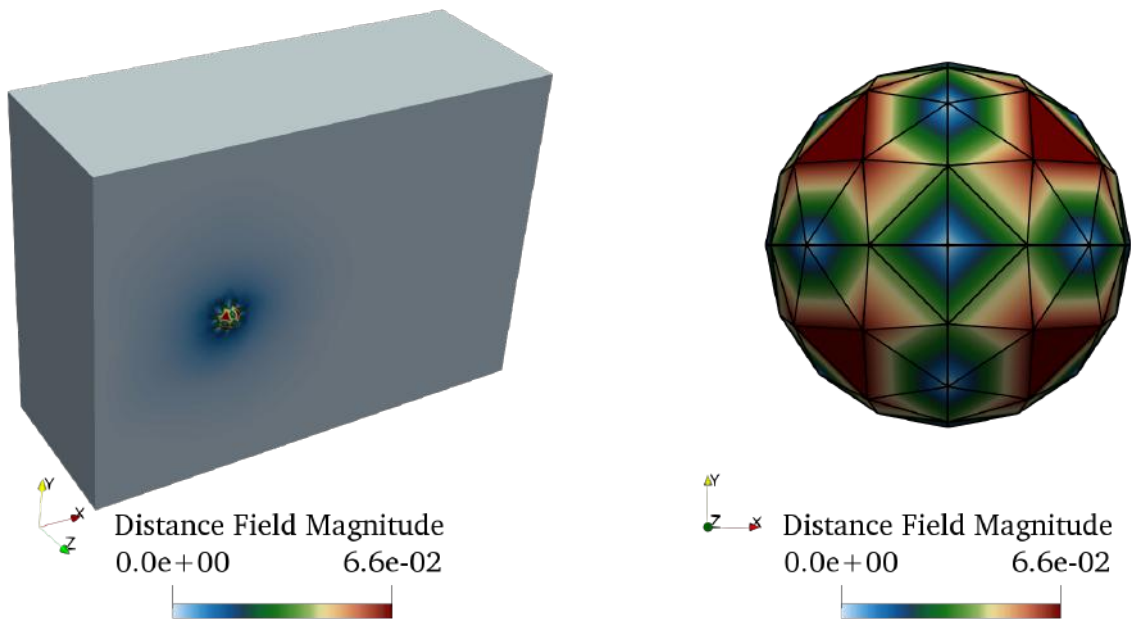


Figure 7.8: Distance Field Magnitude for the 2nd Mesh Level

level 01. The table also shows the time to repositionate the boundary nodes, SS_{time} , the time solve the linear system to repositionate the internal nodes, PCG_{time} , and the number of iterations, PCG_{it} , to achieve the prescribed tolerance.

Mesh Quality

Figure 7.11 shows the information about the elements' quality for all the four mesh levels. It is possible to see that as well as in the YF17 example, at each refinement level

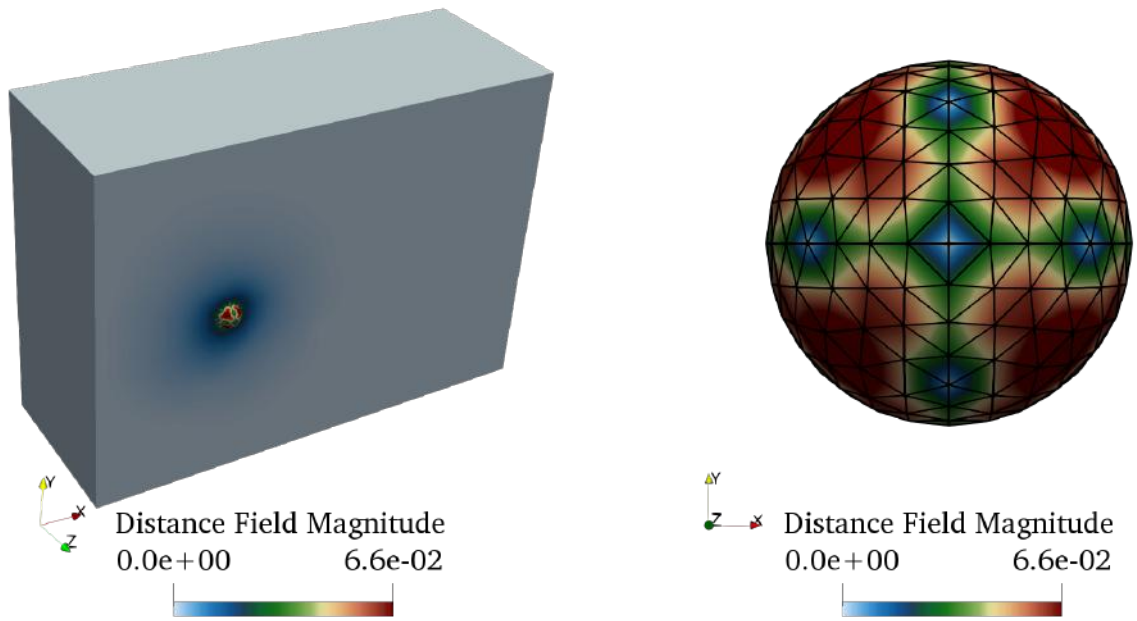


Figure 7.9: Distance Field Magnitude for the 3rd Mesh Level

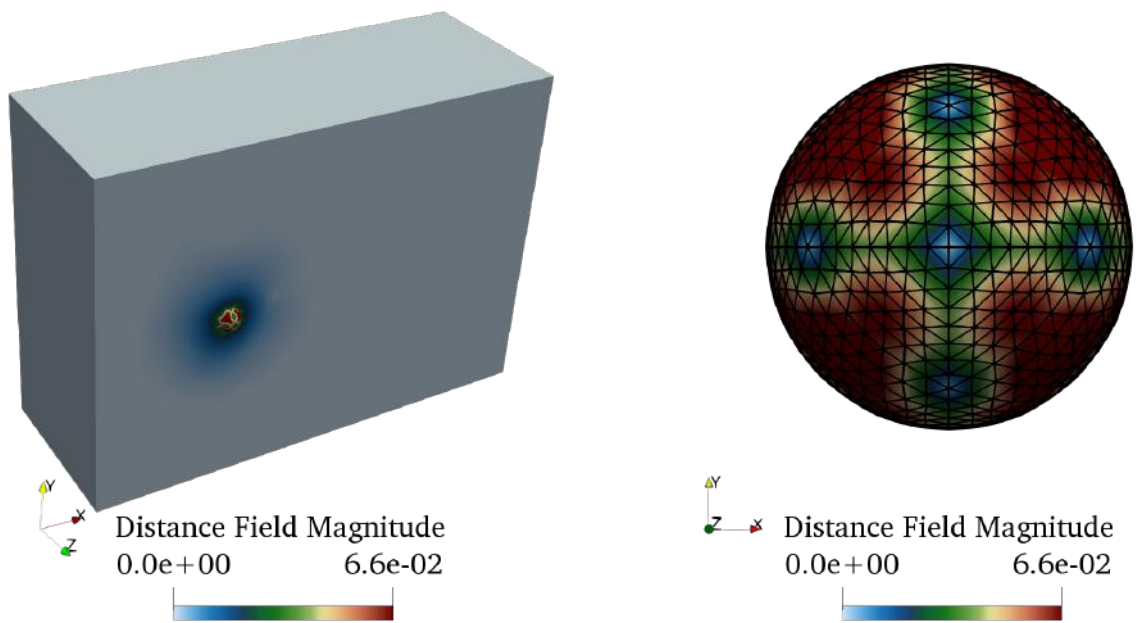


Figure 7.10: Distance Field Magnitude for the 4th Mesh Level

the quality of the elements is deteriorated.

The quality of the elements was measured before and after the surface correction and it induces no improvement of elements' quality.

Table 7.6: Volume Convergence and algorithm performance- Sphere Mesh

Mesh Level	Volume	Error (%)	$SS_{time}(s)$	$PCG_{time}(s)$	PCG_{it}
Reference	0.523551	-	-	-	-
Level 01	0.367964	29.718	-	-	-
Level 02	0.475995	9.083	$2E - 2$	$8E - 3$	4
Level 03	0.504606	3.619	0.12	$4.79E - 2$	3
Level 04	0.510612	2.471	1.584	0.252	1

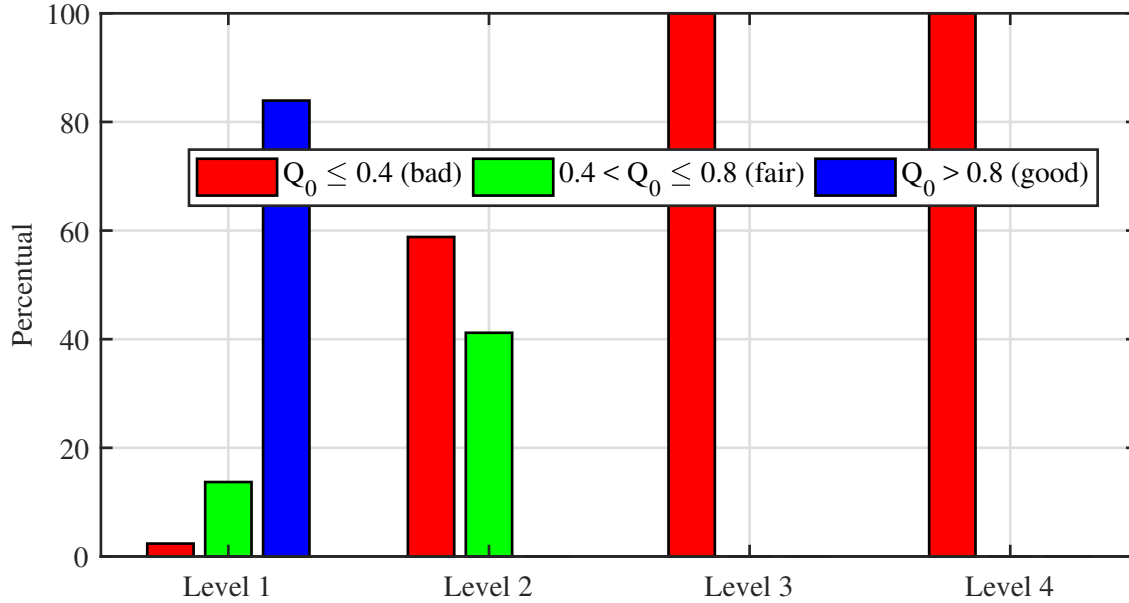


Figure 7.11: Elements Quality for each refinement level.

7.2.2 Case 02

Base and Target Geometry

This problem consists on refining and re-positioning the boundary nodes of a dragon immersed in a fluid domain. The base mesh (level 01) has 6,631,080 elements and 1,244,579 nodes.

Figure 7.12 shows the base geometry whereas Figure 7.13 shows the target geometry.

Mesh Details

Table 7.7 shows the number of equations of the system that solves the problem of repositioning the internal nodes (N_{eq}), the number of elements for the first two levels (N_{el}) and the number of nodes over the boundary surface (N_{Γ}).

Distance Field

Figures 7.14 and 7.15 show some details over the surface of the dragon's head. Figure 7.14 presents the multiplied (or refined) mesh without the application of the boundary



Figure 7.12: Dragon Geometry - Base Mesh.



Figure 7.13: Dragon Geometry - Target Mesh.

smoothing, and it is possible to see that this image is more faceted than Figure 7.14.

Figure 7.16 shows the distance field over the whole surface. In this case, instead of presenting the magnitude of the distance field, the signal is used to indicate that some

Table 7.7: Mesh details - Dragon Mesh

Mesh Level	N_{eq}	N_{el}	N_{Γ}
Reference	-	-	412746
Level 01	-	828885	71281
Level 02	5714094	6631080	285148

nodes were moved in the same direction of the external normal (cold colors) while other nodes are moved in the opposite direction (hot colors).

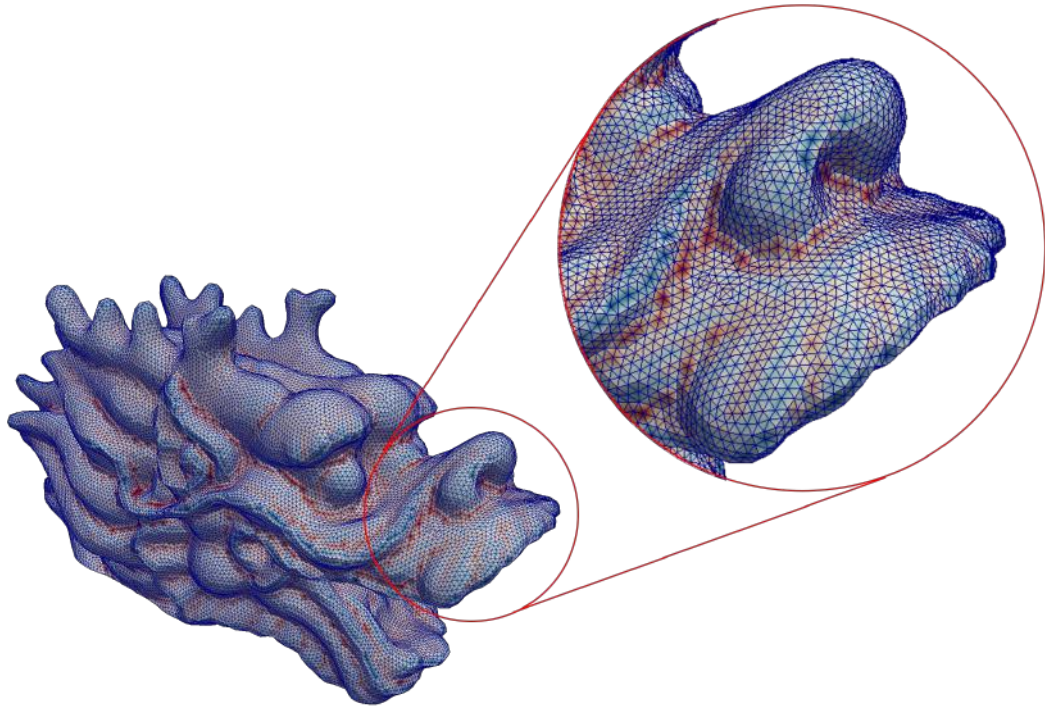


Figure 7.14: Dragon's Head - Level 02 without Boundary Smoothing.

Volume convergence and algorithm performance

Table 7.8 shows the volumetric convergence of the geometry, presenting the relative error related to the real volume (the volume of the reference geometry). The table also shows the time to repositionate the boundary nodes SS_{time} , the time solve the linear system to repositionate the internal nodes, PCG_{time} , and the number of iterations, PCG_{it} to achieve the prescribed tolerance.

Mesh Quality

The base dragon mesh already had a mesh with excellent quality whose almost all elements presented $Q_0 > 0.8$. Even applying the MM algorithm, the mesh quality had suffered almost no deterioration. This observation could indicate that for a mesh with

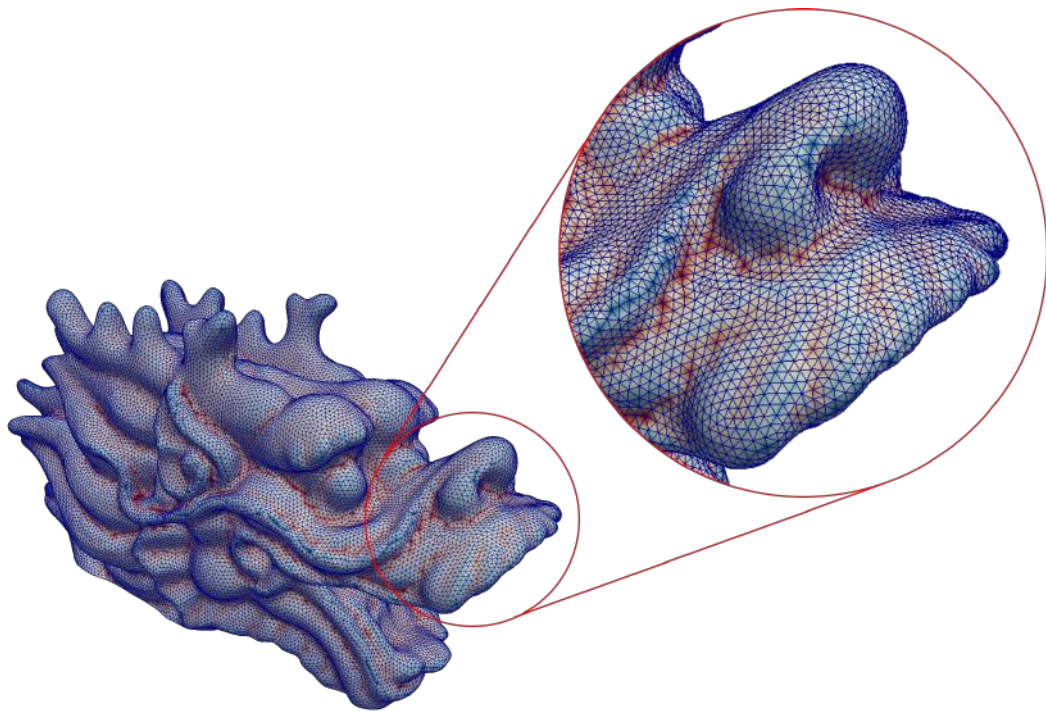


Figure 7.15: Dragon's Head - Level 02 with Boundary Smoothing.



Figure 7.16: Dragon Geometry - Distance Field.

sufficient good quality MM does not affect the mesh quality at all (it is discussed at chapter 8).

Table 7.8: Volume convergence and algorithm performance - Dragon Mesh

Mesh Level	Volume ($u.v.$)	Error (%)	$SS_{time}(s)$	$PCG_{time}(s)$	PCG_{it}
Reference	4114567424	-	-	-	-
Level 01	4109478656	0.124	-	-	-
Level 02	4113457920	0.027	875.64	0.4	4

7.2.3 Case 03

Base and Target Geometry

This problem consists on refining and re-positioning the boundary nodes of an improved riser geometry [81] immersed in a fluid domain. The base mesh (level 01) has 2291547 elements. Figure 7.17 shows the geometry immersed in the fluid domain, whereas Figure 7.18 shows the target geometry.

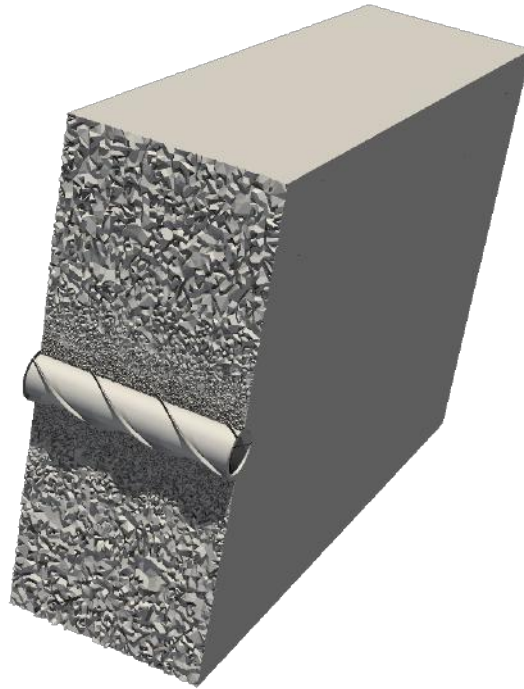


Figure 7.17: Riser Geometry - Overall Mesh.

Mesh Details

Table 7.9 shows the number of equations of the system that solves the problem of repositioning the internal nodes (N_{eq}), the number of elements for the first two levels (N_{el}) and the number of nodes over the boundary surface (N_{Γ}).

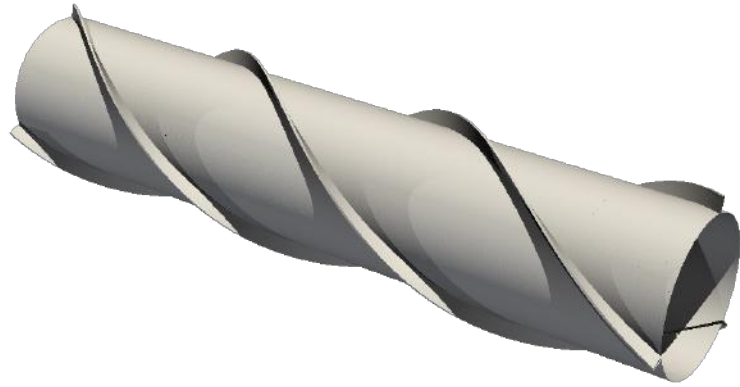


Figure 7.18: Strake Geometry - Target Mesh.

Table 7.9: Mesh details - Riser Mesh

Mesh Level	N_{eq}	N_{el}	N_{Γ}
Reference	-	-	5398848
Level 01	-	2291547	34209
Level 02	17042664	18332376	136458

Distance Field

Figure 7.19 shows the distance field over the whole surface. Since the surface mesh follows a pattern (it is a structured mesh), we can see that the distance field in Figure 7.19 also follows a pattern, which is expected from this case.

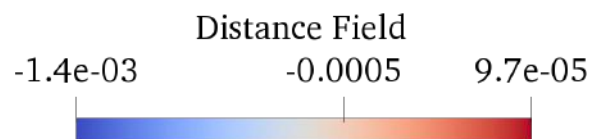
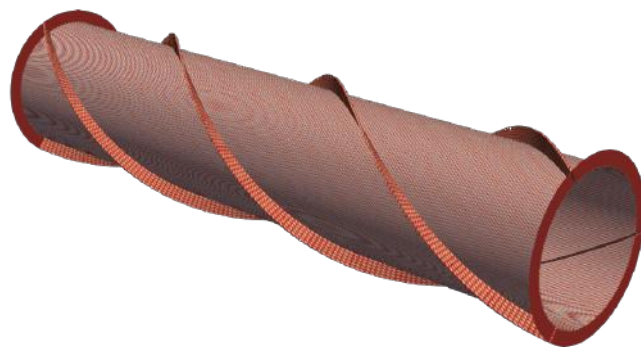


Figure 7.19: Strake Geometry - Distance Field.

Volume Convergence and Algorithm Performance

Table 7.10 shows the volume convergence of the geometry, presenting the relative error related to the real volume (the volume of the reference geometry). Notice that there is a reduction of more than 6 times of the volumetric error between the two refinement levels. The table also shows the time to repositionate the boundary nodes, SS_{time} , the time to solve the linear system to repositionate the internal nodes, PCG_{time} , and the number of iterations, PCG_{it} to achieve the prescribed tolerance.

Table 7.10: Volume Convergence and algorithm performance - Riser Mesh

Mesh Level	Volume ($u.v.$)	Error (%)	$SS_{time}(s)$	$PCG_{time}(s)$	PCG_{it}
Reference	494.484161	-	-	-	-
Level 01	494.262146	0.045	-	-	-
Level 02	494.450073	0.007	472.2	0.42	1

Mesh Quality

Figure 7.20 shows the information about the elements' quality for all the four mesh levels. It is possible to see that as well as in the YF17 and sphere examples, at each refinement level the quality of the elements is deteriorated. As in the sphere example, the quality of the elements was measured before and after the surface correction and it induces no improvement of elements' quality.

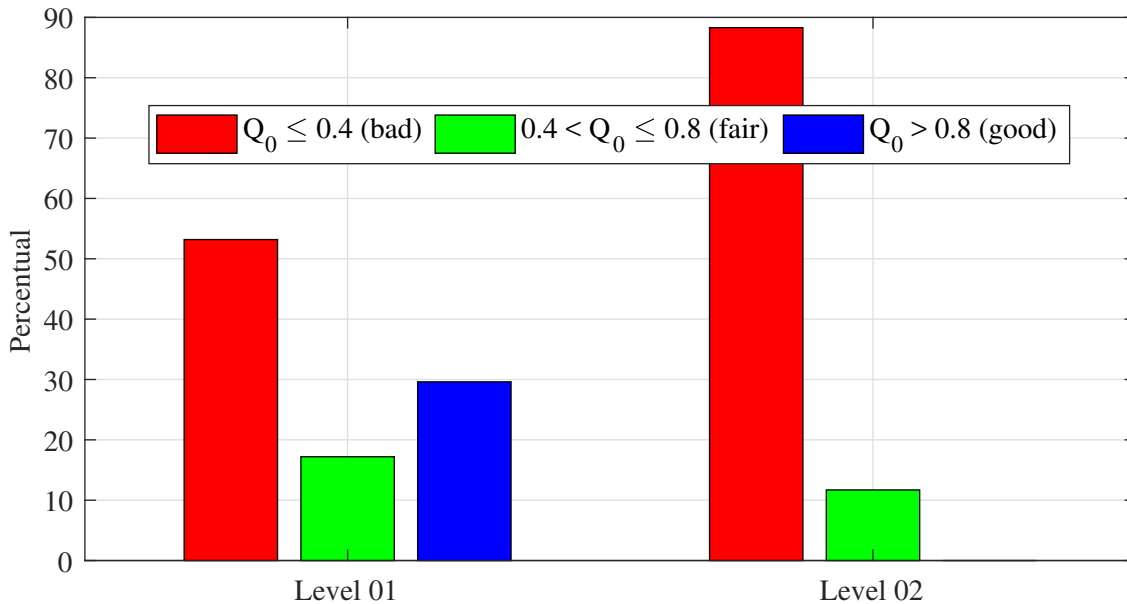


Figure 7.20: Elements Quality for each refinement level.

Chapter 8

Conclusions

The MM scheme involves no communication thanks to the index generation in each processor, which is based on the elegant pairing technique. Experimental results show the strong scalability of the scheme on generating an unstructured grid with 2 billion of tetrahedra. A hot spot analysis on 512 cores demonstrates that the whole process is communication-free.

The surface correction technique showed to be efficient in the sense of approximating the base mesh to the target mesh, diminishing the volumetric error in 12 times in comparison with the base mesh for the best case. The great issue of this technique relies on how to get correctly all information regarding the faces properties given by `FaceTools` while running the problem in parallel, whereas `FaceTools` has problems to distinguish the boundary nodes of the immersed geometry from the communication nodes (of course it is affected by the wrong extraction of the faces).

For each tetrahedron, MM produces 8 new ones with the same or less quality than the original one as showed in Table 7.4. Thus, the greater the number of good elements is, the lower will be the quality degradation at each refinement level. It was possible to see that the scheme to repositioning the internal nodes does almost nothing to the overall mesh quality thanks to the modification presented in section 5.2 that makes the small elements move without suffering with exaggerated distortion. Since we have more small elements (which is more usual), the statistics about elements quality almost do not change. So, if a refinement level has a small number of bad elements, just a few worst elements will appear at the next refinement level, and if a level has a considerable number of bad elements, there will be several new bad elements at the next refinement level.

Considering the current implementation of *FaceTools*, the execution of the code can only be done in serial mode due to its difficult to identify correctly the boundary nodes even considering the communication nodes. This fact, makes the boundary smoothing to take a considerable time to run in case of surfaces with a great number of nodes.

Future works may include the use of optimization techniques to improve the mesh quality, or simply using packages like MESQUITE [82]. The objective function for the

optimization process could be a cell metric quality [83]. By using those techniques, the boundary nodes could be immediately moved without applying a mesh moving scheme for the internal nodes, and solving the problem of bad elements by just imposing the elements to have a good aspect ratio through cell quality techniques.

Bibliography

- [1] PARTHASARATHY, V. N., GRAICHEN, C. M., HATHAWAY, A. F. “A comparison of tetrahedron quality measures”. In: *Finite Elements in Analysis and Design*, v. 15, pp. 255–261, 1993.
- [2] LÖHNER, R., BAUM, J. D. “Scaling Up Multiphysics”. Springer, 2014. doi: 10.1007/978-3-319-06136-8_15.
- [3] KURTH, T., TREICHLER, S., ROMERO, J., MUDIGONDA, M., LUEHR, N., PHILLIPS, E., MAHESH, A., MATHESON, M., DESLIPPE, J., FATICA, M., PRABHAT, HOUSTON, M. “Exascale Deep Learning for Climate Analytics”, oct 2018.
- [4] BERGMAN, K., BORKAR, S., CAMPBELL, D., CARLSON, W., DALLY, W., DENNEAU, M., FRANZON, P., HARROD, W., HILLER, J., KARP, S., KECKLER, S., KLEIN, D., LUCAS, R., RICHARDS, M., SCARPELLI, A., SCOTT, S., SNAVELY, A., STERLING, T., WILLIAMS, R. S., YELICK, K., KOGGE, P. “ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Peter Kogge, Editor & Study Lead”, 2008.
- [5] ROUNTREE, B., LOWENTHAL, D. K., FUNK, S., FREEH, V. W., DE SUPINSKI, B. R., SCHULZ, M. “Bounding energy consumption in large-scale MPI programs”, *Proceedings of the 2007 ACM/IEEE conference on Supercomputing - SC '07*, p. 1, 2007. doi: 10.1145/1362622.1362688.
- [6] DEMMEL, J. “Communication-avoiding algorithms for linear algebra and beyond”. In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 585–585, May 2013. doi: 10.1109/IPDPS.2013.123.
- [7] WILLIAMS, S., WATERMAN, A., PATTERSON, D. “Roofline: An Insightful Visual Performance Model For Multicore Architectures”, *Communications of the ACM*, v. 52, n. 4, pp. 65, 2009. ISSN: 00010782. doi: 10.1145/1498765.1498785.
- [8] WANG, X.-Q., JIN, X.-L., KOU, D.-Z., CHEN, J.-H. “A Parallel Approach for the Generation of Unstructured Meshes with Billions of Elements on

Distributed-Memory Supercomputers”, *International Journal of Parallel Programming*, v. 45, n. 3, pp. 680–710, 2017. ISSN: 1573-7640. doi: 10.1007/s10766-016-0452-3.

- [9] KABELIKOVA, P., RONOVSKY, A., VONDRAKA, V. *Parallel Mesh Multiplication for Code_Saturne*. PRACE, Partnership for Advanced Computing in Europe, Prace white paper available online at.
- [10] HOUZEAUX, G., DE LA CRUZ, R., OWEN, H., VÁZQUEZ, M. “Parallel uniform mesh multiplication applied to a Navier-Stokes solver”, *Computers and Fluids*, v. 80, n. 1, pp. 142–151, 2013. ISSN: 00457930. doi: 10.1016/j.compfluid.2012.04.017.
- [11] SILVA, R. M., LIMA, B. S. J., CAMATA, J. J., ELIAS, R. N., COUTINHO, A. L. G. A. “Communication-Free Parallel Mesh Multiplication for Large Scale Simulations”. In: *VECPAR 2018 Proceedings*, Springer, 2019.
- [12] ROUNDTABLE, I. M. “Conferences - Meshing Research Corner”. Available at <https://imr.sandia.gov/papers/topics.html>.
- [13] CAREY, G. F., GENERATION, C. G. *Adaptation and Solution Strategies*. Taylor & Francis, Series in Computational and Physical Processes in Mechanics and Thermal Sciences, 1997.
- [14] MESRI, Y., ZERGUINE, W., DIGONNET, H., SILVA, L., COUPEZ, T. “Dynamic Parallel Adaptation for Three Dimensional Unstructured Meshes: Application to Interface Tracking”. In: *Proceedings of the 17th International Meshing Roundtable*, pp. 195–212, 2008.
- [15] ELIAS, R. N., CAMATA, J. J., AVELEDA, A., COUTINHO, A. L. “Evaluation of message passing communication patterns in finite element solution of coupled problems”. In: *International Conference on High Performance Computing for Computational Science*, pp. 306–313. Springer, 2010.
- [16] BAUMAN, P. T., STOGNER, R. H. “GRINS: a multiphysics framework based on the libmesh finite element library”, *SIAM Journal on Scientific Computing*, v. 38, n. 5, pp. S78–S100, 2016.
- [17] GASTON, D. R., PERMANN, C. J., PETERSON, J. W., SLAUGHTER, A. E., ANDRES, D., WANG, Y., SHORT, M. P., PEREZ, D. M., TONKS, M. R., ORTENSU, J., MARTINEAU, R. C. “Physics- based multiscale coupling for full core nuclear reactor simulation”, *Annals of Nuclear Energy, Special Issue on Multi-Physics Modelling of LWR Static and Transient Behaviour*, v. 84, pp. 45–54, 2015. doi: <http://dx.doi.org/10.1016/j.anucene.2014.09.060>.

- [18] Logg, A., Mardal, K.-A., Wells, G. (Eds.). *Automated Solution of Differential Equations by the Finite Element Method*, v. 84. Springer, of Lecture Notes in Computational Science and Engineering, 2012.
- [19] BANGERTH, W., HARTMANN, R., KANSCHAT, G. “deal. II-A general-purpose object-oriented finite element library”, *ACM Transactions on Mathematical Software (TOMS)*, v. 33, n. 4, pp. 24, 2007.
- [20] KIRK, B. S., PETERSON, J. W., STOGNER, R. H., CAREY, G. F. “libMesh: A C++ library for parallel adaptive mesh refinement/coarsening simulations”, *Engineering with Computers*, v. 22, pp. 237–254, 2006.
- [21] G., K., KUMAR, V. M. . “Unstructured Graph Partitioning and Sparse Matrix Ordering System”, *Technical Report. Depart. of Comp. Science*, Univ. of Minnesota, Mineapolis, EUA, 1998.
- [22] BANGERTH, W., BURSTEDDE, C., HEISTER, T., KRONBICHLER, M. “Algorithms and data structures for massively parallel generic adaptive finite element codes”, *ACM Transactions on Mathematical Software (TOMS)*, v. 38, n. 2, pp. 14, 2011.
- [23] BURSTEDDE, C., WILCOX, L. C., GHATTAS, O. “p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees”, *SIAM J. Sci. Comput*, v. 3(33), pp. 1103–1133, 2011.
- [24] VAZQUEZ, M., HOUZEAUX, G., KORIC, S., ARTIGUES, A., AGUADO-SIERRA, J., ARÍS, R., MIRA, D., CALMET, H., CUCCHIETTI, F., OWEN, H., TAHA, A., BURNESSE, E. D., CELA, J. M., VALERO, M. “Alya: Multiphysics engineering simulation toward exascale”, *Journal of Computational Science*, v. 14, pp. 15 – 27, 2016.
- [25] YILMAZ, E., ALIABADI, S. “Surface conformed linear mesh and data subdivision technique for large-scale flow simulation and visualization in Variable Intensity Computational Environment”, *Computers and Fluids*, v. 80, n. Complete, pp. 388–402, 2013. doi: 10.1016/j.compfluid.2012.01.017.
- [26] GARGALLO-PEIRÓ, A., HOUZEAUX, G., ROCA, X. “Subdividing triangular and quadrilateral meshes in parallel to approximate curved geometries”, *Procedia Engineering*, v. 203, pp. 310 – 322, 2017. ISSN: 1877-7058. 26th International Meshing Roundtable, IMR26, 18-21 September 2017, Barcelona, Spain.
- [27] OVCHARENKO, A., IBANEZ, D., DELALONDRE, F., SAHNI, O., JANSEN, K. E., CAROTHERS, C. D., SHEPHARD, M. S. “Neighborhood Com-

- munication Paradigm to Increase Scalability in Large-Scale Dynamic Scientific Applications”, *Parallel Computing*, v. 38, pp. 140–156, 2012. doi: <https://doi.org/10.1016/j.parco.2011.10.013>.
- [28] MIRAS, T., CAMATA, J. J., ELIAS, R. N., ALVES, J. L., ROCHINHA, F. A., COUTINHO, A. L. “A staggered procedure for fluid–object interaction with free surfaces, large rotations and driven by adaptive time stepping”, *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, v. 40, n. 4, pp. 239, 2018.
- [29] BEY, J. “Simplicial grid refinement: on Freudenthal’s algorithm and the optimal number of congruence classes”, *Numer. Math*, v. 85, pp. 1–29, 2000.
- [30] ELIAS, R. N., MARTINS, M. A. D., COUTINHO, A. L. G. A. “Parallel Edge-Based Inexact Newton Solution of Steady Incompressible 3D Navier-Stokes Equations”. In: *Lecture Notes in Computer Science*, pp. 1237–1245, 2005. doi: 10.1007/11549468_135.
- [31] ELIAS, R. N., COUTINHO, A. L. G. A. “Stabilized edge-based finite element simulation of free-surface flows”, *International Journal for Numerical Methods in Fluids*, v. 54, n. 6-8, pp. 965–993, jun 2007. ISSN: 02712091. doi: 10.1002/fld.1475.
- [32] ELIAS, R. N., GONCALVES, M. A., COUTINHO, A. L. G. A., ESPERANCA, P. T. T., MARTINS, M. A. D., FERREIRA, M. D. A. S. “Computational Techniques for Stabilized Edge-Based Finite Element Simulation of Nonlinear Free-Surface Flows”, *Journal of Offshore Mechanics and Arctic Engineering*, v. 131, n. 4, pp. 041103, 2009. ISSN: 08927219. doi: 10.1115/1.3124136.
- [33] BAZILEVS, Y., TAKIZAWA, K., TEZDUYAR, T. E. *Computational Fluid-Structure Interaction: Methods and Applications*. Chichester, UK, John Wiley & Sons, Ltd, jan 2012. ISBN: 9780470978771. doi: 10.1002/9781118483565.
- [34] LINS, E. F., ELIAS, R. N., GUERRA, G. M., ROCHINHA, F. A., COUTINHO, A. L. G. A. “Edge-based finite element implementation of the residual-based variational multiscale method”, *International Journal for Numerical Methods in Fluids*, v. 61, n. 1, pp. 1–22, sep 2009. ISSN: 02712091. doi: 10.1002/fld.1941.
- [35] GUERRA, G. M., ZIO, S., CAMATA, J. J., DIAS, J., ELIAS, R. N., MATTOSO, M., B. PARAIZO, P. L., G. A. COUTINHO, A. L., ROCHINHA, F. A. “Uncertainty quantification in numerical simulation of particle-laden flows”, *Com-*

putational Geosciences, v. 20, n. 1, pp. 265–281, feb 2016. ISSN: 1420-0597. doi: 10.1007/s10596-016-9563-6.

- [36] VALLI, A. M. P., CAREY, G. F., COUTINHO, A. L. G. A. “Control strategies for timestep selection in finite element simulation of incompressible flows and coupled reaction-convection-diffusion processes”, *International Journal for Numerical Methods in Fluids*, v. 47, n. 3, pp. 201–231, jan 2005. ISSN: 0271-2091. doi: 10.1002/flid.805.
- [37] COUTINHO, A. L. G. A., MARTINS, M. A. D., SYDENSTRICKER, R. M., ELIAS, R. N. “Performance comparison of data-reordering algorithms for sparse matrix-vector multiplication in edge-based unstructured grid computations”, *International Journal for Numerical Methods in Engineering*, v. 66, n. 3, pp. 431–460, apr 2006. ISSN: 0029-5981. doi: 10.1002/nme.1557.
- [38] RIBEIRO, F. L. B., COUTINHO, A. L. G. A. “Comparison between element, edge and compressed storage schemes for iterative solutions in finite element analyses”, *International Journal for Numerical Methods in Engineering*, v. 63, n. 4, pp. 569–588, may 2005. ISSN: 0029-5981. doi: 10.1002/nme.1290.
- [39] HUGHES, T. J. R., FERENCZ, R. M., HALLQUIST, J. O. “Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients”, *Computer Methods in Applied Mechanics and Engineering*, v. 61, n. 2, pp. 215–248, 1987. ISSN: 00457825. doi: 10.1016/0045-7825(87)90005-3.
- [40] COUTINHO, A. L. G. A., ALVES, J. L. D., LIMA, E. C. P., EBECKEN, N. F. F. “An Element-by-Element Lanczos Solver for Large Sets of FEM Equations”. In: *1st World Congress in Computational Mechanics*, Austin, Texas, 1986.
- [41] SOTO, O., LÖHNER, R., CEBRAL, J., CAMELLI, F. “A stabilized edge-based implicit incompressible flow formulation”, *Computer Methods in Applied Mechanics and Engineering*, v. 193, n. 23-26, pp. 2139–2154, jun 2004. ISSN: 0045-7825. doi: 10.1016/J.CMA.2004.01.018.
- [42] KARANAM, A. K., JANSEN, K. E., WHITING, C. H. “Geometry based pre-processor for parallel fluid dynamic simulations using a hierarchical basis”, *Engineering with Computers*, v. 24, n. 1, pp. 17–26, Mar 2008. ISSN: 1435-5663. doi: 10.1007/s00366-007-0063-0.
- [43] BEY, J. “Simplicial grid refinement: on Freudenthal’s algorithm and the optimal number of congruence classes”, *Numerische Mathematik*, v. 85, n. 1, pp. 1–29, 2000. ISSN: 0029599X. doi: 10.1007/s002110050475.

- [44] LIU, A., JOE, B. “Quality Local Refinement of Tetrahedral Meshes Based on Bisection”, *SIAM Journal on Scientific Computing*, v. 16, n. 6, pp. 1269–1291, 1995. ISSN: 1064-8275. doi: 10.1137/0916074.
- [45] CASONI, E., JÉRUSALEM, A., SAMANIEGO, C., EGUZKITZA, B., LAFORTUNE, P., TIAHJANTO, D. D., SÁEZ, X., HOUZEAUX, G., VÁZQUEZ, M. “Alya: Computational Solid Mechanics for Supercomputers”, *Archives of Computational Methods in Engineering*, v. 22, n. 4, pp. 557–576, 2015. ISSN: 11343060. doi: 10.1007/s11831-014-9126-8.
- [46] ROSENBERG, A. L. “Efficient pairing functions-and why you should care”. In: *Proceedings 16th International Parallel and Distributed Processing Symposium*, pp. 7 pp–, April 2002. doi: 10.1109/IPDPS.2002.1016532.
- [47] TARAU, P. “On Two Infinite Families of Pairing Bijections”, *CoRR*, v. abs/1301.0129, 2013.
- [48] LAWDER, J., KING, P. “Using space-filling curves for multi-dimensional indexing”, *Lecture Notes in Computer Science*, v. 1832, pp. 20–35, 2000.
- [49] K LAWDER, J. “Calculation of Mappings Between One and n-dimensional Values Using the Hilbert Space-filling Curve”, 2000.
- [50] LAWDER, J. K., KING, P. J. H. “Querying multi-dimensional data indexed using the hilbert space-filling curve”, *SIGMOD Rec*, v. 30, pp. 19–24, 2001.
- [51] SZUDZIK, M. “An elegant pairing function”. In: *NKS 2006 Wolfram Science Conference*, 2006. S lides at <http://szudzik.com/ElegantPairing.pdf>.
- [52] SCHROEDER, W., MARTIN, K., LORENSEN, B., KITWARE, I. *The visualization toolkit : an object-oriented approach to 3D graphics*. Kitware, 2006. ISBN: 193093419X.
- [53] SEDGEWICK, R. *Algorithms in C*. Addison-Wesley, 1998. ISBN: 0201314525.
- [54] SAMET, H. *The design and analysis of spatial data structures*. 1990.
- [55] KENNEL, M. B. “KD TREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space”, aug 2004.
- [56] OTAIR, M. “Approximate K-Nearest Neighbour Based Spatial Clustering Using K-D Tree”, *International Journal of Database Management Systems*, v. 5, n. 1, pp. 97–108, feb 2013. ISSN: 09755985. doi: 10.5121/ijdms.2013.5108.

- [57] LOHNER, R. *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods*, v. 508. 2004. ISBN: 9780470519073. doi: 10.1017/S0022112004229603.
- [58] LOHNER, R. *Applied Computational Fluid Dynamics Techniques: An Introduction Based on Finite Element Methods*, v. 508. 2004.
- [59] KARUMANCHI, N. *Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles*. CareerMonk Publications, 2017. ISBN: ,978-8193245279.
- [60] MIKHAIL J. ATALLAH, M. B. *Algorithms and Theory of Computation Handbook, Second Edition, Volume 1: General Concepts and Techniques (Chapman & Hall/CRC Applied Algorithms and Data Structures series)*.
- [61] AUBRY, R., LÖHNER, R. “On the most normal’ normal”, *Communications in Numerical Methods in Engineering*, v. 24, n. 12, pp. 1641–1652, oct 2007. ISSN: 10698299. doi: 10.1002/cnm.1056.
- [62] AUBRY, R., MESTREAU, E. L., DEY, S., KARAMETE, B. K., GAYMAN, D. “On the ’most normal’ normal-Part 2”, *Finite Elements in Analysis and Design*, v. 97, pp. 54–63, 2015. doi: 10.1016/j.finel.2015.01.005.
- [63] PIRZADEH, S. “Three-dimensional unstructured viscous grids by the advancing-layers method”, *AIAA Journal*, v. 34, n. 1, pp. 43–49, jan 1996. ISSN: 0001-1452. doi: 10.2514/3.13019.
- [64] JOHNSON, A., TEZDUYAR, T. “Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces”, *Computer Methods in Applied Mechanics and Engineering*, v. 119, n. 1-2, pp. 73–94, nov 1994. ISSN: 0045-7825. doi: 10.1016/0045-7825(94)00077-8.
- [65] STEIN, K., TEZDUYAR, T., BENNEY, R. “Mesh Moving Techniques for Fluid-Structure Interactions with Large Displacements”, *Journal of Applied Mechanics*, 2003. ISSN: 00218936. doi: 10.1115/1.1530635.
- [66] MASUD, A., BHANABHAGVANWALA, M., KHURRAM, R. A. “An adaptive mesh rezoning scheme for moving boundary flows and fluid-structure interaction”, *Computers and Fluids*, v. 36, n. 1, pp. 77–91, 2007. ISSN: 00457930. doi: 10.1016/j.compfluid.2005.07.013.
- [67] KANCHI, H., MASUD, A. “A 3D adaptive mesh moving scheme”, *International Journal for Numerical Methods in Fluids*, v. 54, n. 6-8, pp. 923–944, jun 2007. ISSN: 02712091. doi: 10.1002/flf.1512.

- [68] MASUD, A., HUGHES, T. J. “A space-time Galerkin/least-squares finite element formulation of the Navier-Stokes equations for moving domain problems”, *Computer Methods in Applied Mechanics and Engineering*, v. 146, n. 1-2, pp. 91–126, jul 1997. ISSN: 00457825. doi: 10.1016/S0045-7825(96)01222-4.
- [69] KNUPP, P. M. “Algebraic Mesh Quality Metrics”, *SIAM Journal on Scientific Computing*, v. 23, n. 1, pp. 193–218, jan 2001. ISSN: 1064-8275. doi: 10.1137/S1064827500371499.
- [70] CAENDISH, J. C., FIELD, D. A., FREY, W. H. “An approach to automatic three-dimensional finite element mesh generation”, *International Journal for Numerical Methods in Engineering*, v. 21, n. 2, pp. 329–347, feb 1985. ISSN: 0029-5981. doi: 10.1002/nme.1620210210.
- [71] BAKER, T. J. “Element quality in tetrahedral meshes”. In: *Proc. 7th Int. Conf. on Finite Element Methods in Flow Problems*, pp. 1018–1024, 1989.
- [72] DE COUGNY, H. L., GEORGES, M. K., SHEPHARD, M. S. *Explicit node point mesh smoothing within the octree mesh generator*. Program for Automated Modeling, Scientific Computation Research Center, Rensselaer Polytechnic Inst., 1990.
- [73] DANNELONGUE, H., TANGUY, P. “Three-dimensional adaptive finite element computations and applications to non-Newtonian fluids”, *International journal for numerical methods in fluids*, v. 13, n. 2, pp. 145–165, 1991.
- [74] GRAICHEN, C., PARTHASARATHY, V. *OCTREE Theoretical Manual*. Relatório técnico, GE-CRD Report, 1991.
- [75] BRANETS, L., CAREY, G. F. “A local cell quality metric and variational grid smoothing algorithm”, *Engineering with Computers*, v. 21, n. 1, pp. 19–28, nov 2005. ISSN: 1435-5663. doi: 10.1007/s00366-005-0309-7.
- [76] FREY, P. J., GEORGE, P. L. *Mesh generation : application to finite elements*. ISTE, 2008. ISBN: 9781118623824.
- [77] CGNS. “Unstructured mesh for YF-17”. Disponível em: <<https://cgns.github.io/CGNSFiles.html>>. Accessed: 2018-08-22.
- [78] SHENDE, S. S., MALONY, A. D. “The TAU parallel performance system”, *The International Journal of High Performance Computing Applications*, v. 20, n. 2, pp. 287–311, 2006.

- [79] GEUZAINÉ, C., REMACLE, J.-F. “Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities”, *International Journal for Numerical Methods in Engineering*, v. 79, pp. 1309 – 1331, 2009.
- [80] ANSYS INC. *Programmer’s Manual for ANSYS - ANSYS Release 11.0*, 2007.
- [81] KORKISCHKO, I., MENEGHINI, J. R. *Experimental investigation of flow-induced vibration on isolated and tandem circular cylinders fitted with strakes*. Relatório técnico, 2010.
- [82] BREWER, M. L., DIACHIN, L. F., KNUPP, P. M., LEURENT, T., MELANDER, D. J. “The Mesquite Mesh Quality Improvement Toolkit.” In: *IMR*, 2003.
- [83] KIM, J., PANITANARAK, T., SHONTZ, S. M. “A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling”, *International Journal for Numerical Methods in Engineering*, v. 94, n. 1, pp. 20–42, 2013. doi: 10.1002/nme.4431.