University of California
Santa Barbara

Simulation of incompressible viscous flows on distributed Octree grids

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Mechanical Engineering

by

Arthur Guittet

Committee in charge:

Professor Frédéric Gibou, Chair
Professor Ted Bennett
Professor Sumita Pennathur
Professor Todd Squires

June 2016

The Dissertation of Arthur Guittet is approved.

Professor Ted Bennett

Professor Sumita Pennathur

Professor Todd Squires

Professor Frédéric Gibou, Committee Chair

March 2016

Simulation of incompressible viscous flows on distributed Octree grids

Arthur Guittet

Curriculum Vitæ
Arthur Guittet

Education

2016	Ph.D. in Mechanical Engineering, University of California, Santa Barbara.
2010	M.S. in Complex Adaptive Systems, Applied physics, Chalmers Tekniska Högskola, Gothenburg, Sweden.
2010	M.S. in Computer Engineering, École Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise, Évry, France.

Publications

- Ásdís Helgadóttir, **Arthur Guittet** and Frédéric Gibou: "On Solving the Poisson Equation with Discontinuities on Irregular Interfaces - GFM and VIM", *Computer Methods in Applied Mechanics and Engineering*, under review.
- Mohammad Mirzadeh, **Arthur Guittet**, Carsten Burstedde and Frédéric Gibou: "Parallel Level-Set Methods on Adaptive Tree-Based Grids", *Journal of Computational Physics*, under review.
- **Arthur Guittet**, Maxime Theillard and Frédéric Gibou: "A stable projection method for the incompressible Navier–Stokes equations on arbitrary geometries and adaptive Quad/Octrees", *Journal of Computational Physics*, 292 (2015), 215-238.
- **Arthur Guittet**, Mathieu Lepilliez, Sébastien Tanguy and Frédéric Gibou: "Solving elliptic problems with discontinuities on irregular domains – the Voronoi Interface Method", *Journal of Computational Physics*, 298 (2015), 747-765.
- Emmanuel Brun, **Arthur Guittet** and Frédéric Gibou: "A local level-set method using a hash table data structure", *Journal of Computational Physics*, 31 (2012), 2528–2536.
- Pierre Dossantos-Uzarralde and **Arthur Guittet**: "A Polynomial Chaos Approach for Nuclear Data Uncertainties Evaluations", *Nuclear Data Sheets*, 109 (2008), 2894-2899.

Conference presentations

- **Arthur Guittet**, Maxime Theillard and Frédéric Gibou: "A Stable Projection Method for the Incompressible Navier-Stokes Equations on Arbitrary Geometries and Adaptive Quad/oc-Trees", SIAM Conference on Computational Science and Engineering (2015), Salt Lake City, Utah.
- **Arthur Guittet**, Maxime Theillard and Frédéric Gibou: "An unconditionally stable Navier-Stokes solver on Adaptive Cartesian grids", So Cal Fluids IX (2015), San Diego, California.

Abstract

Simulation of incompressible viscous flows on distributed Octree grids

by

Arthur Guittet

This dissertation focuses on numerical simulation methods for continuous problems with irregular interfaces. A common feature of these types of systems is the locality of the physical phenomena, suggesting the use of adaptive meshes to better focus the computational effort, and the complexity inherent to representing a moving irregular interface. We address these challenges by using the implicit framework provided by the Level-Set method and implemented on adaptive Quadtree (in two spatial dimensions) and Octree (in three spatial dimensions) grids. This work is composed of two sections.

In the first half, we present the numerical tools for the study of incompressible monophasic viscous flows. After a study of an alternative grid storage structure to the Quad/Oc-tree data structure based on hash tables, we introduce the extension of the level-set method to massively parallel forests of Octrees. We then detail the numerical scheme developed to attain second order accuracy on non-graded Quad/Oc-tree grids and demonstrate the validity and robustness of the resulting solver. Finally, we combine the fluid solver and the parallel framework together and illustrate the potential of the approach.

The second half of this dissertation presents the Voronoi Interface Method (VIM), a new method for solving elliptic systems with discontinuities on irregular interfaces such as the ones encountered when simulating viscous multiphase flows. The VIM relies on a Voronoi mesh built on an underlying Cartesian grid and is compact and second order accurate while preserving the symmetry and positiveness of the resulting linear system.

We then compare the VIM with the popular Ghost Fluid Method before adapting it to the simulation of the problem of the electropermeabilization of cells.

Contents

Permissions and Attributions

- The content of chapter 1 is the result of a collaboration with Emmanuel Brun and Frédéric Gibou, and has previously appeared in the Journal of Computational Physics as "*A local level-set method using a hash-table data structure*" [1]. It is reproduced here with the permission of the publisher under license number 3811670078310.

- The content of chapter 3 is the result of a collaboration with Maxime Theillard and Frédéric Gibou, and has previously appeared in the Journal of Computational Physics as "*A stable projection method for the incompressible Navier–Stokes equations on arbitrary geometries and adaptive Quad/Octrees*" [2]. It is reproduced here with the permission of the publisher under license number 3811670366101.

- The content of chapter 5 is the result of a collaboration with Mathieu Lepilliez, Sébastien Tanguy and Frédéric Gibou, and has previously appeared in the Journal of Computational Physics as "*Solving elliptic problems with discontinuities on irregular domains - the Voronoi Interface Method*" [3]. It is reproduced here with the permission of the publisher under license number 3811670567834.

Introduction

The simulation of fluid flows has always been at the heart of computer science, and the early computers were developed with the goal of predicting the motion of fluids. The understanding of the interaction between fluids and complex irregular structures is critical to many applications, from aerodynamics to hydrodynamics and microfluidics, through biological flows and polymer dynamics.

A common difficulty to all these problems is the necessity to solve the equations of fluid dynamics in the presence of a non-trivial geometry. Two main categories of methods exist to capture irregular boundaries, the explicit methods (body-fitted) that capture the interface by adapting the computational mesh to the geometry, and the implicit methods. The body-fitted methods rely on a mesh with points lying on the irregular interface. This makes the implementation of boundary conditions trivial and lead to very elegant and effective finite elements formulations to solve the fluid dynamic equations. However, the quality of the mesh greatly impacts that of the solution and renders the problem of building an adequate mesh very expensive and complex. On the other hand, implicit methods can be implemented on regular Cartesian mesh, thus eliminating the complexity of generating a mesh of good quality, with the cost of making the implementation of boundary conditions more complex.

Among the various implicit representation that exist, we choose the level-set method framework to represent the irregular boundaries, a powerful tool developed by Osher

and Sethian that captures a moving front as the zero contour of a function existing in the entire computational domain. This framework naturally enables complex topological changes and arbitrary deformations and provides a sharp representation of the interface. Furthermore, the recent development in the level-set community have produced efficient methods for implementing complex boundary conditions on the irregular interface.

A traditional weakness of the level-set method is its so-called mass loss. The equations governing the evolution of the level-set function must be discretized in order to be solved numerically, and the numerical dissipation generated results in the domain enclosed by the zero-level shrinking. A family of methods have been developed to address this issue, such as the particle level-set method and the block grid structures. In chapter 1, we propose and study a new method based on hash tables to store only the part of the Cartesian mesh located close to the interface, thus reducing the memory requirements while conserving the fast O(1) access to the data expected from a Cartesian mesh. However, the performances of this new method do not match that of the Quad/Oc-tree data structure which uses a hierarchically stored computational mesh. In this context, the fast access to the mesh data is preserved and the construction of the mesh is trivial. Furthermore, the adaptivity inherent to this structure makes it possible to focus the computational effort close to the irregular boundary, alleviating the mass loss issue, as well as in the area of the computational domain where the physical structures are located, for instance in a boundary layer or in high vorticity regions.

If using an adaptive Cartesian mesh like the Quad/Oc-tree data structure helps focusing the computational effort where it is most needed, it is not sufficient to tackle complex problems in three spatial dimensions. We therefore prolong the level-set method to parallel forests of Octrees in chapter 2. We use the `p4est` library, a c library that provides the tools to create, partition and balance a forest of Octrees, as well as handle a layer of ghost points, and that has been demonstrated to scale on large systems.

This thesis focuses on incompressible viscous fluids, a category of fluid flows governed by the incompressible Navier-Stokes equations

$$\rho\left(\frac{\partial \underline{\boldsymbol{u}}}{\partial t} + \underline{\boldsymbol{u}} \cdot \nabla \underline{\boldsymbol{u}}\right) = -\nabla p + \mu \Delta \underline{\boldsymbol{u}}, \tag{0.1}$$

$$\nabla \cdot \underline{\boldsymbol{u}} = 0, \tag{0.2}$$

where μ is the viscosity of the fluid, ρ its density, p its pressure and $\underline{\boldsymbol{u}}$ is the velocity field. The standard approach to simulate this kind of flows is the projection method introduced by Chorin. The projection method solves the momentum equation (0.1) neglecting the pressure term to obtain an intermediate velocity field. This field is then projected onto the divergence free subspace by solving the continuity equation (0.2). This method was introduced by Chorin on uniform grids using the Marker-And-Cell (MAC) data layout, where the pressure is locate at the center of the cells and the components of the velocity field are locate at the center of the faces. The extension of the projection method with the MAC layout to Quad/Oc-tree data structures is not trivial and is detailed in chapter 3. We then combine this new fluid solver with the massively parallel implementation of the level-set method in chapter 4 and analyze the performances and the potential of the resulting solver.

The second part of this thesis focuses on a different aspect of fluid flow solvers that is the difficulty generated by the discontinuity encountered at the interface between two fluids. The equations governing multiphase flows are still the incompressible Navier-Stokes equations, but the boundary condition at the interface between the two fluids is now that the stress tensor and the velocity field must be continuous. In practice, this means that the simulation of multiphase flows require a solver able to handle elliptic systems with sharp discontinuities in the solution and its flux. This non-trivial problem is an active area of research and we propose a novel approach based on Voronoi partitions

in chapter 5. This compact method leads to a symmetric positive definite linear system and is second order accurate. We then compare its performance to the widely used Ghost Fluid Method in chapter 6 before illustrating its potential by applying it to the complex non-linear problem of the electropermeabilization of clusters of cells in chapter 7.

Chapter 1

Hash table structures for sparse grids storage

1.1 Introduction

The level-set method was originally introduced by Osher and Sethian [4] and has proven to be efficient for tracking evolving interfaces in numerous cases [5, 6].It relies on the simple idea of embedding a problem in a higher dimensional space and considering the interface as the zero level-set of a function, called the level-set function, in this space. This modeling procedure allows sophisticated behaviors of the interface, such as cusps, sharp corners and topological changes. Applications of the level-set method can be found in various domains such as fluid mechanics, electrodynamics and solid mechanics.

The well-known drawback of this method is the so-called mass loss due to numerical approximations. In practice, the level-set function is stored by sampling its value on a mesh. The values of interest are located close to the interface, where high accuracy is desirable in order to locate correctly the interface as well as its geometrical quantities. Therefore, computations needed in the level-set method are only needed locally to the interface and methods providing this capability are dubbed local level-set methods.

Various approaches exist: Adalsteinsson and Sethian [7] suggested to perform the

calculations for the level-set function evolution solely in a tube located close to the interface. The tube is updated every given number of steps so that it stays located close to the interface. The mesh still covers the entire domain so there is no advantages in terms of memory, but only a fraction of the nodes are processed thus improving significantly the computational time. The same idea of building a tubular grid around the interface has been exploited by Nielsen and Muset [8]. They developed a powerful but intricate data structure to keep track of the points located in the tubular area only. Another approach is to use an octree data structure [9, 10] as for example in the work of Strain [11] Popinet [12], Losasso *et al.* [13] and Min and Gibou [14]. This approach consists of meshing the domain using an octree data structure, which allows to refine the mesh more accurately where the interface lies. Octree grids are very efficient, especially in the case where the grids can be ungraded, but they still suffer from slow lookup performance: data access scales as $O(\log(n))$ in the case of a grid with n nodes. Octree data structures also consume some extra memory to store relevant information needed to represent the structure.

We present an alternative method for building adaptive meshes to track interfaces based on the hash table structure, which we will refer to as *local grid*. This data structure, widespread in computer science, allows a very fast access to elements, up to an O(1) access for some implementations, instead of O($\log(n)$) for octrees, and provides an efficient strategy for storing an implicit surface in term of memory usage. The literature abounds with examples of usage of hash table data structures [15, 16]. In this paper, we present an implementation of the level-set method using a hash table data structure and discuss the benefits and the drawbacks of the method.

1.2 The Hash Table structure

A hash table, or hash map, is a type of data structure often used in computer science. The goal of such structures is to provide efficient access to data, and they often outperforms other classical structures like search trees or lookup tables. It relies on three sets identified as the *keys*, the *buckets* in which the values associated to the keys are stored, and the *hash function*.

The keys can be any type of data, and in our case it will be a two dimensional set of indices (i, j) referencing the grid points in a band around the interface. A bucket is associated to each key, which in practice is an index in an array. The hash function is the crux of the method. Its role is to associate a value to each key in the best possible way, as illustrated in figure 1.1, thus providing an access to the values corresponding to a key in O(1).

There is no general hash function that would give an optimal solution to every problem, and finding an efficient hash function can be a challenging task. This is the bottleneck of the structure, and there is no general method for designing this function. Ideally, the hash function should match each key to a single bucket, in which case it is called a perfect hash function. If this ideal function exists, every element can be accessed in a single lookup. In practice, such a function may be impossible to design, and different keys may be associated to the same bucket, creating so-called hash collisions. There are various strategies for dealing with hash collisions, which can be grouped into two classes: closed hashing and open hashing.

The first strategy to deal with collisions is closed hashing (also called open addressing), which consists in finding another available bucket in the hash table. Consider the case where the key $(i, j-2)$ is associated with the bucket k, and we want to associate a bucket to the key $(i+1, j-1)$ but the hash function produces the already used bucket k. In this

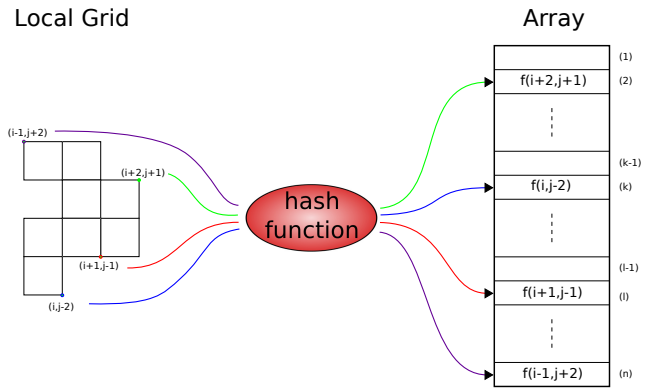


Figure 1.1: Illustration of the hash table data structure. On the left is the set of keys to be associated with the set of values (on the right) using the hash function.

case, alternate buckets need to be probed, for example with a linear probing sequence, until a free bucket is found. This procedure is illustrated in figure 1.2. Various closed hashing methods can be found in the literature, a more detailed description can be found in [17].

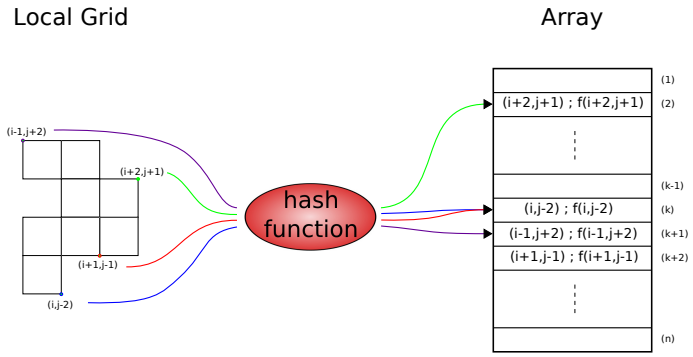


Figure 1.2: Example of a hash collision treated with a closed hashing method using a linear probing sequence.

With the second strategy, called open hashing or closed addressing, each index in the array is pointing to a data structure in which the values are stored. This external structure can be any organized structure, such as a tree or an array. We will use linked lists, in which case one talks of separate chaining or direct chaining. An illustration of

the separate chaining strategy can be found in figure 1.3. If we want to store a new value associated to the key $(i-1, j+2)$ and the hashing function gives the already used bucket l for this key, we just need to add a new element to the linked list stored in bucket l. Note that accessing elements is, in general, no longer done with a single operation. Once the bucket associated to a key is given by the hash function, the linked list it contains needs to be browsed until the right member is found. The number of hash collisions, and hence of the lookup time, depends on the efficiency of the hash function. In the worst case scenario (i.e. using an ill-behaved hash function), a lookup might be done in $O(n)$, but with a reasonably good function the average lookup time remains of the order of $O(1)$.

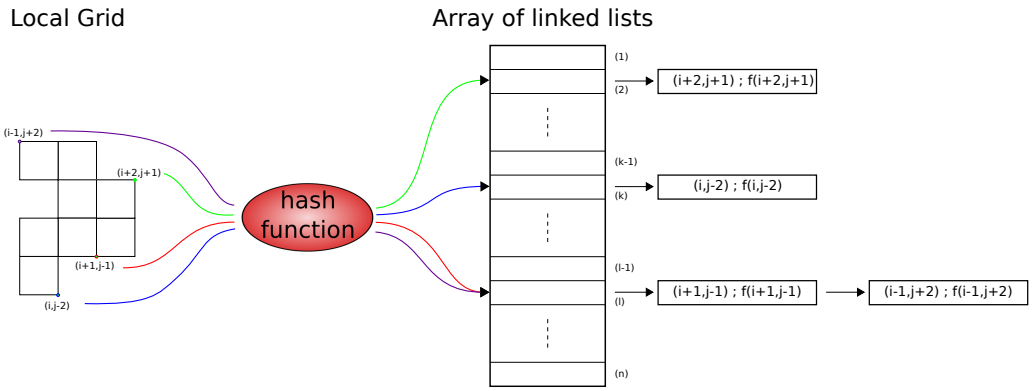


Figure 1.3: Configuration of the hash table structure with an open hashing strategy. Each bucket is pointing to a linked list in which the desired information is stored.

1.3 Implementation of the local level-set method

1.3.1 Presentation of the level-set method

Our goal is to store the local grid on which the level-set function is defined in a hash table data structure. The main idea behind the level-set method is to describe an interface

$\Gamma \in \mathbb{R}^n$ as the zero contour of a higher dimensional function $\phi \in \mathbb{R}^n$. Thus, in two spatial dimensions, a curve is described as $\Gamma = \{(x, y) : \phi(x, y) = 0\}$. The interior region is defined as $\Omega^- = \{\mathbf{x} : \phi(\mathbf{x}) < 0\}$, and the exterior region as $\Omega^+ = \{\mathbf{x} : \phi(\mathbf{x}) > 0\}$. The interface Γ is evolved in time by evolving the level-set function according to the level-set equation:

$$\frac{\partial \phi}{\partial t} \; + \; \mathbf{V} \cdot \nabla \phi \; = \; 0, \tag{1.1}$$

where $\mathbf{V}$ is the velocity field.

1.3.2 The reinitialization equation

The level-set method has been proven to be more robust and of higher accuracy when using the signed distance function to the interface as the level-set function. In order to maintained ϕ as a signed distance function, the following reinitialization equation [18] can be solved for a few iterations:

$$\phi_\tau + \mathrm{sgn}(\phi^0)(|\nabla \phi| - 1) = 0, \tag{1.2}$$

where τ represents a fictitious time and $\mathrm{sgn}(\phi^0)$ denotes the signum of ϕ^0. This algorithm thus reinitializes an arbitrary level-set function ϕ^0 into a signed distance function. The solution of this Hamilton-Jacobi equation produces shocks and rarefactions that can be captured using a combination of a Godunov scheme in space and a Total Variation Diminishing second-order Runge Kutta (TVD-RK2) scheme in time (see Shu and Osher [19], Osher and Sethian [4], and Min and Gibou [14]). In this paper, we use the following discretization:

$$\frac{\mathrm{d}\,\phi}{\mathrm{d}\,\tau} + \mathrm{sgn}(\phi^0)[H_G(D_x^+\phi, D_x^-\phi, D_y^+\phi, D_y^-\phi) - 1] = 0, \tag{1.3}$$

where H_G is the numerical Godunov Hamiltonian defined as:

$$H_G(a,b,c,d) = \begin{cases} \sqrt{\max(|a^+|^2,|b^-|^2)+\max(|c^+|^2,|d^-|^2)} & \text{if } \operatorname{sgn}(\phi^0) \le 0, \\ \sqrt{\max(|a^-|^2,|b^+|^2)+\max(|c^-|^2,|d^+|^2)} & \text{if } \operatorname{sgn}(\phi^0) > 0, \end{cases}$$

with $a^+ = \max(a,0)$ and $a^- = \min(a,0)$. The one-sided derivatives $D_x^{\pm}\phi$ and $D_y^{\pm}\phi$ are discretized using second-order accurate one-sided finite differences:

$$\begin{aligned} D_x^+\phi_{i,j} &= \frac{\phi_{i+1,j}-\phi_{i,j}}{\Delta x} - \frac{\Delta x}{2}\operatorname{minmod}(D_{xx}\phi_{i,j}, D_{xx}\phi_{i+1,j}), \\ &\text{and} \\ D_x^-\phi_{i,j} &= \frac{\phi_{i,j}-\phi_{i-1,j}}{\Delta x} - \frac{\Delta x}{2}\operatorname{minmod}(D_{xx}\phi_{i,j}, D_{xx}\phi_{i-1,j}), \end{aligned}$$

with $D_{xx}\phi$ the second-order derivative of ϕ in the x-direction, computed with a central-difference discretization. The semi-discrete equation (1.3) is discretized in time with the TVD-RK2 scheme of Shu and Osher [19]:

$$\begin{aligned} \frac{\tilde{\phi}^{n+1}-\phi^n}{\Delta\tau} + \operatorname{sgn}(\phi^0)[H_G(D_x^+\phi^n, D_x^-\phi^n, D_y^+\phi^n, D_y^-\phi^n) - 1] = 0, \\ \frac{\tilde{\phi}^{n+2}-\tilde{\phi}^{n+1}}{\Delta\tau} + \operatorname{sgn}(\phi^0)[H_G(D_x^+\tilde{\phi}^{n+1}, D_x^-\tilde{\phi}^{n+1}, D_y^+\tilde{\phi}^{n+1}, D_y^-\tilde{\phi}^{n+1}) - 1] = 0, \end{aligned}$$

and then we define ϕ^{n+1} by simple averaging: $\phi^{n+1} = (\phi^n + \tilde{\phi}^{n+2})/2$.

The reinitialization is required not to change the original location of the interface. This is enforced following the idea of Russo and Smereka [20] of including the interface location, given by ϕ_0, in the stencils of the one-sided derivatives, and its modifications from Min and Gibou [14].

In the case of our local grid, nodes on the outer edge of the band are missing at least one immediate neighbor, which poses problems when approximating the different

derivatives needed in the evolution and the reinitialization of the level-set. For such nodes, we choose to construct the missing neighbors using a linear extrapolation of the known values of ϕ using a fast marching method approach [21, 22]. This could be improved upon by using a higher-order extrapolation.

1.3.3 Evolving the level-set function with a Semi-Lagrangian scheme

If the velocity field is externally generated, i.e. it doesn't depend on the level-set, the level-set equation (1.1) is linear and semi-Lagrangian schemes (SLS) can be used. These schemes are unconditionally stable and thus avoid the standard CFL condition stating that the interface cannot move by more than one grid cell at every time step, in our case $\Delta x_{\text{smallest}}$, the smallest space step in the computational domain. The idea behind SLS is to reconstruct the trajectory of each individual particle of a system by starting from a point $\mathbf{x}$ and integrating numerically the equation governing its motion along its characteristic curves, thus tracing the particle back to its departure point $\mathbf{x_d}$.

In this article, we use a second-order accurate semi-Lagrangian method to solve the level-set equation (1.1) with the velocity field $\mathbf{V}$ externally generated. From the fact that solutions to hyperbolic problems are constant along characteristic curve, we have that for any grid point $\mathbf{x}^{n+1}$, $\phi^{n+1}(\mathbf{x}^{n+1}) = \phi^n(\mathbf{x_d})$, with ϕ^k the level-set function at time k. We use the second-order accurate mid-point method for locating the departure point, as explained in [23, 14]:

$$\begin{aligned} \hat{\mathbf{x}} &= \mathbf{x}^{n+1} - \frac{\Delta t}{2} \cdot V^n(\mathbf{x}^{n+1}), \\ \mathbf{x_d} &= \mathbf{x}^{n+1} - \Delta t \cdot V^{n+\frac{1}{2}}(\hat{\mathbf{x}}). \end{aligned}$$

We define the velocity $V^{n+\frac{1}{2}}$ at the mid-time step $t^{n+\frac{1}{2}}$ linearly from the previous velocities as $V^{n+\frac{1}{2}} = \frac{3}{2}V^n - \frac{1}{2}V^{n-1}$. Since the points $\mathbf{x_d}$ and $\hat{\mathbf{x}}$ are not grid points in general, the associated quantities $\phi^n(\mathbf{x_d})$ and $V^{n+\frac{1}{2}}(\hat{\mathbf{x}})$ are approximated using an interpolation procedure. In this work, we take the non-oscillatory interpolation procedure of [14].

1.3.4 Implementation of the hash table structure

We describe here the two key steps for the implementation of a cartesian grid on a hash table data structure, namely the construction of the local grid and its advection.

Building the adaptive grid

The adaptive grid stored in the hash table data structure is built in two steps: we first construct a full non-graded adaptive cartesian mesh before restraining it to the regions of interest and storing a refinement of those regions using the hash table data structure. We use a quadtree structure because we will compare the performance of the local grid method with a quadtree implementation [14], but in practice a standard uniform grid could serve the purpose of 'initializing' the grid.

The local mesh is then constructed using the most refined cells of the quadtree mesh. Those cells are further refined, the new nodes values being obtained by interpolation of the quadtree nodes values, and the corresponding nodes are stored in a hash table data structure. Since the initial number of nodes is given, we can build the hash table structure together with its hash function in an efficient way. We chose the size of the hash table s to be the smallest prime number larger than the number of nodes to be stored, and the hash function H to be

$$H(n) = i_n \cdot p_1 + j_n \cdot p_2 \pmod{s},$$

where n is the node index, i_n and j_n are the grid coordinates of the node and p_1 and p_2 are two large prime numbers. We choose to define the size s of the hash table in the beginning of the algorithm and we do not modify it afterwards. Since the number of nodes can exceed s after the interface evolves, we handle collisions in the hash table with the direct chaining method presented in section 1.2. Note that closed hashing would require the table to be resized when the number of nodes becomes larger than s.

Advecting the local grid

Constructing the local grid is computationally expensive because a reference mesh is needed, but the strength of the approach, in addition to the reduced number of points to handle, comes from the advection step that is a rather inexpensive and straightforward procedure. With structures like quadtrees, one has to rebuild the structure at each time step in such a way as to enforce that each node has known neighbors, a costly process. In the case of a local grid, there is no need to know the relation between neighboring nodes since the access to any node is in $O(1)$. Therefore, grid nodes can be added or removed very simply and efficiently. The procedure for adapting the local grid to the changes undergone by the interface after advecting the level-set function ϕ is described in algorithm 1.

1.4 Validation

In order to validate our implementation of the local grid on a hash table data structure, we perform typical tests in two spatial dimensions. The local grid is located close to the interface forming a tube of width $5\Delta x$ on each side of the interface. The advection is done using the Semi-Lagrangian scheme in a band of $2\Delta x$ around the interface, and we use a time step $\Delta t = \Delta x$. The information is then propagated to the rest of the tube

```
1: for all node n in the local grid do
2:     if |φ(n)| > threshold then
3:         remove node n from the local grid
4:     else
5:         for all neighbor ngbd not in the local grid do
6:             compute φ(ngbd) by using the fast marching approach to solve |∇φ| = 1
7:             if |φ(ngbd)| ≤ threshold then
8:                 add ngbd to the local grid with the value φ(ngbd)
9:             end if
10:        end for
11:    end if
12: end for
```

Algorithm 1: Algorithm for the advection of the local grid

with a fast marching algorithm. The tube is required to be larger than $2\Delta x$ to allow second order accuracy.

1.4.1 Rotation of a disk

The first test is the rotation of a disk. We consider a domain $\Omega = [-1.5, 1.5]^2$ and a disk of radius $R = 0.3$ centered initially at $(0, 0.5)$. We rotate this disk under the following rigid-body velocity field:

$$\begin{aligned} u(x,y) &= -y, \\ v(x,y) &= x, \end{aligned}$$

and rotate the disk until the final time $t = 2\pi$ is reached, i.e. we perform one complete revolution. The procedure is illustrated in figure 1.4. The accuracy of the procedure, given in table 1.1, is monitored using the error close to the interface as well as the mass loss, which is a good measure of accuracy for the level-set method. The comparison with the quadtree data structure is developed in table 1.2.

The error is computed close to the interface only, in a band of $1.2\Delta x$ with Δx the

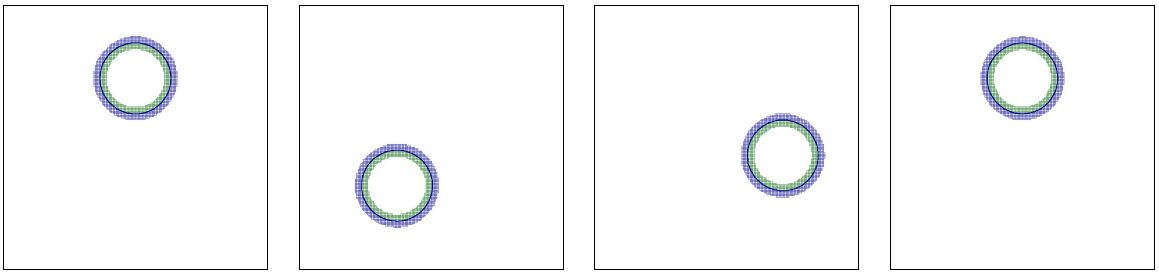


Figure 1.4: Snapshots illustrating a full revolution of a disk using a local grid of equivalent uniform resolution 256×256 stored in a hash table data structure. The disk is initially centered at $(0, 0.5)$ and has a radius $R = 0.3$. The snapshots are taken, from left to right, at time $t = 0$, $t = 2\pi/4$, $t = 2\pi/7$ and $t = 2\pi$. The level-set is evolved using the Semi-Lagrangian approach.

Res.	Time (s)	L_∞ error of ϕ	Rate	L_1 error of ϕ	Rate	Mass loss (%)	Rate
64^2	5	3.21×10^{-2}		2.70×10^{-2}		17.15	
128^2	13	8.20×10^{-3}	1.97	6.65×10^{-3}	2.02	4.40	1.96
256^2	38	2.36×10^{-3}	1.80	1.74×10^{-3}	1.93	1.16	1.92
512^2	139	7.91×10^{-4}	1.58	4.65×10^{-4}	1.90	0.31	1.90
1024^2	530	3.46×10^{-4}	1.19	1.38×10^{-4}	1.75	0.092	1.75

Table 1.1: Accuracy of the local grid stored in a hash table structure for the revolution of a disk. The disk is initially centered on $(0, 0.5)$ and the computation domain is $[-1.5, 1.5]^2$. The disk is evolved with the velocity field $(u, v) = (-y, x)$ until the time $t = 2\pi$. In this article, a 'resolution' of r^2 defines the grid size equivalent to a $r \times r$ discretization on uniform grid.

Local Grid					
Res.	# of nodes	# of slots	# of empty slots	Average occupied slots load	Mem. (Kio)
64^2	404	409	194	1.88	18.94
128^2	806	823	229	1.35	37.78
256^2	1,606	1609	588	1.57	75.28
512^2	3,226	3301	1239	1.56	151.22
1024^2	6,432	6719	2223	1.43	301.50

Quadtree					
Res.	# of nodes	# of slots	# of empty slots	Average occupied slots load	Mem. (Kio)
64^2	309	NA	NA	NA	17.57
128^2	609	NA	NA	NA	35.24
256^2	1,237	NA	NA	NA	72.19
512^2	2,509	NA	NA	NA	146.91
1024^2	5,001	NA	NA	NA	293.15

Table 1.2: Resources used by the present local grid algorithm in comparison with the quadtree data structure for the disk revolution test. The disk is initially centered on $(0, 0.5)$ and the computation domain is $[-1.5, 1.5]^2$. The disk is evolved with the velocity field $(u, v) = (-y, x)$ until the time $t = 2\pi$. The number of nodes used is of the same order for both methods, and so is the memory required. Note that around 35% slots are empty for the local grid scheme, and a better hash function would improve those results. But according to [17] a total load of around 65% is close to the optimal case of 80%.

resolution of the grid, since those points define the interface location, which we are interested in. The loss of mass is given by $|V_{initial} - V_{final}|/V_{initial}$, with V_t the area inside the interface (i.e. for $\phi < 0$) at time t. In practice, V_t is calculated by extending the local grid to the whole negative ϕ region, i.e. to the interior of the domain, and summing the area of the negative region contained in each cell. The area of the negative region contained in grid cells adjacent to the interface is computed as described by Min and Gibou in [24] The grid resolution corresponds to the number of points on a uniform grid with the same Δx.

In terms of memory, the hash table data structure is expected to be more efficient than the quadtree structure. However, for the local grid to provide accurate results, the band around the interface needs to be wide enough as to minimize the error incurred by linearly extrapolated nodes on the edge of the band. Also a smaller band limits the time step we can take in a semi-Lagrangian framework since the departure point could be outside the band. We found that at least $5\Delta x$ on both sides of the interface are needed, while a band of only $3\Delta x$ are needed in the case of the quadtree. Overall, this restriction can lead to structures that have a size equivalent to or higher than a quadtree data structure. The hash function we use provides an average access to the nodes in O(1), since the average load of the hash table occupied slots is close to one. The heaviest load observed is six nodes for a single slot.

1.4.2 Motion under a vortex velocity field

The second test, based on a proposition by [25], is more challenging as the interface thins out under the velocity field: We consider a domain $\Omega = [0, 1]^2$ and a disk of radius $R = 0.15$ centered initially at $(0.5, 0.75)$. We deform the level-set under the divergence

free velocity field:

$$\begin{aligned} u(x,y) &= -\sin^2(\pi x)\sin(2\pi y), \\ v(x,y) &= \sin^2(\pi y)\sin(2\pi x). \end{aligned}$$

This deformation is illustrated in figure 1.5, and the numerical results are collected in tables 1.3 and 1.4. As can be observed, and as expected from the results obtained in [14], the order of convergence is not at good as for the case of the rotation for coarse grids. The method is of order close to two for the mass loss and the L_1 error, and of order slightly more than 1 for the L_∞ error. As explained by Min and Gibou, and reported in [26], this is due to the fact that part of the geometry is under resolved as it deforms. In particular, part of the tail of the interface will always be under-resolved, no matter how high the resolution.

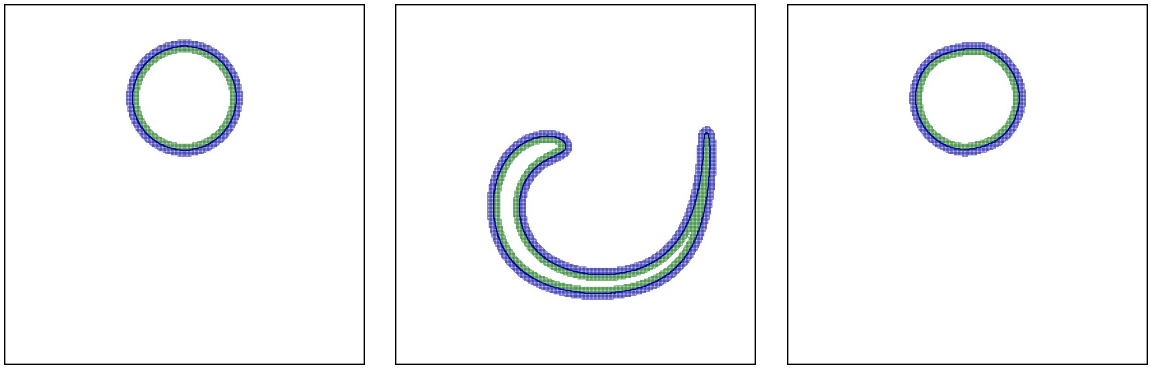


Figure 1.5: Illustration of the deformation of a disk under a vortex velocity field using a local grid of equivalent uniform resolution 256×256 stored in a hash table data structure. The disk is evolved forward until time $t = 1$ and then backward to its initial position. The snapshots have been taken from left to right at respective times $t = 0$, $t = 1 = \frac{1}{2}t_{final}$ and $t = t_{final}$. The scheme used to evolved the level-set is based on the Semi-Lagrangian approach.

The disk can be deformed further under the velocity field, and the largest the deformation is the hardest it is to recover the initial disk shape. This is precisely a situation

Res.	Time (s)	L_∞ error of ϕ	Rate	L_1 error of ϕ	Rate	Mass loss (%)	Rate
64^2	8	3.74×10^{-2}		1.43×10^{-2}		16.34	
128^2	22	1.81×10^{-2}	1.05	4.74×10^{-3}	1.59	5.58	1.55
256^2	76	8.53×10^{-3}	1.09	1.49×10^{-3}	1.67	1.84	1.60
512^2	591	3.98×10^{-3}	1.10	4.72×10^{-4}	1.66	0.61	1.59
1024^2	2,509	1.80×10^{-3}	1.14	1.61×10^{-4}	1.55	0.20	1.61

Table 1.3: Accuracy of the local grid stored in a hash table structure for the evolution of a disk in a vortex velocity field $(u, v) = (-\sin^2(\pi x)\sin(2\pi y),\ \sin^2(\pi y)\sin(2\pi x))$. The disk is initially centered on $(0.5, 0.75)$ and the computation domain is $[0, 1]^2$. The disk is evolved until the time $t = 1$ and is then evolved back to its initial state with the inverse velocity field. In this article, "resolution" means the number of grid points for an uniform grid of the same accuracy. The L_∞ and L_1 errors of ϕ are computed on the nodes adjacent to the interface. As one can observe, the method is of order close to two.

		64^2	128^2	256^2	512^2	1024^2
Local	Number of slots	1103	2713	6427	13,577	28,111
	Min average load	1.00	1.00	1.00	1.05	1.02
	Max average load	1.00	1.03	1.05	1.20	1.19
	Min empty slots	354	798	2086	4,654	1,305
	Max empty slots	657	1722	4341	9,333	6,575
	Min number of nodes	504	1,141	2,367	4,793	9,630
	Max number of nodes	948	2,690	6,376	13,478	27,563
	Min memory usage	18.36	53.48	110.95	224.67	451.25
	Max memory usage	44.44	126.09	298.88	631.78	1,292.02
Quadtree	Min number of nodes	283	588	1,206	2,426	4,844
	Max number of nodes	484	1,143	2,744	6,177	13,147
	Min memory usage	16.34	33.84	70.20	141.52	282.78
	Max memory usage	28.19	68.26	164.14	366.36	776.65

Table 1.4: Resources used by the algorithm in comparison with the quadtree data structure for the evolution of a disk in a vortex velocity field. The disk is initially centered on $(0.5, 0.75)$ and the computation domain is $[0, 1]^2$. The disk is evolved until the time $t = 1$ and is then evolved back to its initial state. The minimum and maximum average loads of the occupied slots over the whole procedure are monitored, together with the number of nodes and the memory usage. The number of nodes used by the local grid is approximately twice the number of nodes used by the quadtree, leading to a structure that requires twice more memory.

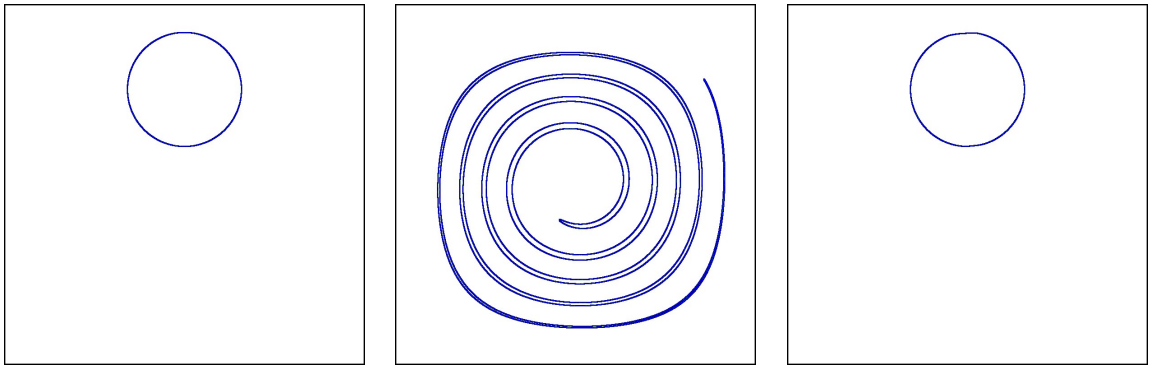


Figure 1.6: Deformation of a disk under the vortex velocity field using a local grid of equivalent uniform resolution 8192×8192. The disk (on the left) is deformed until the time $t = 6$ (second picture), then the velocity field is inverted and the disk is evolved to its initial shape (right picture). The mass loss is 0.20%, and the maximum memory usage is 51 Mio.

where high resolution implementations can provide accurate results. Figure 1.6 illustrates the deformation of the disk until the time $t = 6$, before being rewinded back to its initial state. As can be observed, the final result is close to the initial disk. We find a mass loss of 0.2%, which is quite small for such an extreme case of deformation. Therefore, the local grid we implemented succeeds in capturing the general features of this important deformation.

1.4.3 Case of shock and rarefaction solutions

In the case where the velocity $\mathbf{V}$ in equation (1.1) depends on the level-set function ϕ, equation (1.1) admits nonlinear solutions related to shock and rarefaction waves in conservation laws. In order to illustrate the ability of our framework to capture such solutions, we consider the case of a unit square moving under a normal velocity $\mathbf{V}_n = \pm\mathbf{n}$, where $\mathbf{n}$ is the outward normal to the interface. The level-set equation (1.1) is solved using a Godunov scheme similar to the one presented for the reinitialization procedure in section 1.3.2. Figure 1.7 depicts the correct shock and rarefaction solutions.

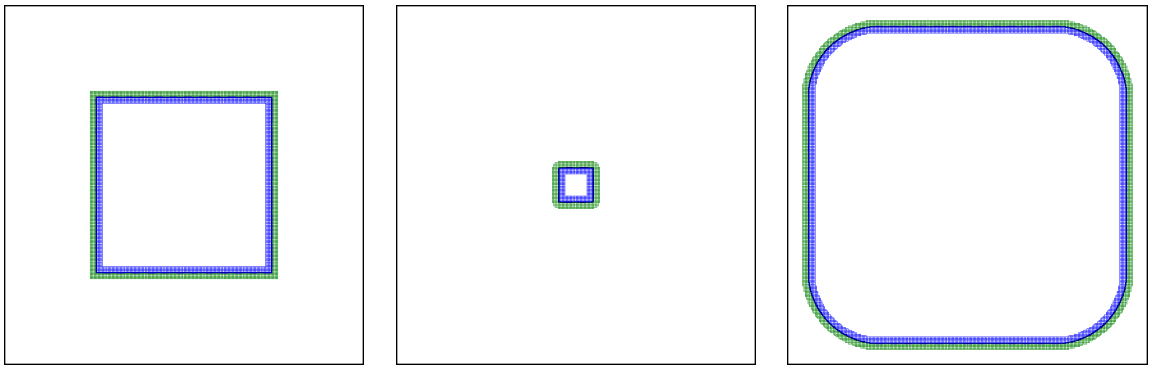


Figure 1.7: Deformation of a square under a normal velocity to the interface. The original square, on the left, is contracted in the middle, and the same original square is expanded on the right. This illustrates the ability of our framework to capture shock and rarefaction solutions.

1.5 Summary

We have presented a successful implementation of the level-set method on a hash table data structure. It is important to mention that the development of the code for the hash table based level-set method, with its linear organization, was easier and more straightforward than the implementation of the quadtree data structure, which is recursive and intricate. We note that the hash function we provide produces satisfactory results and enables a meaningful comparison of the method's performances with the quadtree data structure. We also note that the extrapolation procedure to define missing neighbors is only first-order accurate, which impacts the overall accuracy. The analysis of the numerical tests shows that even if only the nodes close to the interface are stored, the method is less efficient than the quadtree data structure for three main reasons: (1) to obtain accurate results, a rather large band is required close to the interface, which counterbalances the absence of grid nodes far from the interface; (2) the performances are deteriorated by extrapolation procedures on the outer edges of the local grid and (3) the width of the band restricts the time step and slows down the method. These issues may be resolved

by careful development of different algorithms. In addition, the hash table data structure is more suitable for parallelization than the quadtree data structure, but as it is, we find that a quadtree data structure seems more adapted than the hash table data structure for level-set algorithms.

Chapter 2

Parallel Level-Set methods on adaptive tree-based grids

2.1 Introduction

The level-set method, originally proposed by Sethian and Osher [4], is a popular and powerful framework for tracking arbitrary interfaces that undergo complicated topological changes. As a result, the level-set method has been used to a wide range of applications such as multiphase flows, image segmentation, and computer graphics [6, 5]. An important feature of this method is that the location of the interface is defined implicitly on an underlying grid. This convenience, however, comes at a price. First, compared to an explicit method, e.g. front tracking [27, 28], the level-set method is typically less accurate and mass conservation could be a problem, although progress has been made in resolving this issue [29]. Second, the level-set function has to be defined in a one dimension higher space than that of the interface. If only the location of the interface is needed, the added dimension greatly increases the overall computational cost for uniform grids. One way to avoid this problem is by computing the level-set only close to the interface, e.g. as in the narrow-band level-set method [7] or, more recently, by using a hash table to restrict both computation and storage requirements [1].

Another approach that can address both problems is the use of local grid refinement. In [11] the idea of using tree-based grids for level-set calculations was first introduced and later extended in [12, 13] for fluid simulations. More recently, authors in [14] proposed second-order accurate level-set methods on Quadtree (two spatial dimensions) and Octree (three spatial dimensions) grids. The use of adaptive tree-base grids in the context of the level-set method is quite advantageous because (i) it gives fine-grain control over errors, which typically occur close to the interface and (ii) it can effectively reduce the dimensionality of the problem by focusing most of the grid cells close to the interface. Fortunately, constructing the tree is quite simple in the presence of an interface that naturally defines an ideal metric for refinement. However, even though the use of adaptive grids can dramatically reduce the computational cost, performing high-resolution three dimensional calculations of complex interfacial problems, e.g. crystal growth in binary alloys [30], could be prohibitively expensive or even impossible on a serial machine. In this paper we extend the level-set technology on Quad-/Octrees by proposing parallel algorithms for distributed memory machines using a domain decomposition technique.

One of the main challenges in parallelizing algorithms on adaptive grids is handling the grid itself. One option is to replicate the entire grid on each process and to employ serial ordering techniques, as implemented in an earlier version of the `deal.II` library [31], or to use serial graph partitioners such as METIS [32].

This approach, however, is only scalable to a few hundred processes at best and is limited by the size of the grid itself that can fit in memory. Even though parallel general-purpose partitioners have since been popularized [33, 34] and the scalability of partitioning algorithms for unstructured grids has been improved (see e.g. [35]), their use adds extra overhead that can limit the overall scalability. Refining a grid consisting of multiple trees using recursive coordinate bisection has been implemented in the SIERRA framework [36]. Interestingly, tree-based grids have a nice spatial ordering that naturally

leads to the concept of space-filling curves (SFCs) and can be efficiently exploited for parallel load balancing [37, 38, 39].

The idea of using SFCs for parallel partitioning of Quad-/Octrees is not new in itself and has been used by many researchers. For instance, `Octor` [40] uses a Morton curve (also known as Z-curve) for traversing the leaves of an Octree for indexing and load balancing and has been scaled up to 62,000 CPU cores [41]. `Dendro` [42] is an example of a so-called linear Octree code in which new algorithms are introduced for parallel partitioning and the development of a parallel geometric multigrid that has been scaled up to about 32,000 cores [43]. More recently, authors in [44] extended these ideas by optionally allowing for a collection, or a "forest", of connected Octrees, which is partitioned in parallel using a global Morton curve. The `p4est` library [45] provides a publicly available implementation of these algorithms that is equally efficient for a single tree as well as multiple trees and has been shown to scale to more than 458k CPU cores [46]. Applications built with `p4est` have scaled to 1.57M cores [47] and 3.14M cores [48]. In fact, the algorithms presented in this paper are implemented on top of the `p4est` API. Due to the need for multiple adaptation and partitioning operations in each time step, the semi-Lagrangian method we describe below presents a stringent test of the algorithms and implementation both in terms of scalability and absolute run time.

Parallel level-set algorithms can be categorized into two groups: parallel advection algorithms and parallel reinitialization algorithms. Eulerian advection schemes can easily be parallelized but unfortunately are limited by the CFL condition, which could be very restrictive for adaptive grids. Semi-Lagrangian methods combine the unconditional stability of Lagrangian methods and the ease of use of Eulerian grids and have been successfully used for advecting the level-set function on tree-based grids [13, 49, 14]. However, parallelizing the semi-Lagrangian algorithm in a domain decomposition context is not an easy task. The reason for this is twofold. First, depending on the CFL number,

the departure points may end up outside the ghost region and in remote processes that are potentially far away. This requires a very dynamic and nonuniform communication pattern that is challenging to implement. For an adaptive grid, the situation is even more complex due to the asymmetric nature of the communication pattern (cf. section 2.3). Second, load balancing could be an issue for large CFL numbers and nonuniform velocity fields, due to clustering of departure points, which can thus considerably restrict the scalability of the algorithm. Both of these problems, of course, could be avoided by choosing CFL ≤ 1 but that would defeat the purpose of using the semi-Lagrangian algorithm in the first place.

Nonetheless, several parallel semi-Lagrangian algorithms have been proposed. A simple domain decomposition technique was used in [50] where the width of the ghost layer is fixed based on the maximum CFL number to ensure that all departure points are covered by the ghost layer. Good scalings were reported for small CFL numbers (CFL ≤ 2). However, for large CFL numbers, this leads to a large volume of communication that can limit the scalability. In [51] the authors propose a more sophisticated domain decomposition approach which uses a "dynamic ghost layer". Here the width of the ghost layer is dynamically determined at runtime based on information from previous time steps. Unfortunately, this approach also suffers from excessive communication overhead at large numbers of processes. More recently, the authors in [52] used a domain decomposition strategy on a cubed sphere but with a single layer of ghost nodes. Interpolation on remote processes is then handled by sending query points to the corresponding process and asking for the interpolated result. This approach seems to provide good scalability for transporting a single tracer up to about 1000 cores for CFL ~ 10. At higher CFL numbers, the method begins to loose scalability due to an increase in communication volume. Finally, note that although we are mainly interested in parallel semi-Lagrangian methods, one could resort to finite difference or finite element discretization methods if

small CFL numbers are acceptable. Indeed several algorithms of this type have been proposed with applications to modeling dendritic crystal growth [53], multiphase flows [54, 55, 56], and atomization process [57].

In many applications, it is desirable that the level-set function has the signed-distance property, i.e., $|\nabla\phi| = 1$. Generally, there are two approaches to enforce this property, either by solving the pseudo-time transient reinitialization equation [58, 59]

$$\phi_\tau + S(\phi_0)\left(|\nabla\phi| - 1\right) = 0,$$

or by solving the Eikonal equation

$$F(x)|\nabla\phi| = 1$$

with constant speed function $F(x) \equiv 1$. The transient reinitialization equation can be solved using explicit finite differences and thus can easily be parallelized in a domain decomposition approach. Moreover, only a few iterations may be needed if the signed-distance property is only required close to the interface [14]. This is the approach we have chosen in this paper. However, if the signed-distance property is required in the entire domain, solving the Eikonal equation is more computationally efficient. Unfortunately, the most popular algorithm for solving the Eikonal equation, i.e. the Fast Marching Method [21, 5], is inherently sequential due to causal relationship between grid points and cannot be easily parallelized. The Fast Sweeping Method (FSM) [60] is an alternative for solving the Eikonal equation iteratively. The FSM can be more computationally efficient for simple choices of speed function, e.g. as in this context, and for simple interfaces. Moreover, FSM has more potential for parallelization compared to the FMM.

One of the earliest attempt in parallelizing the FMM is reported in [61] where a do-

main decomposition algorithm was introduced. Unlike the serial FMM, however, parallel FMM potentially requires multiple iterations or "rollback operations" to enforce causality across processes. Similar ideas are described in detail in [62]. It should be noted that the number of iterations needed for the parallel FMM to converge greatly depends on the complexity of the interface and on the parallel partitioning and, in general, fewer iterations are required if the domains are aligned with the normals to the interface. Due to the nature of the Eikonal equation, shared memory machines might be a better environment for parallelization. For instance, in [63] the authors use an "adaptive" technique where individual threads implicitly define a domain decomposition at runtime. Unfortunately, this approach does not seem to be more effective than a simple static decomposition. In [64] a parallel FSM method was presented for the first time, which suffered from a plateau in the speedup. A scalable FSM was more recently proposed in [65], where the Cuthill-McKee numbering was utilized to improve scalability. A two-scale, hybrid FMM-FSM was presented in [66] which, albeit being more complicated to implement, promises even better scalability. Finally, a parallel Fast Iterative Method (FIM) was proposed in [67]. The FIM is similar to FMM in that it also maintains a list of "active nodes". However, unlike FMM, FIM avoids sorting the list and allows for concurrent updating of all nodes in an iterative fashion. In this article we choose the pseudo-time transient formulation for two reasons: 1) it is considerably easier to parallelize on Quadtrees and Octrees and 2) we are merely interested in the signed-distance property close to the interface, which only requires a few iterations.

This article is organized as follows: In section 2.2, we briefly review the sequential algorithms and discretization methods for the level-set equation on adaptive tree-based grids. These ideas are then extended in section 2.3 to parallel environments using a domain decomposition method. In section 2.4, we provide several examples that illustrate the scalability of our algorithms. Finally, we close by providing an application of our

method by considering the simulation of the solidification process by solving a Stefan problem in section 2.5.

2.2 The level-set method

The level-set method, introduced in [4], is an implicit framework for tracking interfaces that undergo complicated topological changes. In this framework, an interface is represented by the zero contour of a higher dimensional function, e.g. a curve in two spatial dimensions can be described as $\Gamma = \{(x, y)|\phi(x, y) = 0\}$, where $\phi(x, y)$ is the level-set function. The evolution of the curve under a velocity field $\underline{\mathbf{u}}$ is then obtained by solving the level-set equation:

$$\phi_t + \underline{\mathbf{u}} \cdot \underline{\nabla}\phi = 0. \tag{2.1}$$

When the velocity field does not depend on the level-set function itself, equation (2.1) can be solved using the semi-Lagrangian method. An important advantage of the semi-Lagrangian method over the regular finite difference method is its unconditional stability that allows for arbitrarily large time steps. This is particularly important when using adaptive grids since higher grid resolutions translate into impractically small time steps.

In general, an infinite number of level-set functions can describe the same zero contour and thus the same interface. However, it is desirable to choose a function with the signed distance property $|\underline{\nabla}\phi| = 1$. As detailed in section 2.1, we solve the pseudo-time transient reinitialization equation [58, 59] to achieve this property,

$$\phi_\tau + S(\phi_0)\left(|\underline{\nabla}\phi| - 1\right) = 0, \tag{2.2}$$

where τ is a pseudo time step, ϕ_0 is any level-set function that correctly describes the

interface location and $S(\phi_0)$ is an approximation of the sign function. Here, we do not go into the details of the sequential algorithms for solving equations (2.1) and (2.2). Instead, we note that the parallel algorithms presented in section 2.3 are based on the sequential methods presented earlier in [14] and refer the interested reader to the aforementioned articles and references therein for more details.

2.3 Parallel algorithms

To achieve a good parallel performance of our numerical solver, we must ensure the sufficient scalability of all components. This observation prompted the dedicated development and optimization of several techniques, which we discuss in the present section.

2.3.1 Grid management

Adaptive tree-based grids can significantly reduce the computational cost of level-set methods by restricting the fine grid close to the interface where it is most needed [11]. Moreover, adaptive tree-based grids are easy to generate in the presence of a signed-distance level-set function [14] and can efficiently be encoded using a tree data structure [10]. In order to develop scalable parallel algorithms on these grids, it is necessary to parallelize the data structure and grid manipulation methods such as refinement and coarsening of cells as well as to provide a fast method for grid partitioning and load balancing. The `p4est` library [45] is a collection of such parallel algorithms that has recently emerged and shown to scale up to 458,752 cores [46].

In `p4est` the adaptive grid is represented as a non-overlapping collection of trees that are rooted in individual cells of a common coarse grid (cf. Figure 2.1). This common coarse grid, which we will refer to as the "macromesh", can in general be an unstructured quadrilateral mesh, in two spatial dimensions, or hexahedral mesh, in three spatial

dimensions. In this article, however, it is sufficient to limit discussions to simple uniform Cartesian macromeshes. Moreover, it is implicitly assumed that the macromesh is small enough that it can be entirely replicated on all processes. For instance, in many of the applications that we are considering in this paper the macromesh is simply a single cell. `p4est` allows for arbitrary refinement and coarsening criteria through defining callback functions. In this article the refinement criteria is chosen based on the distance of individual cells to the interface. Specifically, a cell C is marked for refinement if

$$\min_{v \in V(C)} |\phi(v)| \leq \frac{LD}{2}, \tag{2.3}$$

where $V(C)$ denotes the set of all vertices of cell C, L denotes the Lipschitz constant of the level-set function, and D denotes the diagonal size of cell C. Conversely, an existing cell is marked for coarsening if

$$\min_{v \in V(C)} |\phi(v)| > LD. \tag{2.4}$$

We refer to section 3.2 of [44] for details on the parallel refinement and coarsening algorithms implemented in `p4est`.

Once the grid is adapted to the interface, it must be partitioned to ensure load balancing across processes. This is achieved by constructing a Z-curve that traverses the leaves of all trees in order of the tree index (cf. Figure 2.1). A Z-curve is a Space Filling Curve (SFC) with the important property that cells with close Z-indices are also geometrically close (on average) in the physical domain. This is beneficial since it leads to both a reduction in MPI communications and improvements of the cache performance of several algorithms such as interpolation and finite difference calculations. For more details on parallel partitioning in `p4est` one may refer to section 3.3 of [44]. Aside from

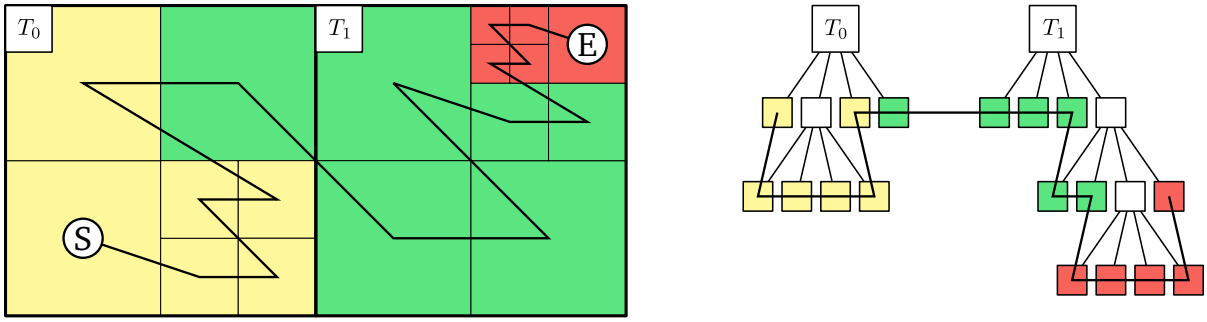


Figure 2.1: Left: a "forest" made up of two trees T_0 and T_1. Parallel partitioning is achieved by first constructing a Z-curve starting at cell "S" and ending at cell "E". Next, the one dimensional curve is split up among processes either uniformly or by assigning different weights to cells. Here processes are represented via different colors. Note how using the Z-curve naturally leads to clustering of most cells in each domain. Right: schematic of a tree data structure representing this forest and its partitioning.

grid manipulation and partitioning, we use two additional features of `p4est`, namely the generation of ghost layer cells and the creation of a globally unique node indexing. These algorithms are detailed in sections 3.5 and 3.6 of [44]. We have specifically extended the latter algorithm such that it can be applied to a non-graded refinement pattern. This is important because we can entirely skip the 2:1 balance function, which was shown to be one of the most time consuming parts of grid adaptation in `p4est` [44].

Finally, in `p4est` trees are linearized, i.e. only the leaves are explicitly stored. However, explicit knowledge of the hierarchal structure of the tree is greatly beneficial in several algorithms, e.g. in search operations needed for the interpolation algorithm. Thus, we introduce a simple reconstruction algorithm that recreates a local representation of the entire "forest" that is only adapted to local cells and, potentially, the ghost layer. This approach is similar to the ideas introduced in [68] and our tests show that in a typical application they amount to less that 1% of the entire runtime. Algorithm 2 illustrates how this reconstruction is performed. Given a forest and a layer of ghost cells from `p4est`, the algorithm generates a local representation of the forest by recursively

refining from the root until reaching the same level and location of all leaves in the local forest and the ghost layer. Note that algorithm 2 does not involve any communication and is load balanced provided that the initial forest is balanced. Figure 2.2 illustrates an example where Algorithm 2 is applied. Note how each process has independently generated a local representation of the forest that is refined to match the same leaves as in the global forest and ghost layer.

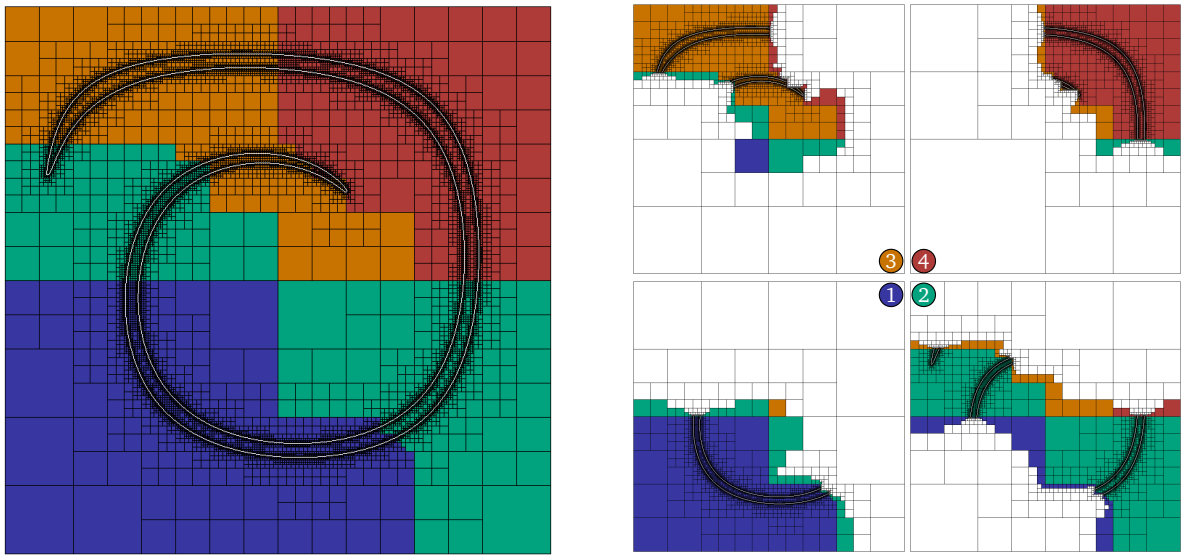


Figure 2.2: Left: a forest refined close to an interface and partitioned among four processes, as indicated by colors. Right: each process independently recreates a local forest that is refined to match the local grid and is as coarse as possible elsewhere. Note that empty cells are fictitious, i.e. they are only required to generate the hierarchal structure and are not matched by any corresponding cell in the global forest.

2.3.2 Interpolation and semi-Lagrangian methods

As indicated earlier, we use the semi-Lagrangian method to solve equation (2.1) when the velocity field is externally generated, i.e. when it does not depend explicitly on the level-set function itself. Let us rewrite equation (2.1) along the characteristic curve $\underline{\mathbf{X}}(t)$

```
1: H ← G.macromesh()                                   ▷ start with the coarsest grid
2: for tr in G.local_trees() do                        ▷ update the hierarchy to match local cells
3:     for c in tr.local_cells() do
4:         H.update_tree(tr, c)
5:     end for
6: end for
7: for c in G.ghost_cells() do                         ▷ update the hierarchy to mach local cells
8:     H.update_tree(c.get_parent_tree(), c)
9: end for
10: return H
11:
12: function H.UPDATE_TREE(tr, c)                      ▷ recursive tree reconstruction
13:     c_l ← H.get_root_cell(tr)
14:     while c_l.level() ≠ c.level() do ▷ recursively search for a cell of the same size as c
15:         if c_l.is_leaf() then c_l.split()          ▷ if the current cell is leaf, split the cell and continue search
16:         end if
17:         h ← c_l.length()                           ▷ select the search path based on cell coordinate.
18:         i ← c.x ≥ c_l.x + h/2
19:         j ← c.y ≥ c_l.y + h/2
20:         k ← c.z ≥ c_l.z + h/2
21:         c_l ← c_l.get_child(i, j, k)
22:     end while
23: end function
```

Algorithm 2: $H \leftarrow$ `Reconstruct (`G`)`: Construction of the local tree hierarchy, H, from the parallel grid, G, supplied by `p4est`. The algorithm starts at the coarsest level, i.e. the macromesh, and recursively splits the each cell to match the finest local and ghost cells generated by `p4est`.

as:

$$\begin{cases} \dfrac{\mathrm{d}\underline{\mathbf{X}}}{\mathrm{d}t} &= \underline{\mathbf{u}}, \\ \dfrac{\mathrm{d}\phi(\underline{\mathbf{X}}(t),t)}{\mathrm{d}t} &= 0. \end{cases} \tag{2.5}$$

The semi-Lagrangian method integrates equations (2.5) backward in time, i.e. starting from the grid G^{n+1} (computed iteratively as explained later on), we simply write $\phi^{n+1}(\underline{\mathbf{X}}^{n+1}) = \phi(\underline{\mathbf{X}}(t^{n+1}),t^{n+1}) = \phi(\underline{\mathbf{X}}(t^n),t^n) = \phi^n(\underline{\mathbf{X}}_d)$. Here, the characteristic curves are chosen such that $\underline{\mathbf{X}}(t^{n+1})$ are the coordinates of grids points of G^{n+1}, and $\underline{\mathbf{X}}_d$ are the departure points, which are computed using the second-order midpoint method [14]:

$$\underline{\mathbf{X}}^\star = \underline{\mathbf{X}}^{n+1} - \frac{\Delta t}{2}\underline{\mathbf{u}}^n(\underline{\mathbf{X}}^n), \tag{2.6}$$

$$\underline{\mathbf{X}}_d = \underline{\mathbf{X}}^{n+1} - \Delta t\underline{\mathbf{u}}^{n+\frac{1}{2}}(\underline{\mathbf{X}}^\star), \tag{2.7}$$

where $\underline{\mathbf{u}}^{n+\frac{1}{2}}$ is obtained via extrapolation from previous times, i.e.:

$$\underline{\mathbf{u}}^{n+\frac{1}{2}} = \frac{3}{2}\underline{\mathbf{u}}^n - \frac{1}{2}\underline{\mathbf{u}}^{n-1}. \tag{2.8}$$

Note that all values at the intermediate point, $\underline{\mathbf{X}}^\star$, and departure point, $\underline{\mathbf{X}}_d$, must be calculated via interpolation from the previous grids G^n and G^{n-1}. Here, we use the stabilized second-order interpolation for $\phi(\underline{\mathbf{X}}_d)$ and the multi-linear interpolation for $\underline{\mathbf{u}}^{n+\frac{1}{2}}(\underline{\mathbf{X}}^\star)$ [14]. Although parallelization of the interpolation process on a shared-memory machine is trivial, the same cannot be said for distributed-memory machines. In fact, the parallel interpolation given in Algorithm 3 is probably the most important contribution of this article since this procedure, which is trivial on uniform grids, is challenging in the case of trees because it is not straightforward to identify which processes owns the departure points. Indeed, complications arise because not all departure points will reside in the domain owned by the current process. Moreover, due to the irregular shapes of

the partitions, one cannot even ensure they are entirely owned by neighboring processes. At best we can only expect that their locations are bounded by a halo of width $w \leq \text{CFL}\,\Delta x_{\min}$ around the local partition, where $x_{\min}$ is the size of the smallest cell in the forest. Naturally, if one enforces $\text{CFL} \leq 1$, one can ensure that the halo is bounded by the ghost layer, which significantly simplifies the communication problem. This assumption, however, defeats the purpose of using a semi-Lagrangian approach, whose purpose is to enable large CFL values.

One remedy to this problem, proposed in [50] for uniform grids, is to increase the size of ghost layer to $\lceil \text{CFL} \rceil$. For large values of the CFL number, however, this approach can substantially increase the communication volume. Moreover, this simple approach does not work in the process of generating G^{n+1} due to repartitioning. Indeed, G^{n+1} is built iteratively and load balancing is enforced by repartitioning at each sub-iteration. Therefore, after one such sub-iteration, the backtracked points can end up outside of the initial ghost layer. An alternative approach would be to handle local and remote interpolations separately. Our remote interpolation algorithm is composed of three separate phases. In the first phase, which we call buffering, every process searches for all departure points inside the local trees. If the point is owned by a local cell, it is added to a local buffer, otherwise we find the process which owns the point and add the point to a separate buffer belonging to the found rank. Note that searching the point in the local tree is performed recursively using the hierarchal reconstruction (cf. Algorithm 2). Moreover, the owner's rank is found by computing the Z-index of the point and then using a binary search on the Z-curve. This is already implemented in `p4est` and explained in details in section 2.5 of [44].

Once buffering is done, every process knows exactly how many messages it needs to send and to which processes. This also implicitly defines processes that will later on send a reply message to this process. However, at this point no process knows which

processes to expect a message from. We solve this problem using a simple *communication matrix* (see Figure 2.3). Our approach is very similar to the "Personalized Census ($\mathcal{PCX}$)" algorithm described in [69]. Another similar approach is the "Notify" algorithm introduced in [70]. Furthermore, the MPI-3 standard introduces non-blocking collectives and Remote Memory Access (RMA) operations which enable new ways of solving the communication problem. For instance, authors in [69] descried the "Non-blocking Consensus ($\mathcal{NBX}$)" and "Remote Summation ($\mathcal{RSX}$)" algorithms which make use of such operations and have better theoretical communication complexities. With the exception of $\mathcal{RSX}$ algorithm, which was not tested in this study, all remaining algorithms produced similar timing and scaling. Thus we have decided to describe our algorithm based on the idea of the *communication matrix*.

To solve the communication problem, we first compute the adjacency matrix of the communication pattern, i.e. we construct the matrix $A_{P\times P}$, where P is the number of processes, such that

$$a_{ij} = \begin{cases} 1 & \text{if process '}i\text{' sends a message to process '}j\text{'}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that this matrix is also distributed among processes, i.e. each row is owned by a separate rank. Next, we compute

$$S_i = \sum_j a_{ij} \quad \text{and} \quad R_i = \sum_j a_{ji},$$

where S_i and R_i denote the number of messages sent and received, respectively. While S_i can be computed trivially, a reduction operation is required to compute R_i. For instance, this can be achieved using a single `MPI_Reduce_scatter` function call. The last phase of the interpolation procedure involves overlapping the computation of interpolated values

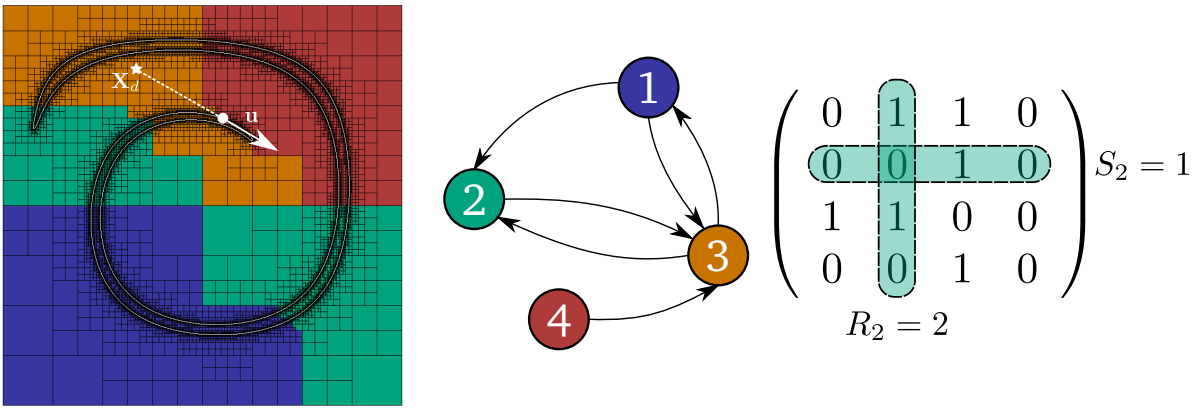


Figure 2.3: Left: the location of back-traced points depends on the magnitude of the local velocity and on the time-step. Although the distance to the departure point is bounded by CFL $\Delta x_{\min}$, one cannot predict the receiving rank without explicitly searching the entire Z-curve. Moreover, the receiving process has no prior knowledge about which processes to check for incoming messages nor does it know anything about the possible message length (i.e. number of points). Middle: a directed graph illustrating the communication pattern among processes with arrows representing the direction in which messages are sent. Right: the adjacency matrix of the communication graph. For each row, the sum of all columns represents the number of messages that need to be sent. Conversely, for each column, the sum of all rows represents the number of messages that need to be received. As detailed in Algorithm 3, this information is enough to build a parallel interpolation scheme.

for local points with the communication of data between processes. This is done by alternating between local calculations and probing for incoming messages from other processes. The interpolation is finished once the values for all local the points have been calculated and all the remote requests have been processed (see Algorithm 3).

Using the interpolation Algorithm 3, we close this section by presenting the final semi-Lagrangian Algorithm 4. The basic idea is to start from an initial guess G_0^{n+1} for the grid and modify it using the refinement (2.3) and coarsening (2.4) criteria until convergence is obtained. Various options are available for G_0^{n+1}. For instance it is possible to start from the macromesh and only perform refinement steps until convergence. This choice, however, is not suitable since the first few iterations do not contain many cells and there is little work for parallelism. Here we simply take the previous grid as the starting point, i.e. $G_0^{n+1} = G^n$. Note that this iterative process is essentially unavoidable since the grid is based on the values of the level-set function at t^{n+1}, which itself is unknown and is to be defined on G^{n+1}. Nonetheless the process converges to the final grid in at most $l_{\max} - l_{\min}$ steps where $l_{\min}$ and $l_{\max}$ denote the maximum and minimum depth of all trees in the forest, receptively.

2.3.3 Reinitialization

Successive application of Algorithm 4, especially for large values of the CFL number, eventually degrades the signed distance property of the level-set function. Thus, it is important to reinitialize the level-set function every few iterations, especially because the quality of generated grid heavily depends on the signed distance property. To achieve this property we solve the pseudo-time transient equation (2.2) using the discretization scheme detailed in [14]. For completeness, we briefly review the scheme. First, we write

```
1: col ← 0, local_buff ← null, remote_buff ← null                    ▷ Phase I – buffering
2: for p in X do
3:     [owners_rank, cell] ← H.search(p)          ▷ search for the owner's rank and cell
4:     if owners_rank = mpirank then                                  ▷ can interpolate locally
5:         local_buff[owners_rank].push_back(p, cell)
6:     else                                                  ▷ requires remote interpolation
7:         remote_buff[owners_rank].push_back(p)
8:         col[owners_rank] ← 1
9:     end if
10: end for
11:
12: for r in [0, mpisize) do   ▷ Phase II – initiate communication and compute number of messages
13:     if col[r] == 1 then
14:         req ← MP_Isend(r, QUERY_TAG, remote_buff[r])  ▷ initiate a non-blocking send to process r
15:         query_requests.push_back(req)
16:     end if
17: end for
18: S ← sum(col)                                    ▷ compute number of messages to send
19: R ← MPI_Reduce_scatter(col, MPI_SUM)   ▷ compute number of messages to receive
20:
21: done ← false                                                  ▷ Phase III – interpolation
22: it ← local_buff[mpirank].begin()
23: while !done do
24:     if it ≠ local_buff[mpirank].end() then
25:         values ← process_local_interpolation(it)   ▷ process local interpolations
26:         ++it
27:     end if
28:     if R > 0 then              ▷ search for interpolation query in the message queue
29:         message ← MPI_Iprobe(MPI_ANY_SOURCE, QUERY_TAG)
30:         if message.is_pending() then
31:             values ← process_remote_queries(message)        ▷ receive, search, and interpolate values
32:             req ← MPI_Isend(message.MPI_SOURCE, REPLY_TAG, values)   ▷ send back interpolated values
33:             reply_requests.push_back(req)
34:             R--
35:         end if
36:     end if
```

```
37:     if S > 0 then                          ▷ search for interpolation reply in the message queue
38:         message ← MPI_Iprobe(MPI_ANY_SOURCE, REPLY_TAG)
39:         if message.is_pending() then
40:             values ← process_replies(message)      ▷ receive remotely interpolated values
41:             S--
42:         end if
43:     end if
44:     done ← S == 0  &  R == 0  &  it == local_buff[mpirank].end()
45: end while
46: MPI_Waitall(query_requests, reply_requests)   ▷ make sure all messages have been received
47: return values
```

Algorithm 3: $values \leftarrow$ `Interpolate` $(H, F, \underline{\mathbf{X}})$: interpolate the value of F, defined on the local tree hierarchy H, at coordinates $\underline{\mathbf{X}}$. This is achieved in three phases as detailed in the text. I) First, interpolation points are buffered in two arrays, those than can be interpolated locally and those that should be sent to other processes. II) In the second phase, each process initiates the remote buffer exchange and computes the number of messages it should expect to receive using the communication matrix idea described in section 2.3.2. III) Finally, each process alternates between computing local interpolation and receiving and performing remote interpolation requests from other processes.

$\Delta t_l \leftarrow \text{CFL} \times G^n\texttt{.hmin()} / \max\{\underline{\mathbf{u}}^n\}$
$\Delta t \leftarrow \texttt{MPI\_Allreduce}(\Delta t_l, \texttt{MPI\_MIN})$ ▷ compute Δt based on CFL condition across all processes
$H^n \leftarrow \texttt{Reconstruct}(G^n)$ ▷ using algorithm 2
$G_0^{n+1} \leftarrow G^n$
while *true* **do**
    $\underline{\mathbf{X}}_d \leftarrow \texttt{ComputeDeparturePoints}(G_0^{n+1}, \underline{\mathbf{u}}^n, \underline{\mathbf{u}}^{n-1}, \Delta t)$ ▷ using equations 2.6 – 2.8
    $\phi^{n+1} \leftarrow \texttt{Interpolate}(H^n, \phi^n, \underline{\mathbf{X}}_d)$ ▷ using algorithm 3
    $G^{n+1} \leftarrow G_0^{n+1}\texttt{.refine\_and\_coarsen}(\phi^{n+1})$ ▷ using equations 2.3 and 2.4 as criteria
    if $G^{n+1} \neq G_0^{n+1}$ **then**
        $G^{n+1}\texttt{.partition()}$
        $G_0^{n+1} \leftarrow G^{n+1}$
    else
        break
    end if
end while
return $[G^{n+1}, \phi^{n+1}]$

Algorithm 4: $[G^{n+1}, \phi^{n+1}] \leftarrow \texttt{SemiLagrangian}\ (G^n, \phi^n, \underline{\mathbf{u}}^n, \underline{\mathbf{u}}^{n-1}, \text{CFL})$: update ϕ^{n+1} from ϕ^n using a semi-Lagrangian scheme and construct the new forest G^{n+1} that is consistent with the zero level-set of ϕ^{n+1}

equation (2.2) in the following semi-discrete form:

$$\frac{\mathrm{d}\phi}{\mathrm{d}\tau} + S(\phi_0)\left(\mathcal{H}_G(D_i^+\phi, D_i^-\phi) - 1\right) = 0, \tag{2.9}$$

where $D_i^+\phi$ and $D_i^-\phi$ are the forward and backward derivatives in the x_i direction and $\mathcal{H}_G$ is the Godunov Hamiltonian defined as:

$$\mathcal{H}_G(a_i, b_i) = \begin{cases} \sqrt{\sum_i \max\left(|a_i^+|^2, |b_i^-|^2\right)} & \text{if} \quad S(\phi_0) \leq 0, \\ \\ \sqrt{\sum_i \max\left(|a_i^-|^2, |b_i^+|^2\right)} & \text{if} \quad S(\phi_0) > 0, \end{cases}$$

where $a^+ = \max(a, 0)$ and $a^- = \min(a, 0)$. Similar to [14], equation (2.9) is integrated in time using the TVD-RK2 scheme with adaptive time-stepping in order to accelerate the convergence to the steady state. Generally it has been observed that adaptive time-stepping considerably improves the convergence rate and only a few iterations are needed if the signed-distance property is desired in a small band around the interface [14]. In this work, and based on previous findings, we use a fixed number of 20 iterations for the reinitialization equation. Of course, it should be noted that it is quite easy to define a custom tolerance as the termination criteria.

Since the computation is based on a local stencil, the parallel implementation of this scheme is mostly trivial. However, one minor point requires further explanation. As suggested in [14], one-sided derivatives $D_i^+\phi$ and $D_i^-\phi$ are computed using second order discretization which require to compute the second-order derivatives. To enable overlap between computation and communications when computing second-order derivatives and also integrating equation (2.9), we use the following common technique. First, we label all local points, L_p, as either private, P_p, or boundary, B_p. Here, the boundary points are the collection of all local points that are regarded as a ghost point, G_r, on at least one

other process, i.e. $B_p = \bigcup_{r,\, r \neq p} G_r$. Private points are defined as the collection of all local points that are not a boundary point, i.e. $P_p = L_p \setminus B_p$. Algorithm 5 illustrates how this labeling can help with overlapping the computation and the communication associated to an arbitrary local operation $y \leftarrow \mathcal{F}(x)$. Note that the `p4est` library already includes all the primitives required for labeling local points without any further communication.

1: **for** $i : B_p$ **do** ▷ I – perform computation on boundary points
2: $y_i \leftarrow \mathcal{F}(x_i)$
3: **end for**
4: *send_req* $\leftarrow$ `MPI_Isend`(y_B) ▷ II – begin updating ghost values
5: *recv_req* $\leftarrow$ `MPI_Irecv`(y_G)
6: **for** $i : P_p$ **do** ▷ III – perform computation on private points
7: $y_i \leftarrow \mathcal{F}(x_i)$
8: **end for**
9: `MPI_Waitall`(*send_req*, *recv_req*) ▷ IV – wait for ghost update to finish
10: **return** y

Algorithm 5: $y \leftarrow$ `Overlap` $(x, \mathcal{F})$: compute $y_i = \mathcal{F}(x_i)$ for all nodes i, where $\mathcal{F}$ is a local operation, while hiding the communication to update the ghost layer

2.3.4 Accuracy

The numerical methods detailed in the previous sections are widely used in the level-set community and their accuracy is studied for example in [14]. However, in order to validate our implementation, we present a brief convergence analysis.

The advection of an irregular boundary using the semi-Lagrangian method is the perfect candidate to demonstrate the accuracy of our implementation as it makes use of the interpolation routine as well as the reinitialization procedure. We select the benchmark problem proposed in [29]. Consider a sphere centered at $(0.35, 0.35, 0.35)$ and with radius

0.15 in a domain $[0,1]^3$ and deformed under the divergence free velocity field

$$\begin{cases} u(x,y,z) = 2\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z), \\ v(x,y,z) = -\sin^2(\pi y)\sin(2\pi x)\sin(2\pi z), \\ w(x,y,z) = -\sin^2(\pi z)\sin(2\pi x)\sin(2\pi y). \end{cases} \tag{2.10}$$

forward in time until $t_{1/2} = 1$ and then backward to its original state at $t_f = 2$. We set the time step to $\Delta t = 5\Delta x_{min}$, where Δx_{min} is the size of the smallest cell in the forest. The level-set function is reinitialized at every time step by applying 20 iterations of the reinitialization procedure. We monitor the volume loss and the error in the interface location at the final time, when the original shape should be recovered, as the finest resolution of the forest increases. The results are reported in table 2.1 and figure 2.4 shows a visualization of the sphere at the initial time, at $t_{1/2}$ when the deformation is maximal, and at the final time. The results are consistent with those reported previously in [14] and indicate second order accuracy for the mass loss.

Finest resolution	L^∞ error on ϕ	Rate	Volume loss (%)	Rate
128^3	1.53E-01	-	1.85E-01	-
256^3	1.18E-01	0.38	3.73E-02	2.31
512^3	9.82E-03	3.58	6.87E-03	2.44
1024^3	2.56E-03	1.94	1.92E-03	1.84
2048^3	1.60E-03	0.67	5.17E-04	1.89

Table 2.1: Study of the convergence of the level-set algorithm using Enright's test [29]. The L^∞ error on ϕ is computed only close to the interface as this is the relevant observable for an advection procedure.

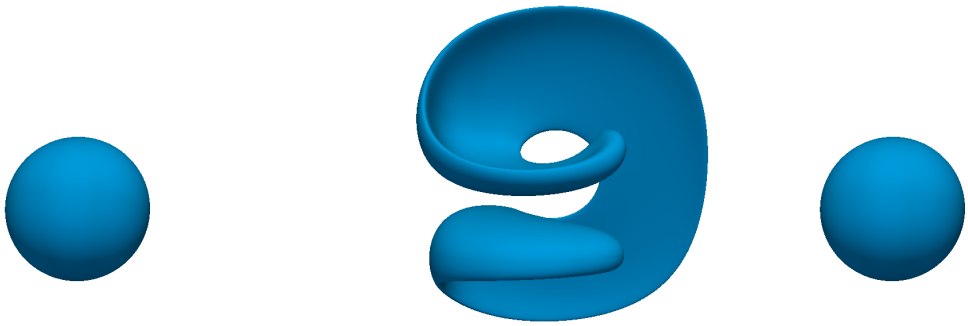


Figure 2.4: Visualization of the deformation undergone by the sphere for the Enright's test at time 0, $t_{1/2}$ and t_f. The forest's finest resolution is equivalent to a uniform grid with 2048^3 cells and contains between 17 (at initial and final times) and 63 (at $t_{1/2}$) million nodes. These results were obtained on the Comet supercomputer with 960 processes (40 compute nodes).

2.4 Scaling results

In this section we present some results that demonstrate the scalability of our algorithms. All of our tests were ran on the Stampede cluster at the Texas Advanced Computing Center (TACC), a resource accessible through the Extreme Science and Engineering Discovery Environment (XSEDE) [71], where we are limited to 4096 cores at most. Each node of Stampede has 2 eight-core Xenon E5-2680 processes clocked at 2.7 GHz with 32 GB of DDR3-1600 MHz memory and interconnected using an InfiniBand network card. Unless mentioned otherwise, in all the tests we have used all 16 cores of every node. Finally, in all cases we report the maximum wall time recorded using PETSc's logging interface which has a temporal resolution of roughly 0.1 μs.

We define parallel efficiency as $e = s \cdot P_1/P$ where $s = t_1/t_P$ is the speed-up, P_1 is the smallest number of processes for which the test was run, t_1 is the time to run the problem on P_1 processes, P is the number of processes and t_P is the time to run the problem on P processes. We note that efficiencies larger than 100% are reported for some cases. This is common and can be hardware related, for instance linked to the problem being locally

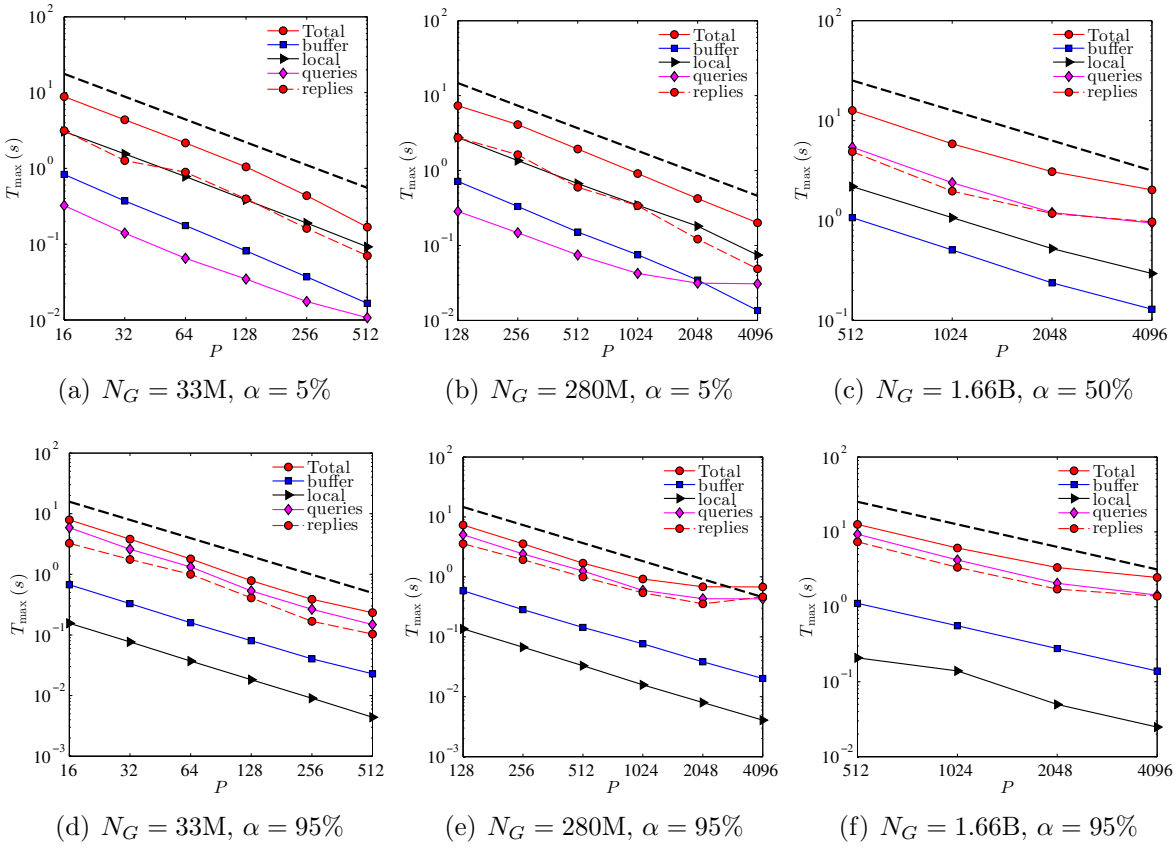
smaller for larger number of processes and thus exploiting the cache better.

2.4.1 Interpolation

In this section we show the results for a simple test to measure the scalability of the interpolation Algorithm 3. The test consists of interpolating a function at a number of random points on a randomly refined Octree in three spatial dimensions. We consider two cases, a small test on a level[1] 9 tree with roughly 33M nodes and a larger test on a level 13 tree with roughly 280M nodes. In both cases the number of randomly generated points is chosen to be equal to the number of nodes and the stabilized second-order interpolation of [14] is performed 10 times to smooth out possible timing fluctuations.

To simulate the effect of different CFL numbers, we generate the random points such that on each process α percentage of them are located outside the process boundary and thus will initiate communication. Scaling results are presented for $\alpha = 5\%$ and $\alpha = 95\%$ for both the small and large problems in Figure 2.5. We also present a third row of results for a much larger problem with roughly 1.66B nodes on a level 14 tree. Excellent scaling is obtained for the small problem for $P = 16 - 512$ even when 95% of the interpolation points belong to a remote process. For the larger problem, however, the communication overhead prevents the algorithm from scaling beyond 2048 processes when $\alpha = 95\%$ (cf. Table 2.2). Note, however, that this is expected since the total time is dominated by communication for $\alpha = 95\%$ and there is very little local work in this case. Indeed, the last row of Figure 2.5 show much better scaling behavior on a larger problem size, e.g. efficiencies are increased from $e = 34\%$ to $e = 68\%$ for $\alpha = 95\%$ on 4096 processes. This is a typical result with strong scaling and simply implies that our algorithms are scalable for sufficiently large problems.

[1]The level is the number of recursive splits allowed for each tree.



(a) $N_G = 33\text{M}$, $\alpha = 5\%$ (b) $N_G = 280\text{M}$, $\alpha = 5\%$ (c) $N_G = 1.66\text{B}$, $\alpha = 50\%$

(d) $N_G = 33\text{M}$, $\alpha = 95\%$ (e) $N_G = 280\text{M}$, $\alpha = 95\%$ (f) $N_G = 1.66\text{B}$, $\alpha = 95\%$

Figure 2.5: Strong scaling of Algorithm 3 for several tests where N_G denotes the number of random interpolation points (which is the same as the number of nodes in the Octree) and α denotes the percentage of these points that are remote for each process. Here "Total" represents the total time spent in the interpolation while "buffer", "local", "queries", and "replies" represent the the timing for different sections (cf. Algorithm 3). The black dashed line represents the ideal scaling. The results indicate excellent scaling for the small test (a-d) and for the large test when $\alpha = 5\%$ (b). For the extreme case (e) the algorithm stops scaling at 2048 processes due to communication overhead. Note, however, that this merely indicates that the problem size is not large enough for this test case. Indeed much better scaling is obtained when the problem size is increased to $N_G = 1.66\text{B}$ points (c-f).

Small Test	P	16	32	64	128	256	512
	$\alpha = 5\%$	100%	101%	102%	106%	127%	165%
	$\alpha = 95\%$	100%	104%	110%	125%	127%	106%
Large Test	P	128	256	512	1024	2048	4096
	$\alpha = 5\%$	100%	89%	95%	101%	109%	114%
	$\alpha = 95\%$	100%	103%	108%	99%	67%	34%
Very Large Test	P	128	256	512	1024	2048	4096
	$\alpha = 50\%$	–	–	100%	108%	102%	78%
	$\alpha = 95\%$	–	–	100%	103%	94%	64%

Table 2.2: Parallel efficiency of the total runtime of the interpolation algorithm for the small (33M nodes), large (280M nodes), and very large (1.66B nodes) tests. Reported efficiencies are based on the lowest number of processes for each test.

2.4.2 Semi-Lagrangian

To test the scalability of the semi-Lagrangian scheme of Algorithm 4, we consider a slightly modified version of the Enright's rotation test [29] presented in section 2.3.4, i.e. we advect a sphere of radius 0.35 located at $(0.4, 0.4, 0.4)$ with a divergence free velocity field given by equation (2.10).

To understand the effect of the CFL number on the scalability of the algorithm we perform one step of the semi-Lagrangian algorithm for CFL = 1, CFL = 10, and CFL = 100. We also perform the test for two different initial girds, a small grid with maximum level $l_{\max} = 10$ and a large grid with maximum level $l_{\max} = 12$. In both cases, the minimum level is $l_{\min} = 0$. After one advection step, these grids have approximately 15M and 255M nodes, respectively.

Unlike many existing applications where the mesh is changed infrequently, our semi-Lagrangian algorithm requires several sub-iterations of the refinement and coarsening operations. As a result, it is expected that refinement and coarsening steps constitute a significant portion of the total runtime which puts stringent scalability requirements on these algorithms. We refer the interested reader to section 3.2 of [44] for detailed description of scalable refinement and coarsening algorithms in `p4est`.

CFL \ #p	16	32	64	128	256	512
1	2	2	3	3	3	3
10	3	3	3	3	3	3
100	6	6	6	6	6	6

(a) $l_{\max} = 10$

CFL \ #p	128	256	512	1024	2048	4096
1	3	3	3	3	3	3
10	3	3	4	4	4	4
100	6	6	6	6	6	7

(b) $l_{\max} = 12$

Table 2.3: Number of sub-iterations required for the grid construction in Algorithm 4 for the rotation test on a (a) level-10 and (b) level-12 Octree with approximately 15M and 255M nodes, respectively. Note how the sub-iteration count increases with the CFL number but is almost independent of the number of processes. The slight dependence between the number of sub-iterations and the number of processes is most likely due to the dependence of round-off errors on the number of processes. Nonetheless, close examination of the Octrees generated (data not shown) reveals that they are identical and independent of the number of processes used to perform the test.

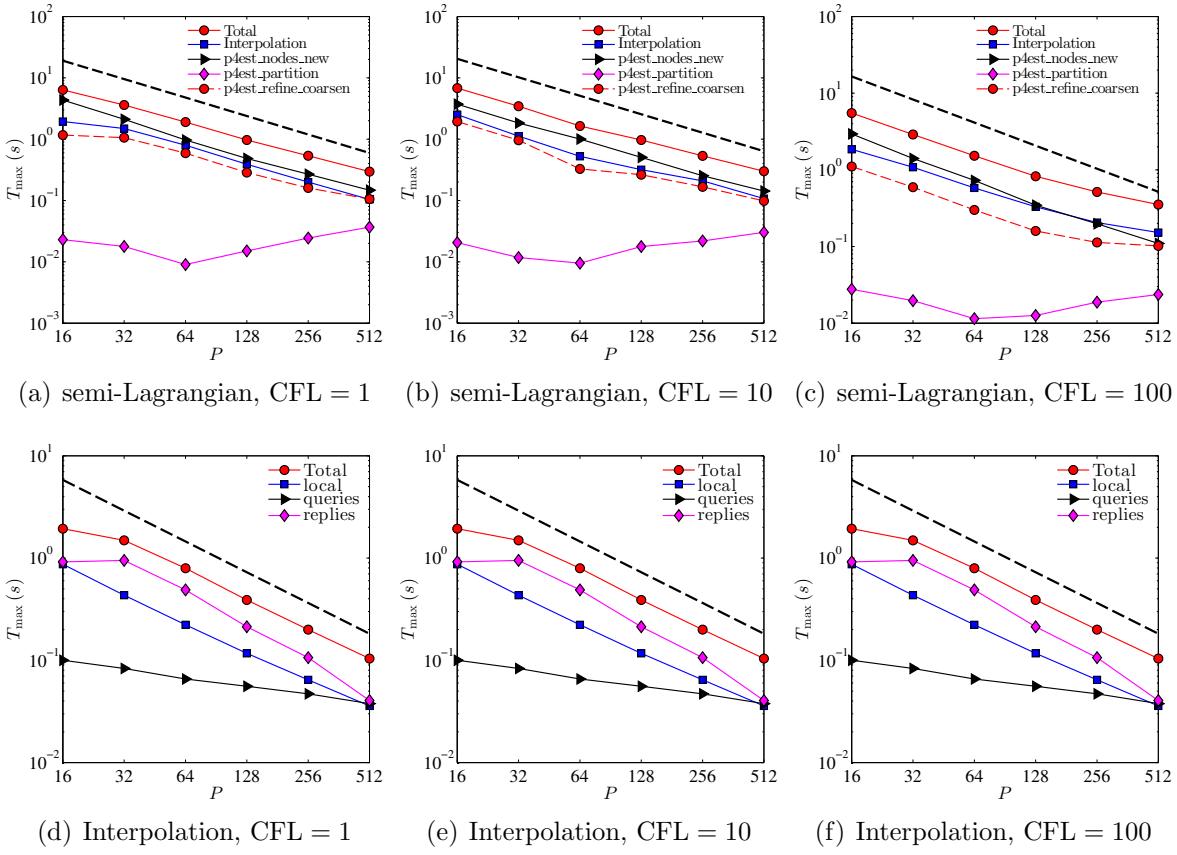
Table 2.3 illustrates the dependence of the number of sub-iterations required to build the grid on the CFL number; as the CFL is increased, the interface travels a farther distance, which necessitates more sub-iterations to generate the grid. Figures 2.6 and 2.7 illustrate the scalability of the algorithm for the small and large problems, respectively. To enable meaningful comparisons between different CFL numbers and number of processes, the maximum time has been scaled by the number of sub-iterations required for the grid construction as reported in Table 2.3. For both problems, excellent scalability is observed for CFL = 1 and CFL = 10. The algorithm even shows good scalability when taken to the extreme, i.e. for CFL = 100.

An increase in the CFL number has two effects on the algorithm. First, a larger fraction of the departure points lands in the domains of remote processes. Moreover, these points are potentially dispersed across a larger number of processes. This means that the communication volume should increase with the CFL number. Second, as more points are shipped to remote processes for interpolation, there is a greater chance that the interpolation load is imbalanced across processes. This is especially true for regions of space in which the streamlines cluster. Both factors can contribute to reducing the scalability of the algorithm at large CFL numbers.

	P	16	32	64	128	256	512
Small Test	CFL = 1	100%	88%	84%	82%	74%	67%
	CFL = 10	100%	99%	104%	88%	80%	71%
	CFL = 100	100%	95%	90%	84%	67%	49%
	P	128	256	512	1024	2048	4096
Large Test	CFL = 1	100%	94%	87%	82%	75%	65%
	CFL = 10	100%	90%	92%	86%	79%	63%
	CFL = 100	100%	94%	90%	84%	70%	57%

Table 2.4: Parallel efficiency of the runtime of a single semi-Lagrangian step. Reported efficiencies are based on the lowest number of processes for each test.

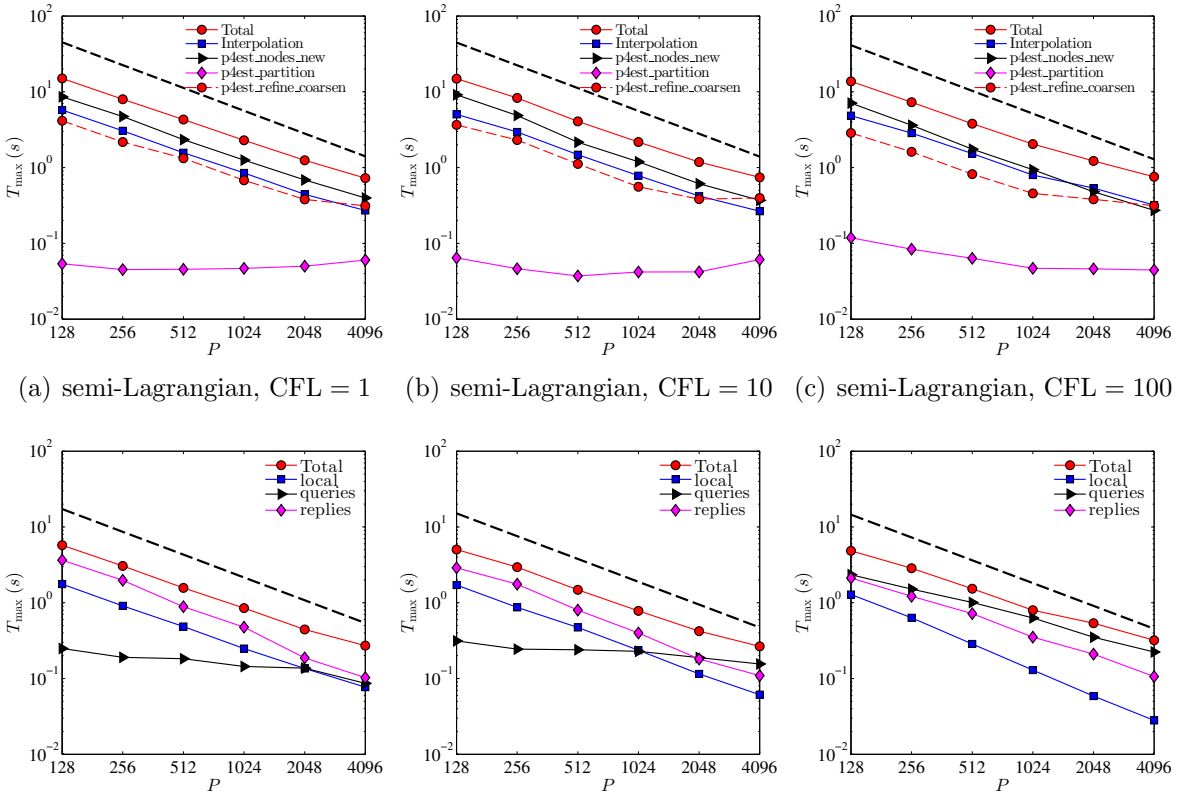
To better understand the importance of the CFL number on the scalability, we have



(a) semi-Lagrangian, CFL = 1 (b) semi-Lagrangian, CFL = 10 (c) semi-Lagrangian, CFL = 100

(d) Interpolation, CFL = 1 (e) Interpolation, CFL = 10 (f) Interpolation, CFL = 100

Figure 2.6: Strong scaling of a single time step of Algorithm 4 for the rotation test on a level-10 Octree with approximately 15M nodes. Top row: scaling of the various components of the algorithm for (a) CFL = 1, (b) CFL = 10, and (c) CFL = 100. Bottom row: breakdown of the various components of the interpolation phase for the same CFL numbers. The solid dashed line represents the ideal scaling. Note that the maximum time has been scaled by the number of sub-iterations required to build the tree (cf. Table 2.3). Here `p4est_nodes_new`, `p4est_partition` and `p4est_refine_coarsen` refer to constructing the global indexing for nodes, partitioning the forest, and the refining/coarsening operation, respectively [44].



(a) semi-Lagrangian, CFL = 1 (b) semi-Lagrangian, CFL = 10 (c) semi-Lagrangian, CFL = 100

(d) Interpolation, CFL = 1 (e) Interpolation, CFL = 10 (f) Interpolation, CFL = 100

Figure 2.7: Strong scaling of a single time step of Algorithm 4 for the rotation test on a level-12 Octree with approximately 255M nodes. Top row: scaling of the various components of the algorithm for (a) CFL = 1, (b) CFL = 10, and (c) CFL = 100. Bottom row: breakdown of the various components of the interpolation phase for the same CFL numbers. The solid dashed line represents the ideal scaling. Note that the maximum time has been scaled by the number of sub-iterations required to build the tree (cf. Table 2.3).

recorded a complete history of the communication pattern in the interpolation step. Figure 2.8 illustrates the effects of the CFL number on different metrics, namely the number of interpolation points[2], N_p, the number of sent and received messages, $N_m = S + R$, and the total communication volume, V_m in megabytes (MB), for $p = 4096$ processes. Furthermore, these values are reported for the first (top row) and last (bottom row) sub-iterations of the semi-Lagrangian algorithm. There are several points to make. First, increasing the CFL number greatly increases the load imbalance, as shown by the spread of the data in Figure 2.8(a). This is because at higher CFL numbers, it is more likely that some processes will receive a larger portion of the backtracked points. Second, increasing the CFL number increases both the communication volume and its spread across processes (cf. Figure 2.8(c)). Interestingly, however, the number of sent and received messages do not seem to be affected by the CFL number. The bottom row of Figure 2.8 exhibits a better balance both in the computation and communication volume in the last sub-iteration of the semi-Lagrangian algorithm. This can be justified by noting that for the final sub-iteration, the partitioning of G^{n+1} is more consistent with the partitioning of the departure points on G^n. Detailed information about the load balancing and the communication patterns is listed in Table 2.5.

2.4.3 Reinitialization

Finally we present the scaling results of our parallel reinitialization algorithm where we extensively make use of Algorithm 5 for overlapping the computations with the communications when computing spatial derivatives. Our test consists in computing the signed distance function to a collection of 100 spheres, whose radii and centers are chosen randomly. The test is performed on a small, level-8 Octree with about 21M and a larger, level-10 Octree with about 337M grid points. In both cases the forest is built on

[2]Note that this includes both the local points and the points queried by other processes.

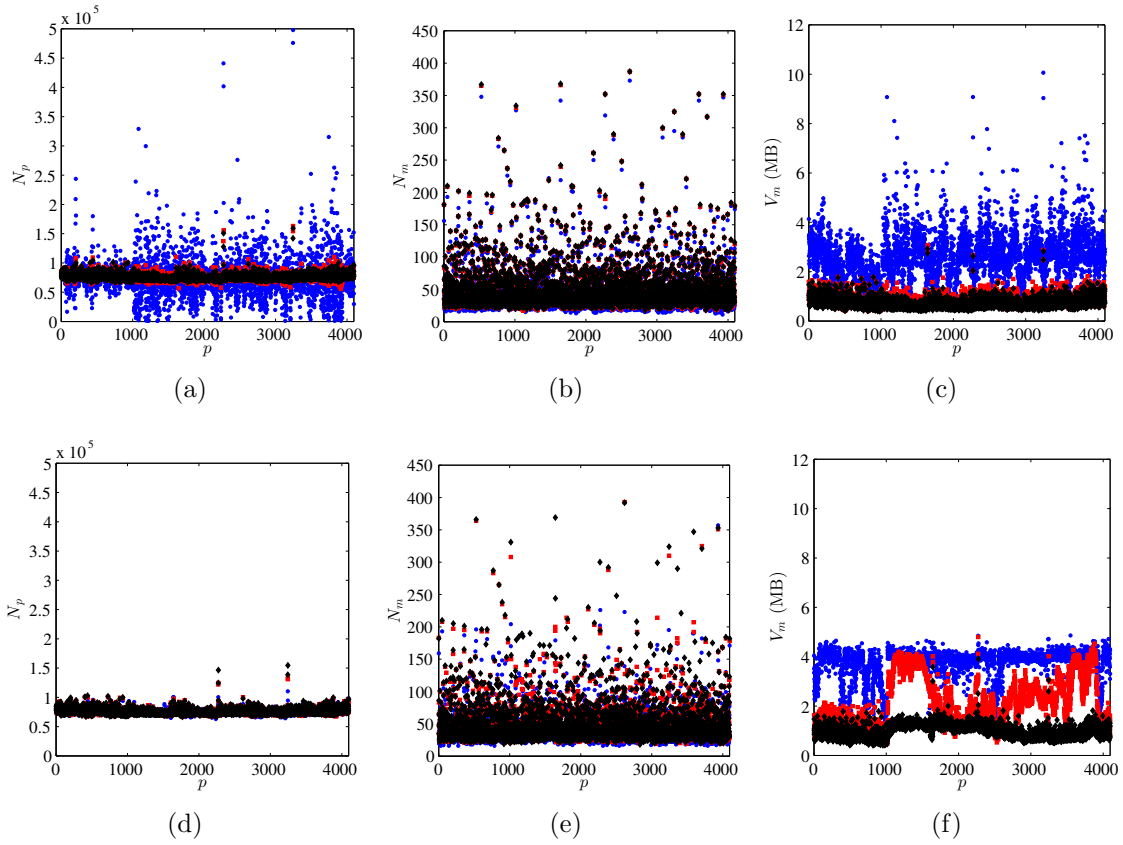


Figure 2.8: Performance indicators of the first (top row) and last (bottom row) sub-iterations of the semi-Lagrangian algorithm for the level-set advection on 4096 processes with CFL = 1 (◆), CFL = 10 (■), and CFL = 100 (●). Increasing the CFL number causes load imbalance during interpolation (a) and increases the communication volume (c). However, the CFL number does not seem to appreciably affect the number of messages sent by the processes (b). During the last semi-Lagrangian sub-iteration, the initial grid G_0 (cf. Algorithm 4) is very close to the final grid. As a result, the load imbalance is considerably improved (d). Curiously, however, the communication pattern does not seem to be change much between first and last sub-iterations (e,f).

CFL	Metric	First sub-iteration				Last sub-iteration			
		min	max	avg	stddev	min	max	avg	stddev
1	N_p	6.67E+04	1.59E+05	7.57E+04	4.86E+03	6.69E+04	1.55E+05	7.57E+04	4.73E+03
	N_m	18	387	47.55	31.72	18	392	47.55	31.19
	V_m (MB)	4.01E-01	2.93E+00	7.25E-01	1.95E-01	4.13E-01	3.91E+00	9.68E-01	2.62E-01
	$T_{\max}$ (s)	6.96E-01				3.67E-01			
10	N_p	5.56E+04	1.63E+05	7.57E+04	6.07E+03	6.65E+04	1.33E+05	7.57E+04	4.50E+03
	N_m	17	387	46.73	31.78	19	393	46.68	27.93
	V_m (MB)	4.01E-01	3.06E+00	8.40E-01	2.21E-01	4.40E-01	4.80E+00	2.14E+00	9.88E-01
	$T_{\max}$ (s)	7.55E-01				3.30E-01			
100	N_p	8.28E+02	4.98E+05	7.57E+04	3.14E+04	6.30E+04	1.10E+05	7.57E+04	4.20E+03
	N_m	11	373	41.18	30.85	16	357	41.77	22.66
	V_m (MB)	5.28E-01	1.01E+01	2.65E+00	9.24E-01	9.76E-01	4.86E+00	3.78E+00	5.69E-01
	$T_{\max}$ (s)	9.25E-01				3.55E-01			

Table 2.5: Detailed load balancing and communication information for the advection test for CFL = 1, CFL = 10, and CFL = 100. Here N_p is the number of interpolation points, $N_m = S + R$ is the number of sent (S) and received (R) messages, and V_m is the total communication volume in megabytes (MB). Note how increasing the CFL number causes load imbalance and increases the communication volume while it does not affect the number of messages sent and received during a sub-iteration of the semi-Lagrangian step.

a $3 \times 3 \times 3$ macro-mesh. Figure 2.9 illustrates that our reinitialization algorithm, and in particular the overlapping strategy presented in Algorithm 5, scales very well (cf. Table 2.6). In general we expect similar scaling results for any local, finite-difference based calculations on Octrees that can efficiently utilize Algorithm 5.

Small Test	P	16	32	64	128	256	512
	e	100%	115%	110%	105%	96%	80%
Large Test	P	128	256	512	1024	2048	4096
	e	100%	95%	95%	89%	82%	67%

Table 2.6: Parallel efficiency of the total runtime for the reinitialization test based on the lowest number of processes for each test.

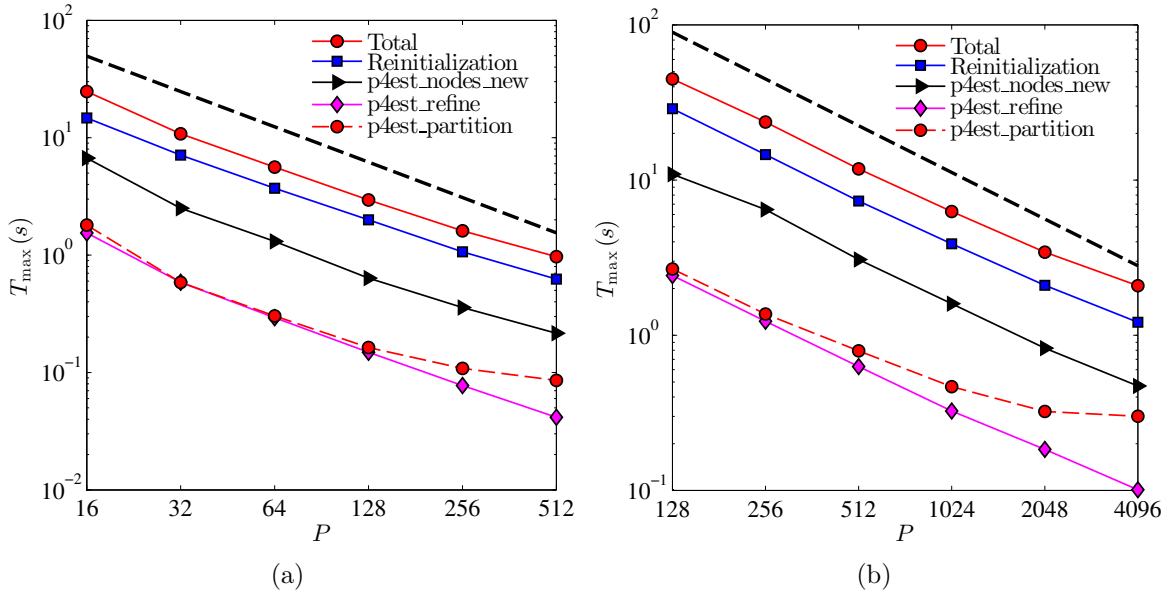


Figure 2.9: Scalability of the reinitialization test for a small (left) and large (right) Octree with roughly 21M and 337M grid points, respectively. The black dashed line represents ideal scaling. Excellent results are obtained in both cases, illustrating the scalability of the overlapping strategy (cf. Algorithm 5).

2.5 Application to the Stefan problem

2.5.1 Presentation of the problem

In this section we apply our approach to the study of the phase transition of a liquid melt to a solid crystaline structure. In the case of a single component melt, and in the absence of convection, the process is dominated by diffusion and can be modeled as a Stefan problem. We decompose the computational domain Ω into two subdomains Ω_l and Ω_s, separated by an interface Γ. The Stefan problem describes the evolution of the temperature T, decomposed into T_s in the solid phase Ω_s and T_l in the liquid phase Ω_l,

as

$$\frac{\partial T_l}{\partial t} = D_l \Delta T_l \quad \text{in} \quad \Omega_l, \tag{2.11}$$

$$\frac{\partial T_s}{\partial t} = D_s \Delta T_s \quad \text{in} \quad \Omega_s. \tag{2.12}$$

The diffusion constants D_l and D_s can be discontinuous across the interface. We prescribe homogeneous Neumann boundary conditions on the edge of the computational domain, $\nabla T \cdot \underline{\mathbf{n}}|_{\partial\Omega} = 0$. At the interface between the solid and the liquid phases, the temperature is given by the Gibbs-Tompson boundary condition [72, 73]:

$$T_s = T_l = T_\Gamma = -\epsilon_c \kappa - \epsilon_v (\underline{\mathbf{u}} \cdot \underline{\mathbf{n}}), \tag{2.13}$$

where κ is the local interface curvature, $\underline{\mathbf{u}}$ is the velocity of the interface, $\underline{\mathbf{n}}$ is the outward normal to the solidification front and ϵ_c and ϵ_v are the surface tension and kinetic undercooling coefficients. The interface velocity $\underline{\mathbf{u}}$ is defined from the jump in the heat flux at the interface,

$$(\underline{\mathbf{u}} \cdot \underline{\mathbf{n}}) = -\left[D_l \frac{\partial T_l}{\partial \underline{\mathbf{n}}} - D_s \frac{\partial T_s}{\partial \underline{\mathbf{n}}} \right]. \tag{2.14}$$

We choose to use an adaptive time step with a CFL = 5, i.e.

$$\Delta t = 5 \ \Delta x_{min} \ \min(1, 1/\max\|\underline{\mathbf{u}}\|), \tag{2.15}$$

where Δx_{min} is the size of the smallest cell of the forest. The general procedure to solve the Stefan problem is presented in Algorithm 6 and we refer the interested reader to [74] for the details of implementation. In implementing the numerical solver, we make use of the popular `PETSc` [75] library for linear algebra and its parallel primitives, such as parallel ghosted vector and scatter/gather operations, which simplifies the implementation.

1: Initialize the forest and ϕ given the initial geometry.
2: Initialize T_s in Ω^+ and T_l in Ω^-.
3: Reinitialize ϕ and compute the local interface curvature κ.
4: Compute T_l^{n+1} and T_s^{n+1} by solving the heat equations (2.11) and (2.12).
5: Extrapolate T_s^{n+1} from Ω^+ to Ω^- and T_l^{n+1} from Ω^- to Ω^+.
6: Compute the velocity field $\underline{\mathbf{u}}$ according to (2.14).
7: Compute the time step dt following (2.15).
8: Evolve the interface and construct the new forest using the Semi-Lagrangian procedure.
9: Interpolate T_s^{n+1} and T_l^{n+1} from the old forest to the new forest.
10: Go to 3 with n=n+1.

Algorithm 6: `General procedure for solving the Stefan problem`

2.5.2 Scalability

The implementation of the Stefan problem relies on the components described in the previous sections, and it is therefore a good synthesis of the performance of the various algorithms. We monitored the performance of the code over five time iterations, as presented in Algorithm 6, for two different maximum resolutions. In both cases, the forest is built on a $20 \times 20 \times 20$ macro-mesh. The maximum tree resolution for the small test is 9, leading to approximately 7M grid points, and the maximum resolution for the large test is 11, corresponding to 105M grid points. The level-set function is reinitialized at every time-step by applying 20 iterations of the reinitialization procedure. The results are presented in figure 2.10, where "`Solution_Extension`" refers to extrapolation procedure (see Algorithm 6 step 5). As expected from the results obtained for each component in the previous sections, our implementation of the Stefan problem exhibits very satisfactory scaling (cf. Table 2.7).

2.5.3 Numerical experiments

We now present the results from a large simulation of the Stefan problem on a $20\times20\times$ 20 macro-mesh and with level-10 Octrees. The Gibbs-Tompson anisotropy undercooling

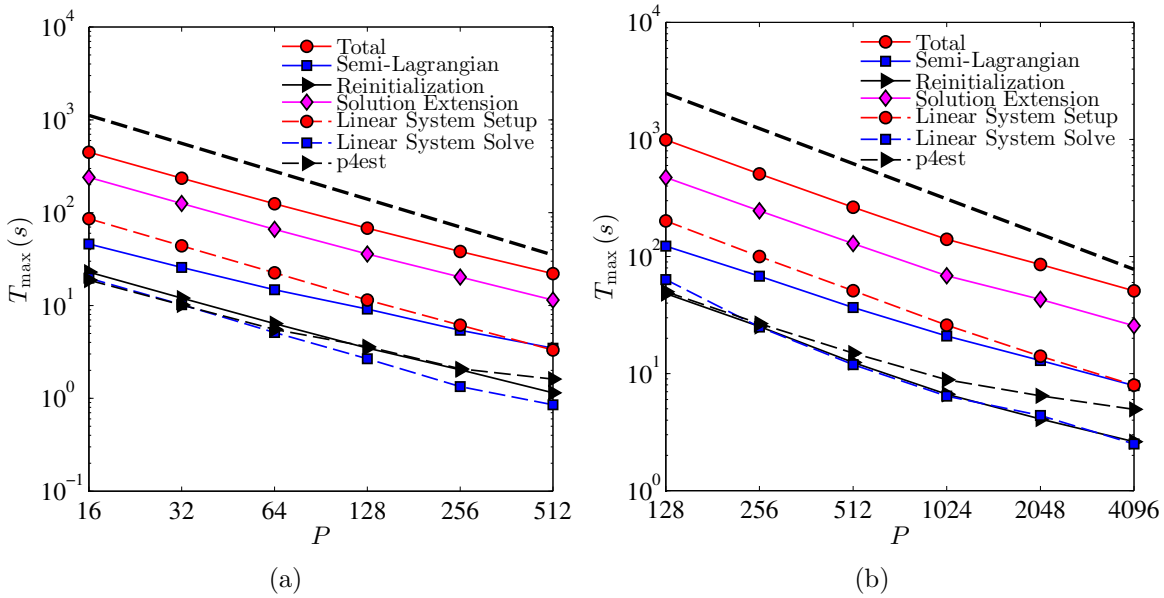


Figure 2.10: Scalability of the Stefan problem for small (left) and large (right) Octrees with roughly 7M and 105M grid points, respectively. The solid dashed line represents perfect scaling. As expected from the scalability analysis of the individual components, we observe excellent results, illustrating the potential of our algorithms.

Small Test	P	16	32	64	128	256	512
	e	100%	95%	90%	82%	73%	64%
Large Test	P	128	256	512	1024	2048	4096
	e	100%	98%	94%	89%	73%	61%

Table 2.7: Parallel efficiency of the total runtime for the Stefan test based on the lowest number of processes for each test.

coefficients in equation (2.13) are defined as

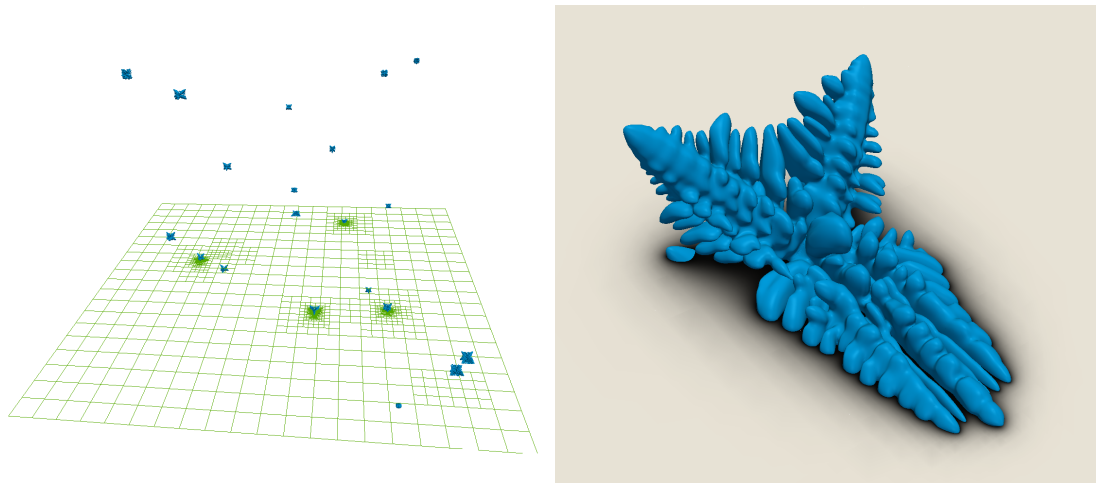
$$\epsilon_c = \left[\epsilon_1 \left(1 + \alpha_1 \cos(3\theta_1)\right) + \epsilon_2 \left(1 + \alpha_2 \cos(3\theta_2)\right)\right] \kappa,$$
$$\epsilon_v = 0,$$

with θ_1 the angle between the normal to the interface $\underline{\mathbf{n}}$ and the x-axis in the (x, y) plane and θ_2 the angle between $\underline{\mathbf{n}}$ and the x-axis in the (x, z) plane. The coefficients

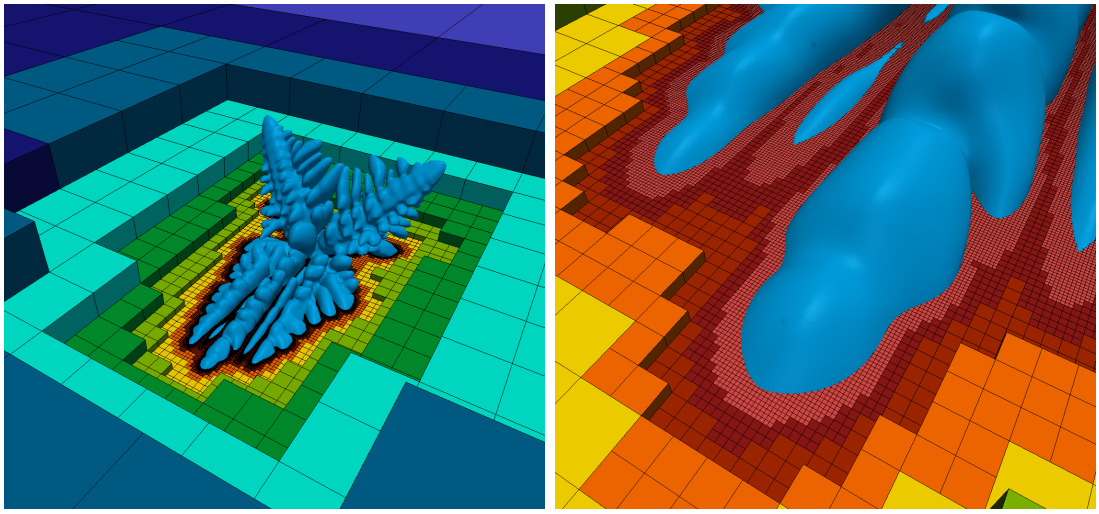
$$\epsilon_1 = 2\ (\sin(x) + \cos(y) + 2) \cdot 10^{-6}, \qquad \epsilon_2 = 2\ (\sin(x) + \cos(z) + 2) \cdot 10^{-6},$$
$$\alpha_1 = \frac{1}{4}(\cos(x) + \sin(y) + 2), \qquad \alpha_2 = \frac{1}{4}(\cos(x) + \sin(z) + 2),$$

are used to enforce a variety of crystal shapes. The computation is initialized with twenty spherical seeds of radius $1.5 \cdot 10^{-3}$ placed randomly in the domain. We take the diffusion coefficients $D_s = D_l = 1$ and set the initial temperatures $T_l^0 = -0.25$ and $T_s^0 = 0$.

The simulation was ran on 256 MPI processes for 6 hours and 30 minutes, resulting in 396 time iterations. Visualizations of the final iteration are presented in figures 2.11 and 2.12. The final iteration of the simulation consisted of 167M grid points whereas a uniform grid with the equivalent finest resolution would lead to $8.59 \cdot 10^{12}$ grid points, i.e. over eight trillion grid points. Our simulation used only 0.002% of the number of grid points needed for the same simulation on a uniform grid. This application demonstrates the ability of our approach to resolve small scale details, while accounting for long range interactions.



(a) (b)



(c) (d)

Figure 2.11: Visualization of the computational mesh (a, c, d) and the temperature field (b) for the Stefan problem simulation.

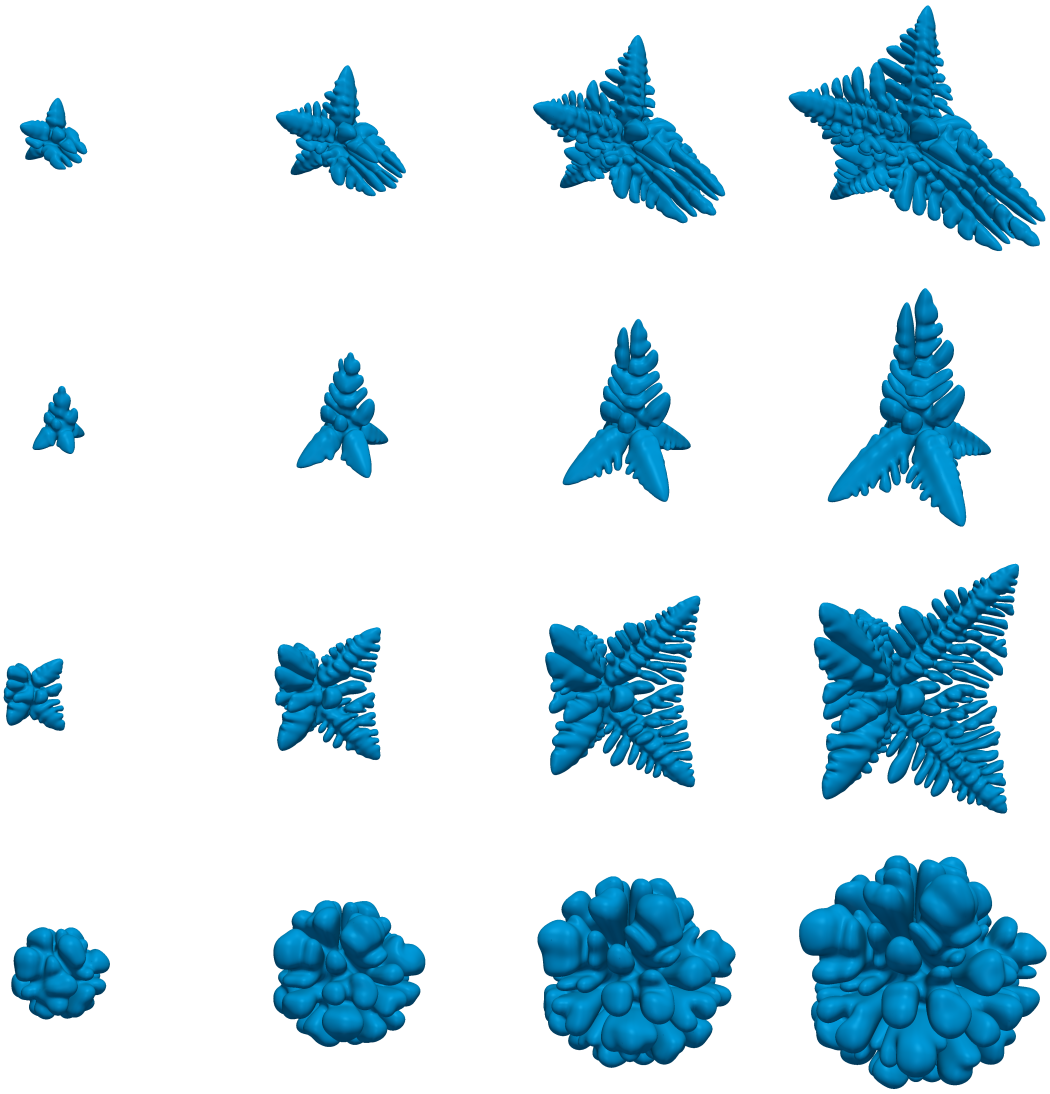


Figure 2.12: Time evolution of four of the crystals obtained for the Stefan problem simulation. The snapshots represent, from left to right, iterations 96, 196, 296 and 396.

2.6 Summary

In this article we have presented parallel algorithms related to the level-set technology on adaptive Quadtree and Octree grids using a domain decomposition approach. These algorithms are implemented using a combination of `MPI` and the open-source `p4est` library. In order to preserve the unconditional stability property of the semi-Lagrangian scheme while enabling scalable computations, we introduced an asynchronous interpolation algorithm using non-blocking point-to-point communications, and demonstrated its scalability.

In particular we showed that the scalability of the semi-Lagrangian algorithm, depends on the CFL number. Great scalability is observed for intermediate CFL numbers, e.g. CFL ~ 10. At higher CFL numbers, however, the departure points are potentially further dispersed across processors, which limits the scalability. This is because the domain decomposition technique used here is based on the Z-ordering of cells and does not take the velocity field information into account. A possible remedy for this problem could be assigning weights to cells based on some estimate of the grid structure after one step of the advection algorithm, e.g. by using a forward-in-time integration of grid points. Such an estimate could also reduce the number of semi-Lagrangian iterations. These ideas are postponed for further investigations. We have also presented a simple parallelization technique for the reinitialization algorithm based on the pseudo-time transient formulation. Both the semi-Lagrangian and the reinitialization algorithms show good scalability up to 4096 processors, the current limit of our account.

Finally, an application of these algorithms is presented in modeling the solidification process by solving a Stefan problem. This application clearly illustrates the applicability of our algorithms to complex multi-scale problems that cannot be treated practically using the domain decomposition techniques on uniform grids. We believe that our findings can

serve as a basis to simulating a wide range of multi-scale and free boundary problems.

Chapter 3

Solving the incompressible Navier-Stokes equations on Quad/Oc-tree grids

3.1 Introduction

The prediction of fluid motion around structures is crucial in many important applications in science and engineering. Examples include the classical study of the aerodynamics of aircrafts or the hydrodynamics of ships, but also more modern applications such as the fluid dynamics occurring during materials processing such as the solidification of liquid metal alloys, the study of artificial swimmers or biological flows. One of the difficulties of solving the equations of fluid dynamics is in dealing with non-trivial geometries, which can be explicitly described (body-fitted approaches) or implicitly captured. We focus here on strategies on *Cartesian* grids, where the geometry is implicitly captured and refer the interested reader to the book by Peric and Ferziger [76] for discussions of body-fitted approaches.

Much of the early work was concerned with compressible flows and strategies based on Cartesian grids were first introduced on uniform grids by Purvis and Burkhalter [77], who used a finite volume approach to solve the two-dimensional potential equation. Later,

researchers proposed solvers for the Euler equations in two [78, 79] and three [78] spatial dimensions. One of the difficulties inherent to fluid flows is their spatial multiscale nature. From boundary layers close to solid boundaries to the generation of vorticity and turbulence, specific regions require a very fine level of detail while other areas can be treated adequately with a coarser mesh. Researchers have proposed strategies to alleviate this problem by designing numerical methods on spatially adaptive grids, which include stretched grids (see e.g. [80, 81]), nested grids (see e.g. [82, 83, 84, 85]), or unstructured meshes (see e.g. [86, 87, 88, 89, 90]). Large parallel codes have also been written and used in commercial applications, e.g. TRANAIR, which is an adaptive Cartesian full potential solver coupled with a viscous boundary layer model [91, 92] or NASA Cart3D [93], which is a solver for the compressible Euler equations, with application to high-speed flows.

We are focusing in this paper on the incompressible Navier-Stokes equations on Octree data structures, which provide the ability to refine/coarsen continuously in space[1]. These approaches follow the general framework of the projection method of Chorin [94], which leverages the Hodge decomposition of vector fields: first an intermediate velocity field is computed, before applying a projection onto the divergence-free subspace (the interested reader is referred to the excellent paper by Brown, Cortez and Minion [95] for a review of different projection methods). In that vein, Popinet [12] introduced a solver on Octrees using finite volume discretizations on the Marker And Cell (MAC) configuration [96]. In this work, the size between adjacent cells is constrained by a 2:1 ratio, which reduces the number of local grid configurations. In turn, this can be exploited to construct second-order approximations of the projection step, albeit leading to a non-symmetric linear system. Later, Losasso *et al.* introduced a solver for the incompressible Navier-Stokes equations and for free surface flows [13]. This approach considers octrees for which the ratio between adjacent cells is not constrained, allowing for increased adaptivity of the

[1]Note, however, the inherent memory and CPU overhead from encoding the tree structure.

computational mesh. The projection step uses a finite volume approach and leads to a symmetric linear system. However, the Hodge variable is only first-order accurate, which impacts on the accuracy of the velocity field. Losasso *et al.* [49] also introduced a second-order solver for the Poisson equation, based on the work of Lipnikov [97], but did not used that solver for fluid simulations. Later, Min and Gibou introduced a solver that uses an approach based on finite differences instead of finite volumes [98]. The linear system for the Hodge variable is non-symmetric but the solution is second-order accurate with second-order accurate gradients, which in turn produces also second-order accurate velocity fields.

However, one of the main challenges when considering a finite difference approach on adaptive meshes is the potential loss of numerical stability. In [98], Min and Gibou showed that the standard projection method cannot be guaranteed to be stable in that framework; it was confirmed numerically and shown to be exacerbated by high size ratios between adjacent cells and high Reynolds numbers. They also introduced the so-called "orthogonal projection" method that guarantees numerical stability (even though their method is not conservative) in the case where Dirichlet boundary conditions for the velocity field are imposed. However, the numerical stability is not guaranteed for inflow/outflow boundary conditions, which limits the range of applications of that approach. An approach based on the MAC grid configuration is more amenable to designing stable projection solvers: following the standard proof of L^2-stability, one can show that a minus transpose relationship of the discrete gradient and divergence operators is enough to guarantee numerical stability in a weighted L^2-norm. Such a constraint can be enforced in a MAC grid sampling of the data, even if no constraint is imposed on the grid; a difference from a node-based approach. In this paper, we present a projection method that is stable, using the Poisson solver introduced in [49] and deriving the numerical approximations of gradient and divergence operators to ensure numerical stability. A

MAC grid discretization also has the desirable property of being conservative.

Another challenge is the representation of the interface between the fluid and the irregular boundaries, and more specifically how to impose the boundary conditions in an implicit framework. A common approach is to use Peskin's immersed boundary method [99, 100]. However, we seek to avoid the smoothing of the solution induced by a delta formulation and the subsequent decrease in accuracy in the L^∞-norm near the boundaries; thus we represent the location of the interface implicitly with a level-set function [4] and impose the boundary conditions sharply on the interface. Several strategies have been introduced, e.g. the rasterization approach used in Losasso *et al.* [13] or the Heaviside formulation of Batty *et al.* [101]. However, Ng *et al.* showed that treatments such as these lead to a method that does not converge in the L^∞-norm [102] while a finite-volume/cut-cell approach in a level-set framework produces accurate results. We use that approach.

The last challenge in solving the incompressible Navier-Stokes equations is the time step restriction usually resulting from the discretizations of the advection (CFL condition) and the viscous ($\Delta t = O(\Delta x^2)$) terms. We circumvent those issues by employing a BDF semi-Lagrangian scheme [23] for the advection term and by treating the viscous term implicitly. Since we opt for a MAC layout, producing a compact accurate implicit solver for the viscous term is not straightforward, but it can be done with a finite volume approach where the control volumes are the elements of a Voronoi partition, as discussed in [103].

3.2 The numerical method

3.2.1 The projection method

We consider a computational domain $\Omega = \Omega^- \cup \Omega^+$, where the solution to the incompressible Navier-Stokes equations is computed in Ω^-. The boundary of Ω^- is denoted by Γ and that of Ω is denoted by $\partial\Omega$. The incompressible Navier-Stokes equations, in the case of a fluid with uniform viscosity μ and uniform density ρ, are written as

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\Delta\mathbf{u} + \mathbf{f} \quad \text{in } \Omega^-, \tag{3.1}$$

$$\nabla\cdot\mathbf{u} = 0 \quad \text{in } \Omega^-, \tag{3.2}$$

where t is the time, $\mathbf{u} = (u, v, w)$ is the velocity field, p is the pressure and $\mathbf{f}$ includes the external forces such as gravity. The classical approach to solve these equations was introduced by Chorin in 1967 [94] and is commonly referred to as the projection method. The first step of the method is to compute an intermediate velocity field $\mathbf{u}^*$ using the momentum equation (3.1). We choose to ignore the pressure gradient term in that step and we refer the reader to [95] for different variants of the projection method. If the temporal derivative was discretized with a forward Euler step, the first step would be solving for $\mathbf{u}^*$ in:

$$\rho\left(\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} + \mathbf{u}^n\cdot\nabla\mathbf{u}^n\right) = \mu\Delta\mathbf{u}^* + \mathbf{f}. \tag{3.3}$$

The second step is based on the Helmoltz-Hodge decomposition, which states that a twice continuously differentiable bounded vector field can be decomposed into a curl-free component and another divergence-free component. This means that we can decompose $\mathbf{u}^*$ as

$$\mathbf{u}^* = \mathbf{u}^{n+1} + \nabla\Phi, \tag{3.4}$$

where Φ will be refered to as the Hodge variable. Applying the divergence operator to this equation yields:

$$\nabla \cdot \nabla \Phi = \nabla \cdot \mathbf{u}^*. \tag{3.5}$$

Therefore, in order to compute the Hodge variable one can solve a simple Poisson equation. Once Φ has been computed, the intermediate velocity field $\mathbf{u}^*$ is projected onto the divergence-free subspace to obtain $\mathbf{u}^{n+1}$:

$$\mathbf{u}^{n+1} = \mathbf{u}^* - \nabla \Phi. \tag{3.6}$$

Since we are solving for the intermediate velocity field implicitly, Chorin's projection method requires the inversion of two linear systems, resulting from the discretization of equations (3.3) and (3.5). We now present the boundary conditions associated with those two steps before describing the discretization of the inertial and temporal terms.

3.2.2 Enforcing the boundary conditions

We focus on single phase flows around rigid objects, thus the possible boundary conditions are the no-slip condition at the interface between the fluid and objects and the influx/outflux conditions at the boundaries of the computational domain. The no-slip condition translates into a Dirichlet boundary condition on the velocity field

$$\mathbf{u}_{|\Gamma} = \mathbf{u}_{|b.c.}, \tag{3.7}$$

where $\mathbf{u}_{|b.c.}$ is given and is zero in the case of a static object and equal to the velocity of the object if the object is in motion. From equation (3.4) we deduce the boundary

condition to enforce on $\mathbf{u}^*$ when solving for the intermediate velocity field:

$$\mathbf{u}^*_{|\Gamma} = \mathbf{u}_{|b.c.} + \nabla\Phi^{n+1}_{|\Gamma}. \tag{3.8}$$

When solving equation (3.5) for the Hodge variable, we enforce a no-flux condition on the boundaries where $\mathbf{u}_{|b.c.}$ is prescribed, i.e.,

$$(\nabla\Phi \cdot \mathbf{n})_{|\Gamma} = 0, \tag{3.9}$$

where $\mathbf{n}$ is the outward normal vector to the interface between the fluid and the solid. Note that this makes sure that the boundary condition on the normal component of $\mathbf{u}$ is respected, however spurious slip can be introduced. Note that Φ^{n+1} is not known when we solve for the intermediate velocity field $\mathbf{u}^*$. We use the field computed at the previous time step Φ^n as an approximation and solve for $\mathbf{u}^*$ and Φ iteratively until convergence of both fields before moving on to the next time step. In practice we observed that Φ^n is a good initial guess for Φ^{n+1} and very few to no iterations are required, as discussed in sections 3.4.1 and 3.4.1.

The influx and outflux boundary conditions on $\partial\Omega$ are implemented through Neumann boundary conditions. Similarly to the Dirichlet case, the boundary condition on the intermediate velocity field $\mathbf{u}^*$ should be

$$(\nabla\mathbf{u}^* \cdot \mathbf{n})_{|\partial\Omega} = (\nabla\mathbf{u} \cdot \mathbf{n})_{b.c.} + (\nabla\nabla\Phi^{n+1} \cdot \mathbf{n})_{|\partial\Omega}, \tag{3.10}$$

while the boundary condition enforced when solving for the Hodge variable becomes a Dirichlet boundary condition to satisfy the compatibility criteria for the boundary conditions, with the enforced value being linked to the pressure (see equation (3.14)). However, as we will explain later, the Hodge variable is known to second-order accuracy

only, making the approximation of the double gradient term inaccurate. In practice, this term is nothing but a correction term and approximating the Neumann boundary condition on the intermediate velocity field by $(\nabla \mathbf{u}^* \cdot \mathbf{n})_{|\partial\Omega} = (\nabla \mathbf{u} \cdot \mathbf{n})_{b.c.}$ is reasonable and leads to convergent results in the velocity field.

3.3 Discretization and stability on the quadtree data structure

3.3.1 The level-set method and the Marker And Cell method on quadtree

The level-set method

We use the level-set framework to represent the interface between the fluid and the solid objects, thus enabling a sharp discretization of the boundary conditions: the interface Γ is represented by the zero contour of the so-called level-set function, generally denoted by ϕ, $\Gamma = \{(x, y)|\phi(x, y) = 0\}$. Following the standard notations, we define $\Omega^- = \{(x, y)|\phi(x, y) < 0\}$ to be the fluid subdomain and $\Omega^+ = \{(x, y)|\phi(x, y) > 0\}$ to be the solid subdomain.

If infinitely many level-set functions have a zero-contour matching Γ, it is convenient to use a signed distance function to represent the interface. The level-set function is therefore reinitialized to a signed distance function at every time iteration with a fast marching algorithm. The fast marching algorithm is initialized by iterating a second-order Total Variation Diminishing Runge-Kutta scheme a few times on the reinitialization equation

$$\frac{\partial \phi}{\partial \tau} + \text{sign}(\phi)\,(|\nabla \phi| - 1) = 0,$$

where τ is a fictitious time. The fast marching method is described in detail in [22, 21] and the iterative scheme can be found in [14]. We chose this combination of methods to preserve the location of the interface accurately and propagate the information to the rest of the domain rapidly.

The Marker And Cell data layout

The classical approach for direct numerical simulation of the Navier-Stokes equations on uniform mesh is to use the Marker And Cell (MAC) [96] layout for the data. The Hodge variable is thus located at the center of the cells, the x-velocity at the center of the vertical faces, the y-velocity at the center of the horizontal faces and the level-set values on the vertices, as shown in figure 3.1. While the MAC layout has been widely applied to the incompressible Navier-Stokes equations and can easily be proven to lead to a stable discretization of the projection method on uniform meshes, its implementation on quadtree data structures is more challenging.

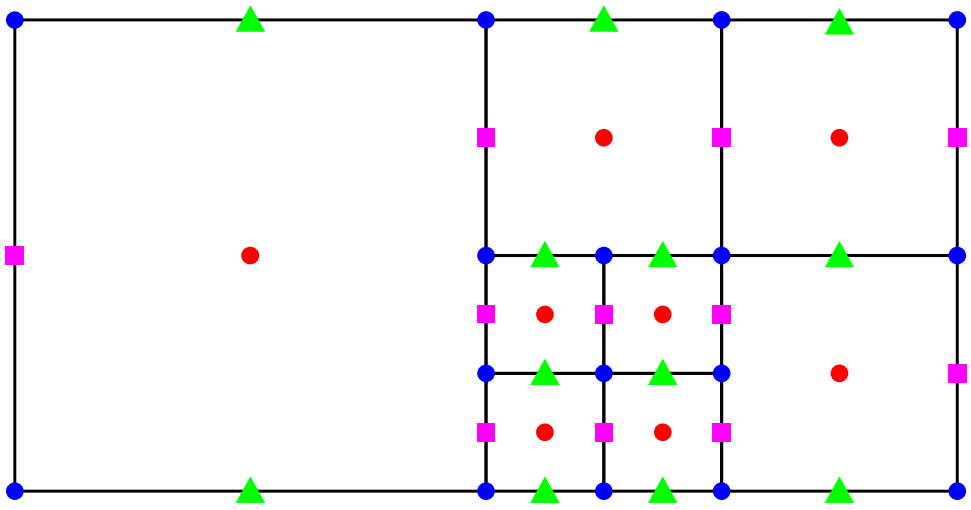


Figure 3.1: Layout of the data on the quadtree mesh, showing the pressure (●), the horizontal velocity (■), the vertical velocity (▲) and the level-set values (●).

The first difficulty encountered with this layout consists in interpolating the quantities

defined on the faces and at cell centers. The standard quadratic interpolations can be used to find a second-order accurate approximation of the vertex values at any point since a square can always be defined with vertices, but this is not true for the velocity and Hodge values on an adaptive MAC grid. To overcome this complication, we use weighted quadratic least squares interpolations. This means that every time an interpolation is performed, a neighborhood of points (either face centers or cell centers) must be gathered and a small 6 by 6 (10 by 10 in three dimensions) symmetric positive definite system must be inverted. We use a Cholesky decomposition for this purpose. We choose to gather all direct and second order neighbors to create the stencil to make sure that the system is overdetermined, where the second order neighbors are defined as the neighbors of the neighbors. The general procedure to gather the neighbors is presented in algorithm 7. We choose the weights to be $w_i = \frac{1}{\sqrt{(x-x_i)^2+(y-y_i)^2}}$, where (x, y) are the coordinates of the point where the value is interpolated at and (x_i, y_i) are the coordinates of the neighbor point number n_i involved in the construction of least squares linear system.

The result is a very costly interpolation procedure that must be used sparingly (albeit one that is intrinsically embarrassingly parallelizable). Taking this into account, we interpolate the velocities from the faces to the nodes after each iteration to minimize the cost of the interpolations in the Semi-Lagrangian scheme, which we will present in the next section. The interpolations on node-based values are computationally inexpensive and straightforward, as explained in [14].

Extrapolating face and cell centered quantities

A second obstacle resides in the extension of quantities across the fluid-solid interface. A second-order iterative scheme like that applied to the reinitialization equation can be used for values defined on the vertices, however the layout of faces and cell centers values is too cumbersome for a simple implementation. We designed a geometric extrapolation

let (x,y) be the coordinates where the value is to be interpolated
let *ngbd* be the list of neighboring cells, initially empty
find the cell C_{xy} containing (x,y) and add it to *ngbd*
for $dir \in \{-1, 1\}$ **do**
    find the neighbor cells of C_{xy} in the direction $(dir, 0)$ and add them to *ngbd*
    find the neighbor cells of C_{xy} in the direction $(0, dir)$ and add them to *ngbd*
end for
let N be the length of *ngbd*
for $n < N$ **do**
    find the neighbors of $ngbd(n)$ in all four directions
    add them to *ngbd* if they are not already in *ngbd*
end for

Algorithm 7: Algorithm to gather the neighborhood of points required by the least squares interpolation procedures.

procedure to circumvent this challenge. For each point located in the solid subdomain, we compute the projection of the point on the interface and follow the normal to the interface in the negative subdomain to gather well-defined values. We then compute the extrapolated value by evaluating at the extrapolation point a second degree Newton polynomial, built from the values gathered and the boundary condition.

Discretizations, boundary conditions and constraint on the grid

The finite difference discretizations on the quadtree data structure present two challenges. The first challenge comes from the grid structure and the existence of T-junction nodes, i.e. nodes for which there is a missing neighbor node in one of the Cartesian directions. The second challenge is the presence of an irregular interface. Both examples are depicted on figure 3.2.

The T-junctions problem is solved by computing a ghost neighbor v_g in the Cartesian direction where a node is missing, as explained in [14]. Given a node-sampled function $\phi : \{v_i\} \to \mathbb{R}$, we define $\phi_g = \phi(v_g)$ as

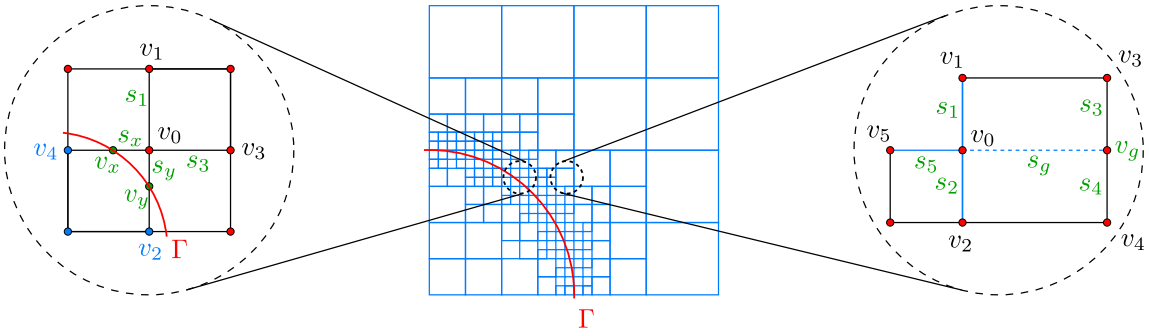


Figure 3.2: Mesh configurations requiring a special treatment for the finite difference discretizations.

$$\phi_g = \frac{s_3\phi_4 + s_4\phi_3}{s_3 + s_4} - \frac{s_3 s_4}{s_1 + s_2}\left(\frac{\phi_1 - \phi_0}{s_1} + \frac{\phi_2 - \phi_0}{s_2}\right).$$

This compact discretization provides a third order accuracte interpolation of ϕ_g. The first and second order derivatives of ϕ in the x-direction are then obtained as

$$D_x\phi_0 = \frac{\phi_g - \phi_0}{s_g} \cdot \frac{s_5}{s_5 + s_g} + \frac{\phi_0 - \phi_5}{s_5} \cdot \frac{s_g}{s_5 + s_g},$$
$$D_x\phi_0 = \frac{\phi_g - \phi_0}{s_g} \cdot \frac{2}{s_5 + s_g} - \frac{\phi_0 - \phi_5}{s_5} \cdot \frac{2}{s_5 + s_g}.$$

The treatment of Dirichlet boundary conditions on the irregular interface is explained in [104] and necessitates to locate the interface between two neighbor nodes (using linear interpolations). Consequently we impose uniformity of the quadtree close to the interface. Following the notations introduced in figure 3.2, we compute the distances s_x and s_y by constructing a quadratic interpolant of the level-set function respectively in the x and y directions and computing the roots of the interpolant. The values ϕ_x and ϕ_y on the interface being prescribed, we can then define

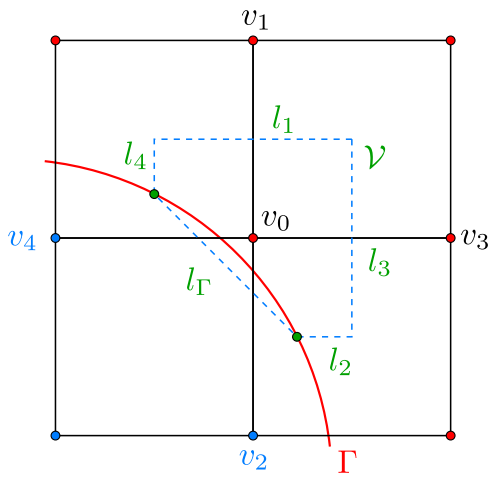


Figure 3.3: Geometrical configuration for the finite volume discretization close to an irregular interface with Neumann boundary conditions.

$$D_{xx}\phi_0 = \frac{2}{s_x + s_3} \cdot \left(\frac{\phi_3 - \phi_0}{s_3} - \frac{\phi_0 - \phi_x}{s_x} \right),$$
$$D_{yy}\phi_0 = \frac{2}{s_y + s_1} \cdot \left(\frac{\phi_1 - \phi_0}{s_1} - \frac{\phi_0 - \phi_y}{s_y} \right).$$

Neumann boundary conditions on the irregular interface are handled with a finite volume approach. Provided that the mesh is uniform close to the interface, the control volume $\mathcal{V}$ around a node is a square. The situation is represented in figure 3.3 together with the notations we use. We can then discretize a poisson equation with the Neumann boundary condition $(\nabla\phi \cdot \mathbf{n})_{|\Gamma}$ as

$$-\Delta\phi = f \quad \Rightarrow \quad \sum_i l_i \frac{\phi_0 - \phi_i}{\delta} = \mathcal{A}f + l_\Gamma (\nabla\phi \cdot \mathbf{n})_{|\Gamma},$$

where δ is the size of the smallest cells and $\mathcal{A}$ is the area enclosed by the control volume $\mathcal{V}$.

Note that since the mesh is constrained to be uniform close to the irregular interface,

the geometric configuration is the same whether we are considering node, face or cell centered values. Therefore, the treatment of the boundary conditions we just described aslo holds for the discretization of the momentum equation and of the projection step.

3.3.2 Discretization of the momentum equation

Discretization of the advection term

We discretize the left-hand side of equation (3.3) for the intermediate velocity $\mathbf{u}^*$ using a semi-Lagrangian approach and a second-order Backward Difference Formula scheme, as described in [98, 105], with an adaptive time step. The general term corresponding to the advection of a field $\mathbf{u}$ by a velocity field $\mathbf{v}$,

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{u},$$

can be discretized at time t_{n+1} by interpolating $\mathbf{u}$ along the characteristic curve of the equation through $\mathbf{u}^{n+1}$. The characteristic curve through $\mathbf{u}^{n+1}$ is followed backward in time to find the departure points $\mathbf{x}_d^n$ and $\mathbf{x}_d^{n-1}$ as

$$\hat{\mathbf{x}} = \mathbf{x}^{n+1} - \frac{\Delta t_n}{2} \cdot \mathbf{v}^n(\mathbf{x}^{n+1}),$$
$$\mathbf{x}_d^n = \mathbf{x}^{n+1} - \Delta t_n \cdot \mathbf{v}^{n+\frac{1}{2}}(\hat{\mathbf{x}})$$

and

$$\bar{\mathbf{x}} = \mathbf{x}^{n+1} - \Delta t_n \cdot \mathbf{v}^n(\mathbf{x}^{n+1}),$$
$$\mathbf{x}_d^{n-1} = \mathbf{x}^{n+1} - (\Delta t_n + \Delta t_{n-1}) \cdot \mathbf{v}^n(\bar{\mathbf{x}})$$

where Δt_{n-1} and Δt_n are the adaptive time steps respectively from t_{n-1} to t_n and from t_n to t_{n+1}. The intermediate velocity $\mathbf{v}^{n+\frac{1}{2}}(\hat{\mathbf{x}})$ is interpolated from the velocity fields at time t_{n-1} and t_n as

$$\mathbf{v}^{n+\frac{1}{2}}(\hat{\mathbf{x}}) = \frac{2\Delta t_{n-1} + \Delta t_n}{2\Delta t_{n-1}} \mathbf{v}^n(\hat{\mathbf{x}}) - \frac{\Delta t_n}{2\Delta t_{n-1}} \mathbf{v}^{n-1}(\hat{\mathbf{x}}).$$

We then interpolate $\mathbf{u}$ at the departure points $\mathbf{x}_d^{n-1}$ and $\mathbf{x}_d^n$ from the values stored at the vertices with quadratic interpolation procedures[2] and we denote them $\mathbf{u}_d^{n-1}$ and $\mathbf{u}_d^n$. The advection equation formulated along a characteristic curve becomes

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{u} = \frac{d\mathbf{u}}{ds} \tag{3.11}$$

where the characteristic curve has been parametrized by $(\mathbf{x}(s), t(s))$. Solving for equation (3.11) to first order yields the familiar scheme

$$\frac{\partial \mathbf{u}^{n+1}}{\partial s} \approx \frac{\mathbf{u}^{n+1} - \mathbf{u}_d^n}{\Delta t_n}.$$

Note that in the context of the level-set function advection, the advection term is equal to zero and we have the familiar update $\mathbf{u}^{n+1} = \mathbf{u}_d^n$. We choose to discretize (3.11) with a second order backward difference scheme, and therefore the time derivative of $\mathbf{u}$

[2]Although one should replace quadratic interpolations by multilinear ones when a new maximum is introduced, we have only used quadratic interpolations in all our examples without encountering any instabilities.

along the characteristic is approximated by

$$\frac{\partial \mathbf{u}^{n+1}}{\partial s} \approx \alpha \frac{\mathbf{u}^{n+1} - \mathbf{u}_d^n}{\Delta t_n} + \beta \frac{\mathbf{u}_d^n - \mathbf{u}_d^{n-1}}{\Delta t_{n-1}}. \tag{3.12}$$

Expanding $\mathbf{u}_d^{n-1}$ and $\mathbf{u}_d^n$ into

$$\begin{aligned} \mathbf{u}_d^{n-1} &= \mathbf{u}^{n+1} - (\Delta t_n + \Delta t_{n-1})\partial_s \mathbf{u}^{n+1} + \frac{(\Delta t_n + \Delta t_{n-1})^2}{2}\partial_s^2 \mathbf{u}^{n+1}, \\ \mathbf{u}_d^n &= \mathbf{u}^{n+1} - \Delta t_n \partial_s \mathbf{u}^{n+1} + \frac{\Delta t_n^2}{2}\partial_s^2 \mathbf{u}^{n+1} \end{aligned}$$

and replacing them in equation (3.12) leads to

$$\begin{aligned} &\alpha\left(\partial_s \mathbf{u}^{n+1} - \frac{\Delta t_n}{2}\partial_s^2 \mathbf{u}^{n+1}\right) + \frac{\beta}{\Delta t_{n-1}}\left(\Delta t_{n-1}\partial_s \mathbf{u}^{n+1} - (\Delta t_{n-1}\Delta t_n + \frac{\Delta t_{n-1}^2}{2})\partial_s^2 \mathbf{u}^{n+1}\right) \\ &\qquad = \partial_s \mathbf{u}^{n+1} \\ \Rightarrow \quad &(\alpha+\beta)\partial_s \mathbf{u}^{n+1} - (\alpha\frac{\Delta t_n}{2} + \beta\Delta t_n + \beta\frac{\Delta t_{n-1}}{2})\partial_s^2 \mathbf{u}^{n+1} = \partial_s \mathbf{u}^{n+1} \\ \Rightarrow \quad &\begin{cases} \alpha + \beta = 1 \\ \alpha\frac{\Delta t_n}{2} + \beta\Delta t_n + \beta\frac{\Delta t_{n-1}}{2} = 0 \end{cases} \\ \Rightarrow \quad &\begin{cases} \alpha = \frac{2\Delta t_n + \Delta t_{n-1}}{\Delta t_n + \Delta t_{n-1}} \\ \beta = -\frac{\Delta t_n}{\Delta t_n + \Delta t_{n-1}} \end{cases} \end{aligned}$$

The discretization of the left-hand side of equation (3.3) for the intermediate velocity $\mathbf{u}^*$ using a semi-Lagrangian approach and a second-order Backward Difference Formula

scheme with an adaptive time step is therefore

$$\rho\left(\alpha\frac{\mathbf{u}^* - \mathbf{u}_d^n}{\Delta t_n} + \beta\frac{\mathbf{u}_d^n - \mathbf{u}_d^{n-1}}{\Delta t_{n-1}}\right) = \mu\Delta\mathbf{u}^* + \mathbf{f}. \tag{3.13}$$

This temporal discretization is unconditionally stable, lifting any restriction on the time step. From the above discretization (3.13) and the Hodge decomposition (3.4), the pressure can then be recovered as

$$p = \alpha\frac{\rho}{\Delta t_n}\Phi - \mu\Delta\Phi. \tag{3.14}$$

Implicit discretization of the viscous term

The first step of the projection method consists in solving the momentum equation without the pressure gradient term (3.3). Note that for a fluid with uniform viscosity the two (respectively three in three dimensions) components of the velocity field are decoupled, therefore we can solve for the horizontal and the vertical components separately. A finite volume approach based on Voronoi control volumes, as introduced in [103] and applied in [106] in the context of Chimera grids, provides a compact implicit second-order solver for data localized either at the center of the vertical or horizontal edges of a non-graded quadtree.

We propose a finite volume solver where the control volumes are the cells of the Voronoi diagram built with the data points. The Voronoi diagram of a set of data points, or seeds, is the set of Voronoi cells such that the voronoi cell of a seed consists of all the points closer to that seed than to any other. See figure 3.4 for an example of a Voronoi cell. We construct the Voronoi cells in parallel using a simple geometric algorithm we developed in two dimensions and exploiting the software `Voro++` [107] in three dimensions. In the latter, for each point we start by gathering the potential voronoi neighbors before

calling `Voro++` on this reduced set. This drastically improves the computation time and enables simple and very efficient parallelization of the procedure, since it is intrinsically embarassingly parallel.

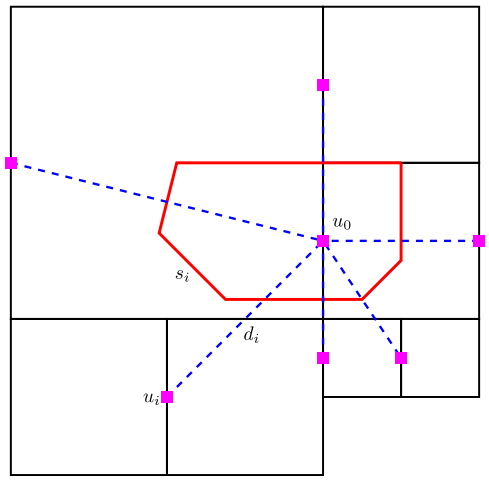


Figure 3.4: Example of a Voronoi cell for a horizontal velocity data point u_0. We call d_i the distance between the data point u_0 and a neighbor data point u_i which is involved in the construction of the Voronoi cell, and s_i the length of the edge (or area of the surface in three dimensions) connecting u_0 and u_i.

The Voronoi diagram has desirable properties for a second-order discretization. It connects data points though an edge (or face in three dimensions) that is the bissector line (or bisector plane) of the segment joining the two data points. While this corresponds to a second-order accurate discretization of the flux at the edge on uniform grids, it is not necessarily the case on quad/oc-trees. However, the flux is orthogonal to the edge (or face) connecting two data points and numerical examples on highly arbitrary grids show second order accuracy. Furthermore, the Voronoi diagram is a tessellation of the plane. Finally, we build the Voronoi diagram inside the fluid only so that the edges (or faces) of the Voronoi cells next to the interface lie on the interface, facilitating the discretization of Neumann boundary conditions. In two dimensions, we enforce this directly on the Voronoi representation by cutting the Voronoi cells close to the interface by a plane (see

configuration in figure 3.3). The location of the plane is obtained by finding the points on the edges of the cell where the level-set function is zero. In three dimensions, we use the `Voro++` software far from the interface and construct the Voronoi cells by hand close to the interface with the direct extension of the two dimensional algorithm, knowing that the mesh is uniform close to the interface.

The momentum step (3.3) of the projection method with the left-hand side discretized can be restated in a finite volume formulation as

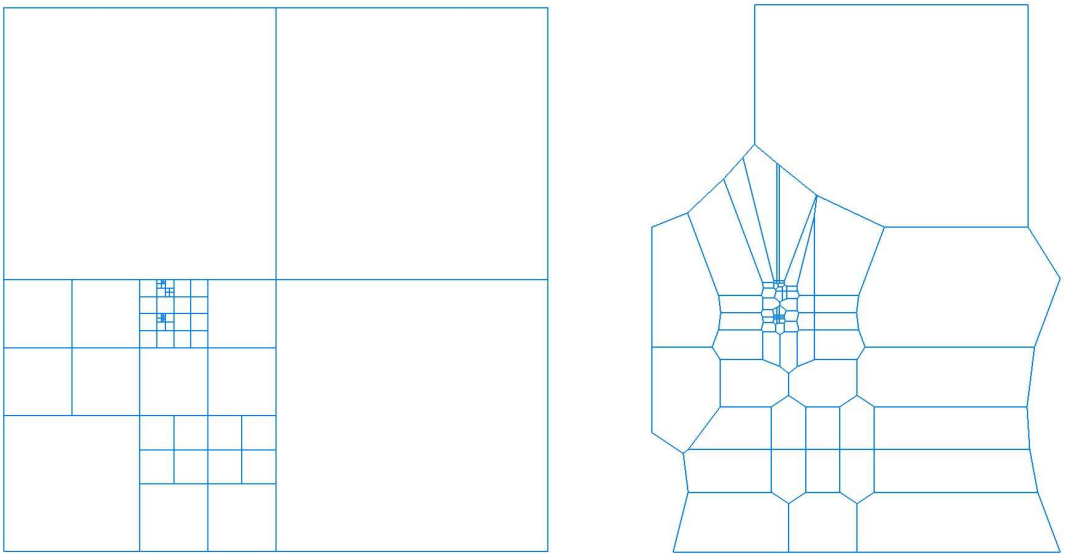
$$\begin{aligned} \int_{\mathcal{C}} \rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) &= \int_{\mathcal{C}} \mu \Delta \mathbf{u} \\ \implies \rho \mathrm{Vol}(\mathcal{C}) \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) &= \mu \int_{\partial \mathcal{C}} \nabla \mathbf{u} \cdot \mathbf{n}, \end{aligned} \tag{3.15}$$

where $\mathcal{C}$ is the Voronoi cell, $\mathrm{Vol}(\mathcal{C})$ its volume, $\partial\mathcal{C}$ its contour and $\mathbf{n}$ its outer normal. We can then discretize the right-hand-side of (3.15) for data point u_0 as

$$\mu \int_{\partial \mathcal{C}} \nabla \mathbf{u} \cdot \mathbf{n} = \mu \sum_{i \in \mathrm{Voro}(u_0)} s_i \frac{u_i - u_0}{d_i}, \tag{3.16}$$

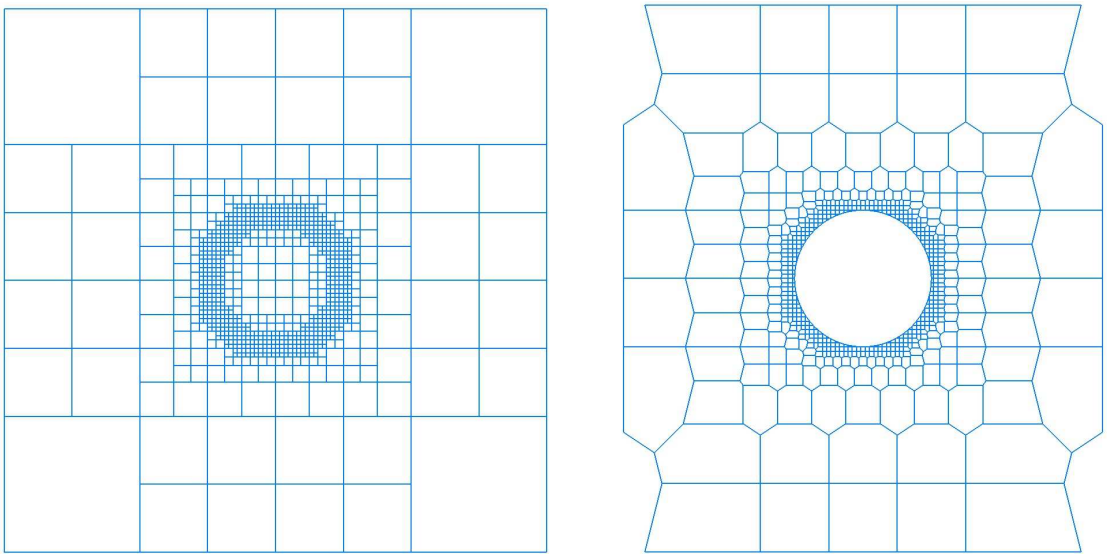
where $\mathrm{Voro}(u_0)$ is the set of neighbors of u_0 involved in its Voronoi cell, d_i is the distance between the two data points u_0 and u_i, and s_i is the length of the edge (or the area of the face in three dimensions) connecting the two data points (see figure 3.4). The system resulting from this Voronoi diagram based solver is obviously symmetric positive definite and numerical experiments indicate second-order accuracy on highly non-graded meshes (figure 3.5) and with both Dirichlet and Neumann boundary conditions on an irregular interface (figure 3.6).

Neumann boundary conditions are implemented naturally by integrating $(\nabla \mathbf{u} \cdot \mathbf{n})_{|b.c}$ on the edges (or faces) of the Voronoi cells in contact with the boundary. We enforce



number of recursive splits	L^∞ error	order
0	$3.21 \cdot 10^{-2}$	-
1	$1.76 \cdot 10^{-2}$	0.87
2	$5.60 \cdot 10^{-3}$	1.65
3	$1.53 \cdot 10^{-3}$	1.87
4	$3.97 \cdot 10^{-4}$	1.95

Figure 3.5: Convergence of the Voronoi diagram based Poisson solver on an arbitrary grid. The top left figure shows the quadtree mesh and the top right figure shows the corresponding Voronoi diagram for the x-velocity. The initial tree is level 1/8, with a maximum difference of 7 levels between two adjacent cells. The test function is $u(x, y) = \cos(x)\sin(y)$. The successive trees are obtained by recursively splitting all the cells of the initial tree and the convergence of the solver is presented. The Voronoi cells on the edge of the domain are missing on the top right figure because Dirichlet boundary conditions are enforced, so they do not need to be computed. Even on this highly non-graded mesh, the solver is second-order accurate.



quadtree level	Neumann		Dirichlet	
(min / max)	L^∞ error	order	L^∞ error	order
4/7	$1.45 \cdot 10^{-3}$	-	$1.48 \cdot 10^{-3}$	-
5/8	$3.05 \cdot 10^{-4}$	2.25	$3.92 \cdot 10^{-4}$	1.91
6/9	$7.48 \cdot 10^{-5}$	2.03	$1.01 \cdot 10^{-4}$	1.96
7/10	$1.88 \cdot 10^{-5}$	1.99	$2.54 \cdot 10^{-5}$	1.98
8/11	$4.81 \cdot 10^{-6}$	1.97	$6.39 \cdot 10^{-6}$	1.99

Figure 3.6: Convergence of the Voronoi diagram based Poisson solver. The top left figure shows the quadtree mesh and the top right figure shows the corresponding Voronoi diagram for the x-velocity. The domain is $\Omega = [-1, 1]^2 \backslash C\{(0, 0), 0.25\}$, and the test function is $u(x, y) = \cos(x)\sin(y)$. We successively enforce Dirichlet and Neumann boundary conditions on the edge of the computational domain and on the interface and report the respective results in the table. A level n quadtree means it has undergone n recursive splits, so that the equivalent uniform grid would have a resolution of $2^n \times 2^n$. The Voronoi cells on the edge of the domain are missing on the top right figure because Dirichlet boundary conditions are enforced, so they do not need to be computed. The solver is second-order accurate.

Dirichlet boundary conditions through a finite difference approach as explained in [104]. Consequently, we impose uniformity of the quadtree grid close to the interface. The linear system resulting from this discretization is solved with the numerical solvers provided by the Petsc libraries [75, 108, 109]. We choose to use the Bi-Conjugate Gradient Stabilized iterative solver in combination with the Hypre multigrid preconditioner. For the case of Neumann boundary conditions on both the interface and the edge of the computational domain, the linear system is ill-posed and we remove the nullspace from the matrix using the Petsc procedures [108].

3.3.3 Discretization of the projection step

The projection step is resolved by solving the poisson equation (3.5) to compute the value of the Hodge variable Φ at the center of the cells. Such a discretization was introduced in [13], which produced first-order accurate solutions for the Hodge variable. Later [49] presented a second-order accurate Poisson solver for the Hodge variable, but did not apply it to the projection step. Here, we use the Poisson solver of [49] and show how to define a discrete divergence operator to ensure numerical stability.

Second order discretization of the flux of the Hodge variable on the faces

We call c_i the quadtree cell containing Φ_{c_i}, and we will use the notations presented on figure 3.7 where Φ_{c_0} is at the center of the large cell c_0, $\text{Ngb}_L(c_0)$ are all the indices of the small cells connected with the same large cell in the left direction (e.g. in the case presented, $\text{Ngb}_R(c_2) = \text{Ngb}_L(c_0) = \{1, 2, 3\}$), S_{c_i} is the length of the edge connecting c_0 and c_i, and δ_{fc_i} is the signed distance between the center of c_i and the face f where $\nabla\Phi|_f \cdot \mathbf{n}_f$ is being discretized, $\mathbf{n}_f$ being the outer normal to the face f. Hence, if the face f is to the left of c_i then $\delta_{fc_i} > 0$, and $\delta_{fc_i} < 0$ if f is to the right of c_i. Note that we are

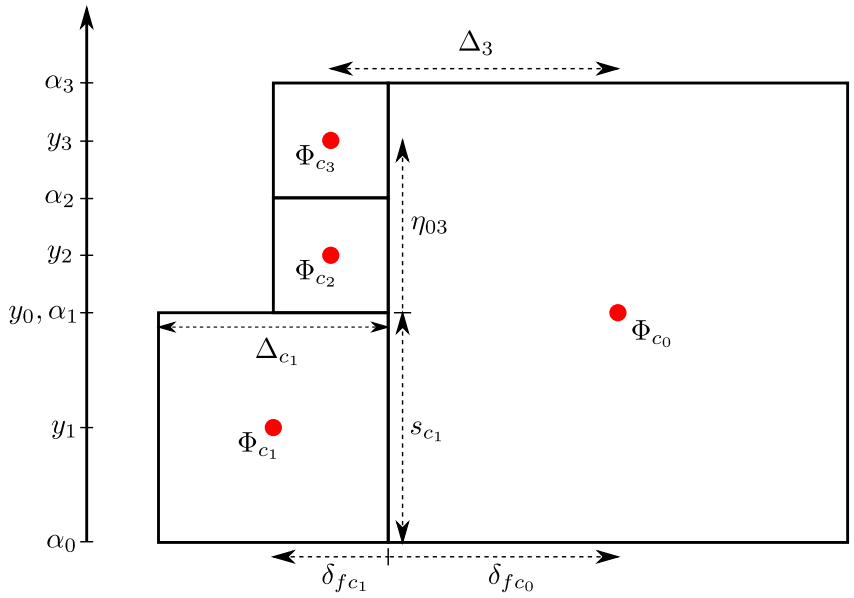


Figure 3.7: Nomenclature used in the discretization of the projection step (section 3.3.3).

using a finite volume approach on cartesian grid, so for a vertical face only $\partial_x \Phi$ needs to be discretized. First, we define the average distance:

$$\Delta_f = \sum_{i \in \mathrm{Ngb}_L(c_0)} \frac{s_{c_i}}{s_{c_0}} (\delta_{fc_0} - \delta_{fc_i}) = \sum_{i \in \mathrm{Ngb}_L(c_0)} \frac{s_{c_i}}{s_{c_0}} \Delta_i.$$

We then define the flux of Φ on the left face f of c_0 as:

$$\nabla \Phi|_f \cdot \mathbf{n}_f = (\partial_x \Phi)_{c_0,L} = \sum_{i \in \mathrm{Ngb}_L(c_0)} \frac{s_{c_i}}{s_{c_0}} \left(\frac{\Phi_{c_0} - \Phi_{c_i}}{\Delta_f} \right).$$

From our notations comes $(\partial_x \Phi)_{c_i | i \in \mathrm{Ngb}_L(c_0), R} = -(\partial_x \Phi)_{c_0,L}$, i.e. the discretization is symmetric and the flux is conserved (note that the normals are pointing in opposite directions, cancelling the minus sign when computing the flux). Intuitively, including the neighbors in the discretization of the gradient allows for compensating the orthogonal component. We now show that this discretization is second-order accurate at the center

of the face f of cell c_k. We call η_{ik} the signed distance between the center of c_i and the center of c_k. Then,

$$\begin{aligned}
(\partial_x \Phi)_{c_k,R} &= \frac{1}{s_{c_0}\Delta_f} \sum_{i \in \mathrm{Ngb}_R(c_k)} s_{c_i}(\Phi_{c_0} - \Phi_{c_i}) \\
&= \frac{1}{s_{c_0}\Delta_f} \sum_{i \in \mathrm{Ngb}_R(c_k)} s_{c_i}(\Phi|_f + \delta_{fc_0}\partial_x\Phi|_f + \eta_{k0}\partial_y\Phi|_f - \Phi|_f - \delta_{fc_i}\partial_x\Phi|_f \\
&\qquad\qquad - \eta_{ki}\partial_y\Phi|_f + o(\delta_{fc_i}, \eta_{ki})) \\
&= \frac{1}{s_{c_0}\Delta_f} \sum_{i \in \mathrm{Ngb}_R(c_k)} s_{c_i}\left(\Delta_i \partial_x\Phi|_f + \eta_{i0}\partial_y\Phi|_f + o(\delta_{fc_i}, \eta_{ki})\right) \\
&= \partial_x\Phi|_f + \frac{\partial_y\Phi|_f}{s_{c_0}\Delta_f} \sum_{i \in \mathrm{Ngb}_R(c_k)} s_{c_i}\eta_{i0} + o(\delta_{fc_i}, \eta_{ki}),
\end{aligned}$$

and writing y_i the ordinate of the center of cell c_i, and α_i the ordinate of the center of the edges of the cells,

$$\begin{aligned}
\sum_{i \in \mathrm{Ngb}_R(c_k)} s_{c_i}\eta_{i0} &= \sum_{i \in \mathrm{Ngb}_R(c_k)} s_{c_i}(y_0 - y_i) \\
&= s_{c_0}y_0 - \sum_{i=0}^{n} \frac{1}{2}(\alpha_{i+1} - \alpha_i)(\alpha_{i+1} + \alpha_i) \\
&= s_{c_0}y_0 - \frac{1}{2}\sum_{i=0}^{n}(\alpha_{i+1}^2 - \alpha_i^2) \\
&= s_{c_0}y_0 - \frac{1}{2}(\alpha_n^2 - \alpha_0^2) \\
&= s_{c_0}y_0 - \frac{1}{2}(\alpha_n - \alpha_0)(\alpha_n + \alpha_0) \\
&= s_{c_0}y_0 - s_{c_0}y_0 \\
&= 0.
\end{aligned}$$

This proves that the proposed discretization is second-order at the center of the cell faces, and therefore the Poisson solver for the cell-centered data is second-order accurate.

Neumann boundary conditions are enforced naturally on the irregular interface with a finite-volume discretization while Dirichlet boundary conditions are applied through a finite difference approach, making sure that the mesh is uniform close to the interface [110, 111, 104]. Again, the resulting linear system is solved using the Petsc libraries with a Bi-Conjugate Gradient Stabilized iterative solver combined with the Hypre multigrid preconditioner. The cost of assembling the linear system, which scales linearly with the number of cells, is negligible compared to that of computing the solution.

Discretization of the divergence operator and stability of the projection step

We modify our notation slightly (see figure 3.7 to clearly distinguish between the set of all faces Ω_F and the set of all cells Ω_C) to express the gradient and the divergence as linear operators. We now call $\mathrm{N_f}(c)$ the set of the faces in contact with cell c and $\mathrm{N_c}(f)$ the set of the cells in contact with the largest direct cell neighbor of face f. Furthermore, we define

$$A_f = \frac{1}{2} \sum_{c_i \in \mathrm{N_c}(f)} s_{c_i},$$

which is equivalent to the s_{c_0} from the previous section, but with these new notations we abolish the distinction between the largest cell and the smaller ones. We can then rewrite Δ_f and the gradient operator for any face $f \in \Omega_F$ as:

$$\begin{aligned} \Delta_f &= \frac{1}{A_f} \sum_{c_i \in \mathrm{N_c}(f)} s_{c_i} |\delta_{fc_i}| \\ \nabla\Phi|_f &= \frac{1}{A_f} \sum_{c_i \in \mathrm{N_c}(f)} \frac{s_{c_i}\Phi_{c_i}}{\Delta_f} \frac{\delta_{fc_i}}{|\delta_{fc_i}|} \end{aligned} \tag{3.17}$$

Our goal is to design the discrete divergence operator D as the minus transpose of

the discrete gradient operator G for some given metrics L_F and L_C:

$$L_F G = -(L_C D)^T$$

This leads to the definition of the discrete divergence operator for any cell $c \in \Omega_C$

$$\nabla \cdot \mathbf{u}^*|_c = -\frac{1}{\Delta_c} \sum_{f_i \in \mathrm{N_f}(c)} \frac{s_{f_i} u_i}{A_{f_i}} \frac{\delta_{f_i c}}{|\delta_{f_i c}|}, \tag{3.18}$$

where Δ_c is the size of cell c. We now prove that the projection method is stable on non-graded adaptive cartesian mesh for this choice of operators. We first clarify the expressions for L_F and L_C, and we then make use of the adjoint property to demonstrate the stability of the method.

Equations (3.17) and (3.18) are the definitions for the discrete linear operators $G : \Omega_C \to \Omega_F$ and $D : \Omega_F \to \Omega_C$. The coefficients for those two operators are given by the following expressions:

$$\begin{aligned} G_{fc} &= \frac{1}{A_f} \frac{s_c}{\Delta_f} \frac{\delta_{fc}}{|\delta_{fc}|} \qquad \text{if } c \in \mathrm{N_c}(f),\ 0 \text{ otherwise,} \\ D_{cf} &= -\frac{1}{\Delta_c} \frac{s_f}{A_f} \frac{\delta_{fc}}{|\delta_{fc}|} \qquad \text{if } f \in \mathrm{N_f}(c),\ 0 \text{ otherwise.} \end{aligned}$$

We then define the diagonal operators

$$\begin{aligned} L_{S_C} : \Omega_C \to \Omega_C, \qquad L_{S_F} : \Omega_F \to \Omega_F, \\ L_{\Delta_C} : \Omega_C \to \Omega_C, \qquad L_{\Delta_F} : \Omega_F \to \Omega_F, \end{aligned}$$

with respective diagonal coefficients

$$(L_{S_C})_{cc} = \frac{1}{s_c}, \qquad (L_{S_F})_{ff} = \frac{1}{S_f},$$
$$(L_{\Delta_C})_{cc} = \Delta_c, \qquad (L_{\Delta_F})_{ff} = \Delta_f.$$

From these definitions, it follows that:

$$(L_{\Delta_F} G L_{S_C})_{fc} = \frac{1}{A_f}\frac{\delta_{fc}}{|\delta_{fc}|} = -(L_{\Delta_C} D L_{S_F})_{cf}$$

and therefore the operator $L_{\Delta_F} G L_{S_C}$ is the minus transpose of $L_{\Delta_C} D L_{S_F}$. Written in a more compact form,

$$L_F G = -(L_C D)^T,$$

where $L_F = L_{S_F}^{-1} L_{\Delta_F}$ and $L_C = L_{S_C}^{-1} L_{\Delta_C}$. From equation (3.6) we know that the discrete fields $\mathbf{u}^*$ and Φ satisfy

$$\|\mathbf{u}^{n+1}\|_{L_F} = \|\mathbf{u}^* - G\Phi\|_{L_F} = \|\mathbf{u}^*\|_{L_F} - 2 < \mathbf{u}^* | L_F | G\Phi > + \|G\Phi\|_{L_F}, \tag{3.19}$$

where $||\cdot||_{L_F}$ is the norm associated to the metric L_F and $<\cdot>$ is the standard scalar product. We can make use of the minus transpose property to rewrite the right-hand side as

$$< \mathbf{u}^* | L_F | G\Phi > = - < L_C D\mathbf{u}^* | \Phi >$$

From equation (3.5) we know that

$$DG\Phi = D\mathbf{u}^*$$

and therefore

$$< \mathbf{u}^* | L_F | G\Phi >= - < L_C D G\Phi | \Phi >$$

Using the minus transpose property again gives

$$< \mathbf{u}^* | L_F | G\Phi >= - < G\Phi | L_F | G\Phi >= \|G\Phi\|_{L_F}$$

We now plug the above expression back into equation (3.19) to find

$$\begin{aligned} \|\mathbf{u}^{n+1}\|_{L_F} = \|\mathbf{u}^* - G\Phi\|_{L_F} &= \|\mathbf{u}^*\|_{L_F} - 2\|G\Phi\|_{L_F} + \|G\Phi\|_{L_F} \\ &= \|\mathbf{u}^*\|_{L_F} - \|G\Phi\|_{L_F}, \end{aligned}$$

which implies that $||\mathbf{u}^{n+1}||_{L_F} \leq ||\mathbf{u}^*||_{L_F}$ and therefore the proposed projection is stable in the $\|\cdot\|_{L_F}$ norm. Since we are working in finite dimension, we can conclude that our method is stable.

Numerical validation of the projection step

In order to verify numerically the stability of the projection method presented in the previous section, we consider the velocity field $\mathbf{U}^* = (u^*, v^*)$ with

$$\begin{aligned} u^*(x,y) &= \sin(x)\cos(y) + x(\pi - x)y^2(\frac{y}{3} - \frac{\pi}{2}), \\ u^*(x,y) &= -\cos(x)\sin(y) + y(\pi - y)x^2(\frac{x}{3} - \frac{\pi}{2}), \end{aligned}$$

in the domain $\Omega = [0, \pi]^2$. This vector field can be decomposed into $\mathbf{U}^* = \mathbf{U} + \nabla\phi$ where $\mathbf{U}$ is the divergence-free field $(\sin(x)\cos(y), -\cos(x)\sin(y))$ and $\phi = (\frac{x^3}{3} - \frac{\pi x^2}{2})(\frac{y^3}{3} - \frac{\pi y^2}{2})$. We apply our projection method iteratively on the highly non-graded mesh depicted in figure 3.8 and monitor the evolution of the x- and y- components of $\|\mathbf{U}^n - \mathbf{U}\|_\infty$ in figure

3.9. As expected, the projection is stable.

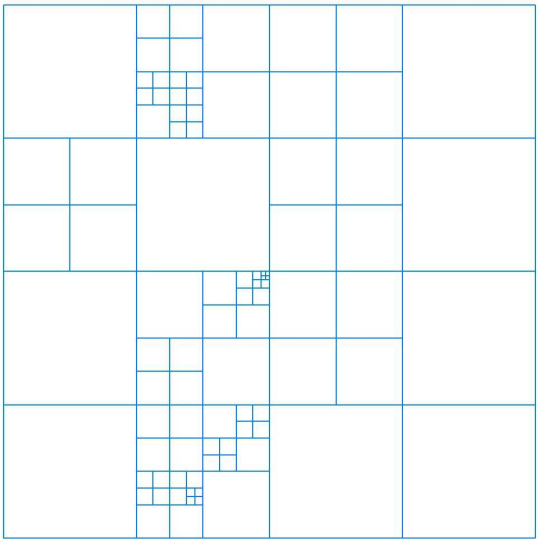


Figure 3.8: Highly non-graded mesh used to test the stability of the projection step.

We then employ the described method to solve the Poisson equation $\Delta\Phi = f$ on the domain $\Omega = [0,\pi]^2 \setminus C\{(\frac{\pi}{2},\frac{\pi}{2}), 0.8\}$ with $f(x,y) = -2\cos(x)\sin(y)$ and compare the results with the exact solution $\Phi_{exact}(x,y) = \cos(x)\sin(y)$. We enforce Neumann and Dirichlet boundary conditions and monitor the second-order convergence of our solver, given in figure 3.10.

3.4 Numerical examples

In this section we propose some validation examples in two and three spatial dimensions and demonstrate the efficiency of our solver. For each example, we also provide the runtime on an Intel i7-2600 CPU with 16 GiB RAM, compiled with gcc 4.8.2 on a linux kernel 3.13.0-24 and using the Petsc library 3.5.1 and the Voro++ library 0.4.6.

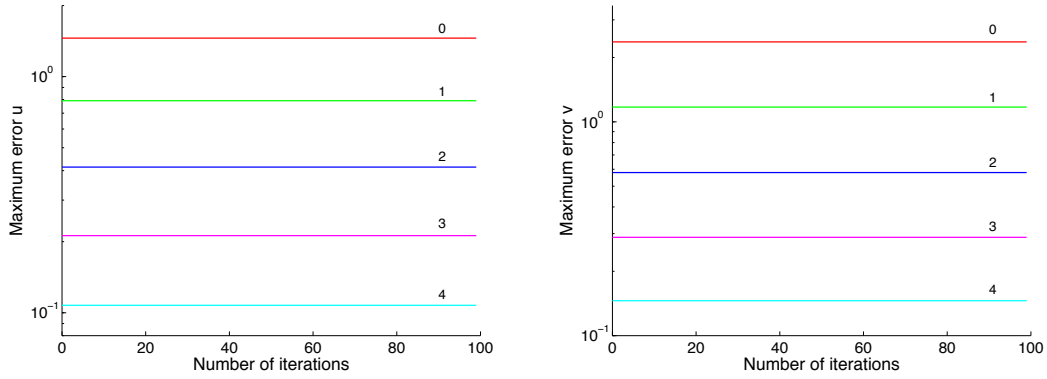


Figure 3.9: Left: x-component of $\|\mathbf{U}^n - \mathbf{U}\|_\infty$. Right: y-component of $\|\mathbf{U}^n - \mathbf{U}\|_\infty$. The numbers correspond to the number of recursive splitting applied to all the cells of the original mesh 3.8.

quadtree level	Neumann		Dirichlet	
(min / max)	L^∞ error	order	L^∞ error	order
4/7	$7.82 \cdot 10^{-3}$	-	$4.62 \cdot 10^{-3}$	-
5/8	$1.95 \cdot 10^{-3}$	2.00	$1.18 \cdot 10^{-3}$	1.97
6/9	$4.82 \cdot 10^{-4}$	2.02	$2.98 \cdot 10^{-4}$	1.99
7/10	$1.20 \cdot 10^{-5}$	2.01	$7.49 \cdot 10^{-5}$	1.99
8/11	$3.00 \cdot 10^{-6}$	2.00	$1.88 \cdot 10^{-5}$	2.00

Figure 3.10: Convergence of the cell-based poisson solver. The test solution is $f(x, y) = \cos(x)\sin(y)$ and the computational domain is $\Omega = [0, \pi]^2 \setminus C\{(\frac{\pi}{2}, \frac{\pi}{2}), 0.8\}$. The middle column presents the results for Neumann boundary conditions on the domain boundaries and on the interface, and the right column shows the results for Dirichlet boundary conditions on both the interface and the domain boundaries. We observe second-order convergence.

3.4.1 Examples in two spatial dimensions

We start by analysing the convergence of the solver on an analytic example before testing it on three classical benchmark problems: the driven cavity problem, the vortex shedding past a cylinder and the drag on a periodically oscillating cylinder.

Convergence analysis

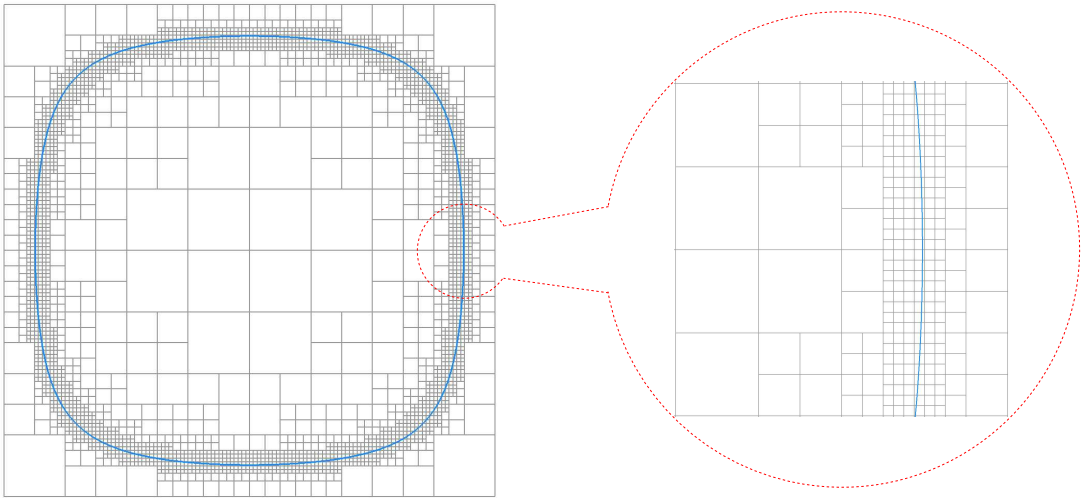


Figure 3.11: Computational domain and coarsest mesh used for example 3.4.1. The minimum and maximum resolutions of the quadtree are respectively 3 and 7. The zoom clearly depicts a non-graded part of the mesh.

In this first example we consider the following analytical solution for the Navier-Stokes equation:

$$
\begin{aligned}
u(x,y) &= \cos(t)\sin(x)\cos(y) \\
v(x,y) &= -\cos(t)\cos(x)\sin(y) \\
p(x,y) &= 0
\end{aligned}
$$

The geometry, depicted in figure 3.11, is defined by $\Omega^- = \{(x,y) \in [0,\pi]^2 | \sin(x)\sin(y) \geq .2\}$ reinitialized to a distance function, and we set $\mu = 1$, $\rho = 1$. We apply the appropriate forcing term

$$\begin{aligned} F_x =& -\rho \sin(t)\sin(x)\cos(y) + \rho\cos^2(t)\sin(x)\cos(x) + 2\mu\cos(t)\sin(x)\cos(y) \\ F_y =& \quad \rho \sin(t)\cos(x)\sin(y) + \rho\cos^2(t)\sin(y)\cos(y) - 2\mu\cos(t)\cos(x)\sin(y) \end{aligned}$$

and we monitor the convergence of the solver as the mesh is refined. The simulations are run until a final time $t_f = \frac{\pi}{3}$ with an adaptive time step defined at each iteration by $\Delta t_n = \frac{\Delta x}{\max\|\mathbf{u_n}\|}$. The whole simulation requires 9.61 seconds for a mesh of level 3/7.

A profiling of the first iteration of the solver on this example for a mesh of level 5/9 and with all parallel optimizations deactivated is presented in figure 3.12. Note that the extrapolation procedures take a significant time because they rely on the least square interpolations. We observe that a large portion of the time (approximately 56.3% of the total runtime) is spent interpolating quantities with the least square algorithm. Of course, this profiling is only indicative and would be different on a larger mesh since the various algorithms scale differently. In addition, some sections can be very efficiently parallelized (for instance the least square interpolation procedure which is embarassingly parallelizable). Finally, only the significant portions of the code are accounted for, which explains why the subsections of the program do not add up to 100%.

The results are reported in tables 3.1 and 3.2 and in figure 3.13. We also monitor the convergence of the Hodge variable as the procedure to enforce the boundary condition decribed in section 3.2.2 is iterated. The convergence of the Hodge variable over a typical time iteration on a mesh of level 5/9 is presented in table 3.3.

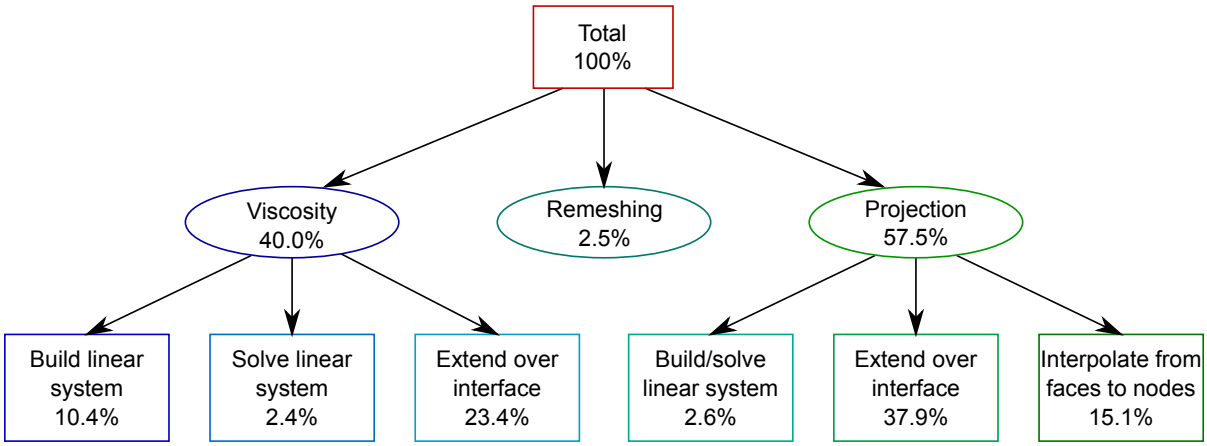


Figure 3.12: Analysis of the time spent in the various parts of the program for one iteration of example 3.4.1. The percentages correspond the fraction of the total runtime occupied by each section of the program. The extension procedures rely on the least square interpolations and therefore represent a large portion of the runtime. Note that only the most time consuming procedures are reported (we ignore 5.7% of the total procedure).

quadtree level (min / max)	L^1 error	order	L^∞ error	order
3/7	$5.48 \cdot 10^{-3}$	-	$2.18 \cdot 10^{-2}$	-
4/8	$1.76 \cdot 10^{-3}$	1.64	$8.65 \cdot 10^{-3}$	1.33
5/9	$5.56 \cdot 10^{-4}$	1.66	$3.28 \cdot 10^{-3}$	1.40
6/10	$1.61 \cdot 10^{-4}$	1.79	$1.64 \cdot 10^{-3}$	1.00
7/11	$3.62 \cdot 10^{-5}$	2.15	$3.84 \cdot 10^{-4}$	2.09
8/12	$8.59 \cdot 10^{-6}$	2.08	$2.45 \cdot 10^{-4}$	0.65

Table 3.1: Convergence of the x-component of the velocity for the example of section 3.4.1

quadtree level (min / max)	L^1 error	order	L^∞ error	order
3/7	$3.17 \cdot 10^{-4}$	-	$1.62 \cdot 10^{-3}$	-
4/8	$3.47 \cdot 10^{-5}$	3.19	$2.47 \cdot 10^{-4}$	2.71
5/9	$4.48 \cdot 10^{-6}$	2.96	$5.85 \cdot 10^{-5}$	2.08
6/10	$7.20 \cdot 10^{-7}$	2.64	$1.44 \cdot 10^{-5}$	2.02
7/11	$7.41 \cdot 10^{-8}$	3.28	$2.81 \cdot 10^{-6}$	2.36
8/12	$1.14 \cdot 10^{-8}$	2.70	$6.72 \cdot 10^{-7}$	2.06

Table 3.2: Convergence of the Hodge variable Φ for the example of section 3.4.1

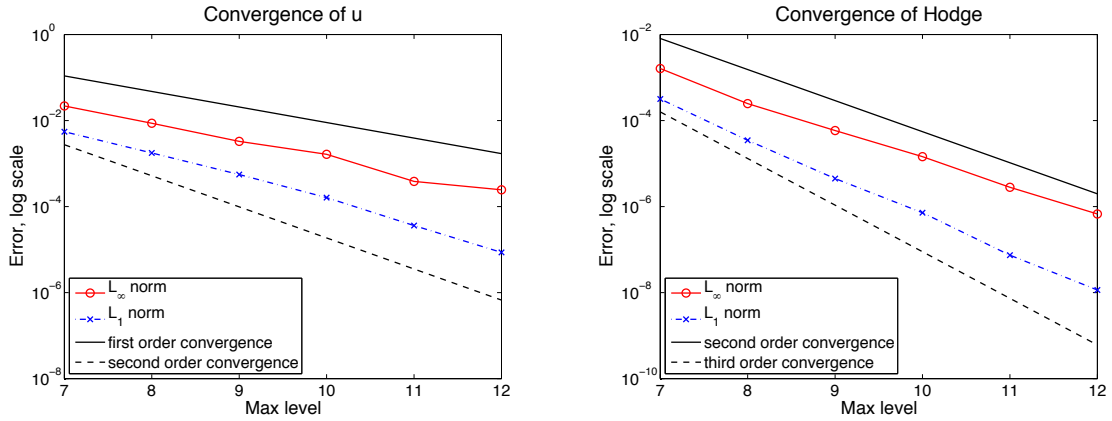


Figure 3.13: Visualization of the convergence of the error on the horizontal component of the velocity field and of the Hodge variable for example 3.4.1. We observe an order one for the x-velocity and two for the Hodge variable, both in L^∞ norm. The velocity is also second order accurate in the L^1 norm.

iteration	error
1	$1.19 \cdot 10^{-1}$
2	$3.13 \cdot 10^{-2}$
3	$6.37 \cdot 10^{-3}$
4	$3.15 \cdot 10^{-3}$
5	$2.16 \cdot 10^{-3}$
6	$1.56 \cdot 10^{-3}$

Table 3.3: Typical convergence of the Hodge variable on a mesh of level 5/9 as the algorithm to enforce the correct boundary conditions is iterated, i.e. after computing $\mathbf{u}^*$ and Φ successively as described in section 3.2.2. The error between two iterations is the relative error defined by $e = \max \frac{|\Phi^k - \Phi^{k+1}|}{|\Phi^{k+1}|}$.

Driven cavity

The driven cavity problem, for which Ghia *et al.* [112] have published experimental data, is one of the standard validation problems thoroughly studied in the literature. It is an excellent test for our solver when the domain does not contain any irregular interfaces. Consider a domain $\Omega = [0,1]^2$, with the top wall moving with unit velocity and a no-slip boundary condition for the velocity on the four walls. We choose a density $\rho = 1$ and Reynolds numbers $Re = 1000$ and $Re = 5000$. The refinement criterion we use is to refine where the gradient of velocity is large. Precisely, we compute $\gamma = \frac{1}{2}\left(\max_N |\nabla \mathbf{u}|\right) \cdot \Delta / \left(\max_{\Omega^-} \|\mathbf{u}\|\right)$, where Δ is the size of the local cell and N refers to the vertices of the local cell. We refine the cell if $\gamma > .01$.

Figure 3.14 shows the excellent agreement of the steady-state solution of our solver with the benchmark simulations found in [112] and [113], while figure 3.15 depicts the mesh and the streamlines of the steady-state solution for both Reynolds numbers. The full simulation takes 11 minutes for a mesh of level 6/8.

Karman vortex street

We propose to validate our solver with an irregular geometry through the analysis of the vortex shedding of a flow past a sphere, as first studied by Dennis and Chang [114]. The standard setup for this test consists of a cylinder of radius $r = .5$ located at $(8, 8)$ in a domain $\Omega = [0, 32] \times [0, 16]$, with an imposed velocity $u = U_\infty = 1$ on the left, top and bottom walls, and an homogeneous Neumann boundary condition for the velocity on the right wall. We impose homogeneous Neumann boundary conditions for the Hodge variable on the left, top and bottom wall, and for the problem to be well posed we impose a Dirichlet boundary condition on the right wall, which we set to zero.

The values for the drag and lift coefficients on the sphere, together with the Strouhal

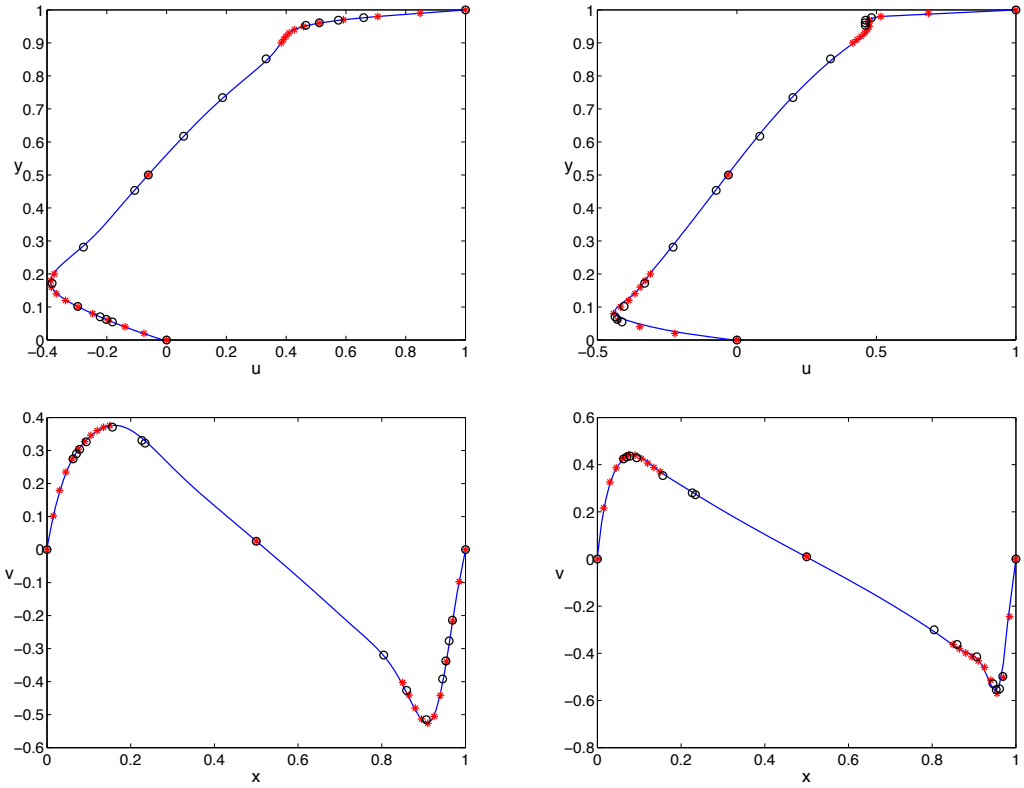


Figure 3.14: x- and y- components of the velocity field in the driven cavity from example 3.4.1 after 37 seconds. The black circles are the experimental results from Ghia *et al.* [112] and the red circles are taken from Erturk *et al.* [113]. The line represents the results obtained with our solver and with a time step $\Delta t = 5\Delta x$. The figures show the results for $Re = 1000$ on a quadtree of minimum level 6 and maximum level 8 on the left, and $Re = 5000$ on a quadtree of level 7/9 on the right.

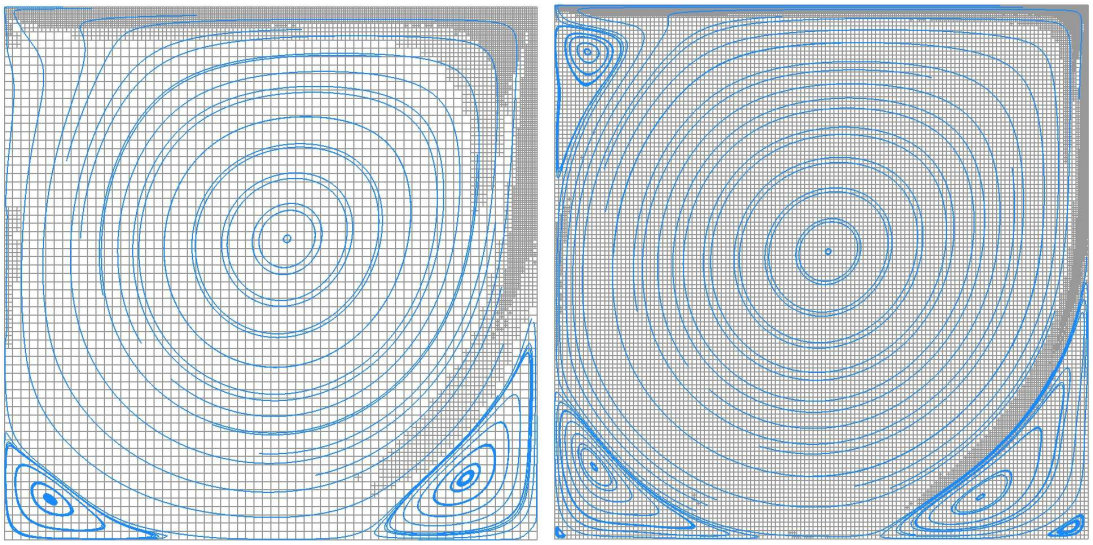


Figure 3.15: Visualization of the mesh (grey) and the streamlines (blue) for the driven cavity benchmark. Left: $Re = 1000$ with a mesh of level 6/8. Right: $Re = 5000$ with a mesh of level 7/9.

number, have been tabulated through numerical simulations in the literature, for example in [102, 115, 116, 117]. We compute the forces exerted on the cylinder via the geometric integration procedure of [24, 118] as:

$$\mathbf{F} = \int_{\Gamma} (-p + 2\mu \underline{\underline{\boldsymbol{D}}})\mathbf{n},$$

where Γ is the contour of the sphere, $\underline{\underline{\boldsymbol{D}}}$ is the symmetric stress tensor and $\mathbf{n}$ is the outward normal to the cylinder. The drag and lift coefficients are obtained respectively from the x- and y- components of $\mathbf{F}$ through scaling by $\rho r U_{\infty}^2$. The Strouhal number is computed by taking the Fourier transform of the lift coefficient history to obtain the main frequency f of the system and scaling it by $2r/U_{\infty}$. All the simulations in this section were performed with a time step $\Delta t = \Delta x / \max\|\mathbf{u}\|$. The refinement criterion is the same as in the driven cavity case, except that in addition, we impose a uniform band around the sphere in order to capture the boundary layer properly.

We observe that some numerical results presented by previous authors, for instance our previous work [102] and the references therein, correspond to an underesolved simulation and therefore do not give converged values for the lift and drag. We ran our solver with a uniform mesh, i.e. a quadtree with identical minimum and maximum levels, thus reproducing the method presented in [102], for various resolutions and observed the convergence of the drag coefficient. The convergence of the drag coefficient with the resolution of the mesh is presented in figure 3.16 and demonstrates the need for a high mesh resolution in order to obtain meaningful results. We observe that even with an adaptive mesh of maximum level 10, corresponding to a uniform resolution of 1024^2, the boundary layer is barely resolved for $Re = 100$, as illustrated in figure 3.17. Keeping this observation in mind, we compare the drag and lift coefficients we found, along with the Strouhal number, with the tabulated values from the literature. The results are reported in figure 3.19 and table 3.4 and 3.5 for $Re = 100$ and $Re = 200$, while the visual results are displayed in figure 3.18. In addition, we note that a time iteration with 100,000 leaves completes in 4.5 seconds.

Resolution	Drag coefficient C_d
7	1.250
8	1.341
9	1.390
10	1.404

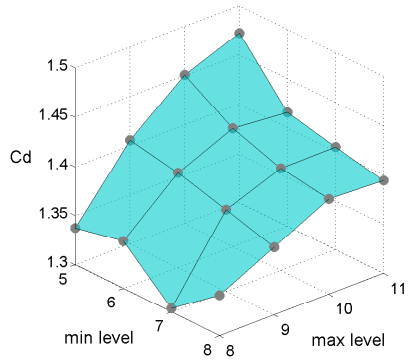


Figure 3.16: Left: convergence of the drag coefficient on a uniform mesh. Right: convergence of the drag coefficient on an adaptive mesh. For both cases, the resolution corresponds to the number of recursive splitting of the mesh, so that a resolution n for a uniform mesh leads to 2^{2n} cells. The Reynolds number is $Re = 100$.

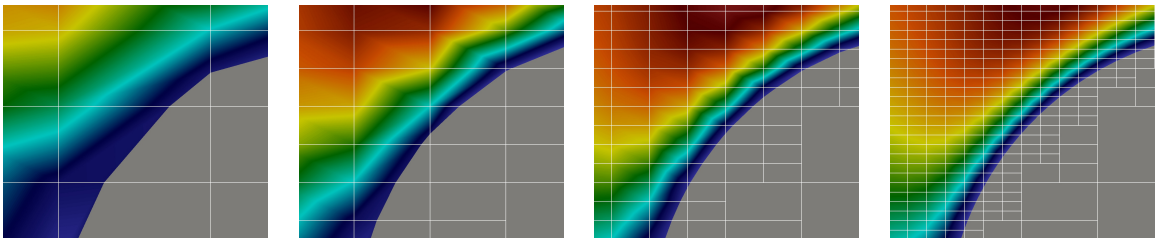


Figure 3.17: Illustration of the boundary layer on the surface of the cylinder for example 3.4.1. The snapshots are taken at time $t = 50$ for $Re = 100$ with an increasing maximum resolution of the adaptive mesh from 7 on the left, corresponding to a uniform mesh of 128^2, to 10 on the right, equivalent to a uniform resolution of 1024^2. This is also a good example of a non-graded mesh with a jump of three levels between neighbor cells for the highest resolution.

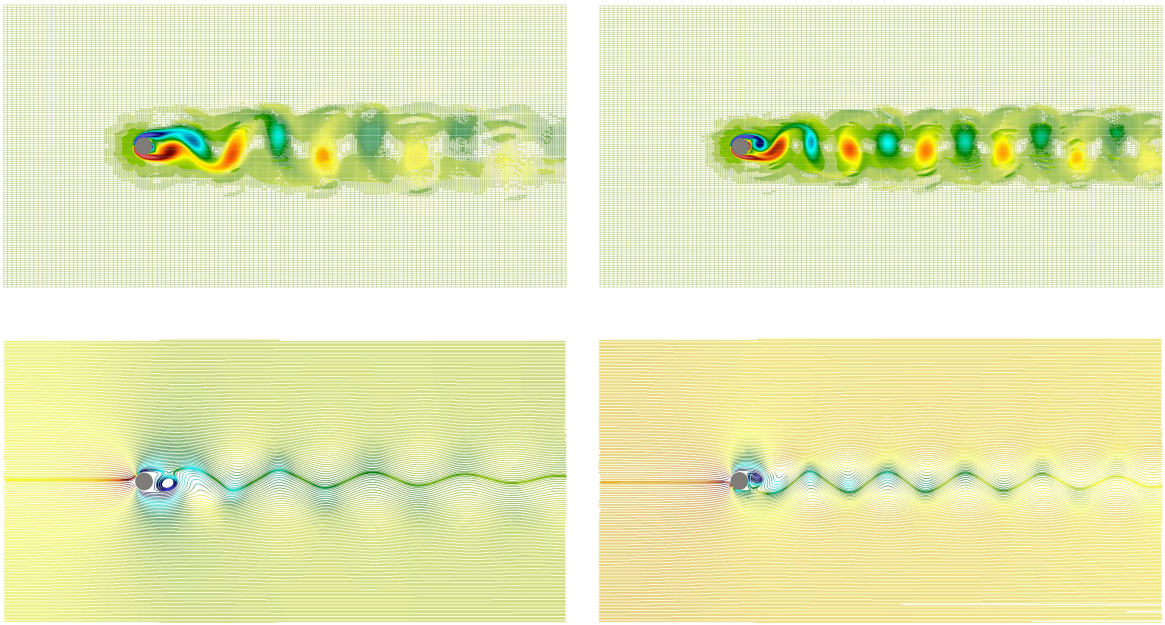


Figure 3.18: Visualization of the mesh and the vorticity (on top) and of the streamlines colored with the pressure (bottom), for example 3.4.1 with $Re = 100$ on the left and $Re = 200$ on the right, and a quadtree of level 7/10 in both cases.

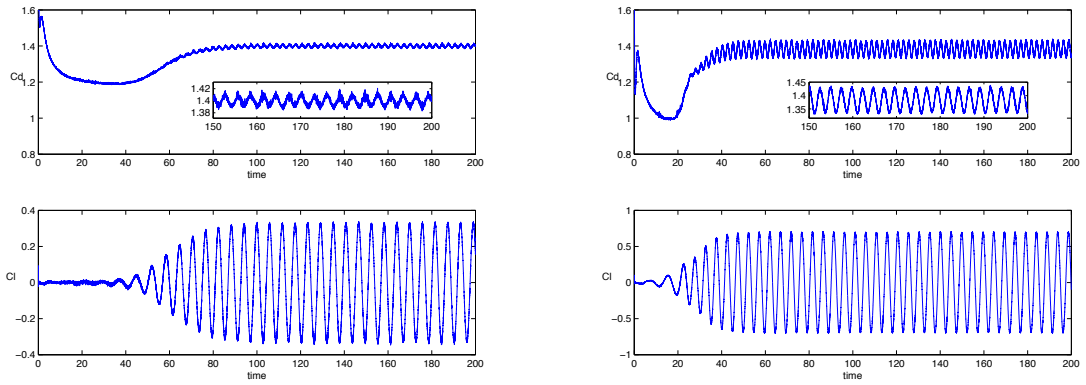


Figure 3.19: Evolution of the drag and lift coefficients for $Re = 100$ (left) and $Re = 200$ (right), for a quadtree of level 7/10.

	Drag coefficient C_d		Lift coefficient C_l	
	$Re = 100$	$Re = 200$	$Re = 100$	$Re = 200$
Ng *et al.* [102]	1.368 ± 0.016	1.373 ± 0.050	± 0.360	± 0.724
Braza *et al.* [115]	1.364 ± 0.015	1.40 ± 0.05	± 0.250	± 0.750
Calhoun [116]	1.330 ± 0.014	1.172 ± 0.058	± 0.298	± 0.668
Engelman *et al.* [119]	1.411 ± 0.010	–	± 0.350	–
Present	1.401 ± 0.011	1.383 ± 0.048	± 0.331	± 0.705

Table 3.4: Drag and lift coefficients for example 3.4.1

	$Re = 100$	$Re = 200$
Rosenfeld *et al.* [120]	-	0.20
Braza *et al.* [115]	0.16	0.20
Mahir *et al.* [121]	0.172	0.192
Laroussi *et al.* [117]	0.173	0.199
Muddada *et al.* [122]	0.170	-
Present	0.172	0.203

Table 3.5: Strouhal number for example 3.4.1

Oscillating cylinder

A popular test to demonstrate the ability of a solver to handle moving complex interfaces is that of a periodically oscillating cylinder in a viscous fluid. Two non-dimensional parameters characterize the flow, the Reynolds number and the Keulegan-Carpenter number. The Reynolds number is defined like previously as $Re = 2r\rho u_m/\mu$ with u_m the maximum velocity of the cylinder and the Keulegan-Carpenter number is defined as $KC = u_m/2rf$ where f is the frequency of the oscillation. We consider a cylinder of radius $r = 0.05$ in a domain $\Omega = [-1, 1]^2$ oscillating in the $x-$direction so that at time t, the cylinder is centered on $(x_c, 0)$ with

$$x_c = X_0(1 - \cos(2\pi f t))$$

where $X_0 = 0.7957D$ (with $D = 2r$) is the amplitude of the oscillation. We enforce Dirichlet boundary conditions on the edge of the computational domain and set $Re = 100$ and $KC = 5$. We compute the forces on the cylinder by direct integration as described in the previous section. For this example, we iterate our solver twice at each time step as described in section 3.2.2 in order to properly enforce the moving boundary condition. We choose an adaptive mesh of minimum level 6 and maximum level 12, and we set the adaptive time step as $\Delta t = 2\Delta x/\max\|\mathbf{u}\|$ where Δx is the size of the smallest cell. One time iteration with a mesh consisting of 119,086 leaves necessitates 15.89 seconds. Note that a uniform mesh with the same maximum resolution would have over 67 billion cells. The history of the drag coefficient in the oscillatory direction is presented in figure 3.20 together with some reference data from [123, 124, 125].

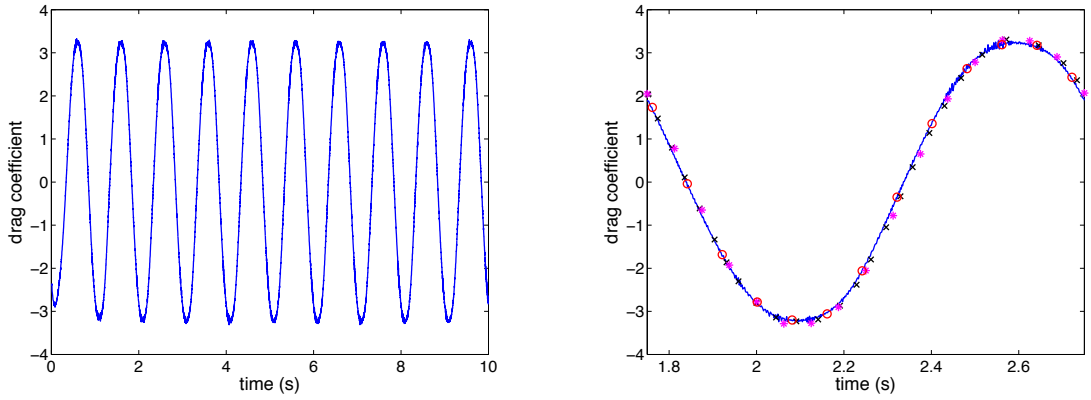


Figure 3.20: History of the drag coefficient for an oscillating cylinder with $Re = 100$ and $KC = 5$. The black crosses are the original experimental data from Dutsch *et al.* [123], the red circles correspond to the data from Seo and Mittal [124] and the purple stars were computed by Liao *et al.* [125].

3.4.2 Examples in three spatial dimensions

Flow past a sphere

Our first three-dimensional example is a flow past a sphere. We use the same setup as in two spatial dimensions, i.e. a sphere of radius $r = 0.5$ centered at $(8, 0, 0)$ in a domain $\Omega = [0, 32] \times [0, 8] \times [0, 8]$, and compute the drag coefficient and the Strouhal frequency when applicable. The drag coefficient is computed the same way as for the two dimensional case by direct integration [24, 118]:

$$C_D = \frac{\mathbf{F}_x}{\frac{1}{2}\rho \mathcal{A} U_\infty^2} = \frac{1}{\frac{1}{2}\rho \mathcal{A} U_\infty^2} \int_\Gamma (-p + 2\mu \underline{\underline{\boldsymbol{D}}}) \mathbf{n}_x,$$

where $\mathcal{A} = \pi r^2$ is the cross section of the sphere, $U_\infty = 1$ is the imposed inlet velocity, μ is the viscosity, $\rho = 1$ is the density, p is the pressure and $\underline{\underline{\boldsymbol{D}}}$ is the symmetric stress tensor. The Strouhal number is obtained as in the two-dimensional case. The four simulations were performed on an octree of minimum level 6 and maximum level 10 refined according

to the same rule as in the two dimensional case. The $Re = 350$ case generated the largest mesh with up to 2.85 million leaf cells, corresponding to 0.27% of the number of cells of the equivalent uniform mesh. One iteration of the solver on a mesh with 1.3 million leaves completed in 3 minutes.

The results are presented in tables 3.6 and 3.7 and figures 3.21 and 3.22. We observe a good agreement with the literature. For a Reynolds number of 350, we observe two distinct frequency peaks in the Fourier analysis (see figure 3.21). The main frequency is reported in table 3.7 while the second fundamental frequency we find is 0.047.

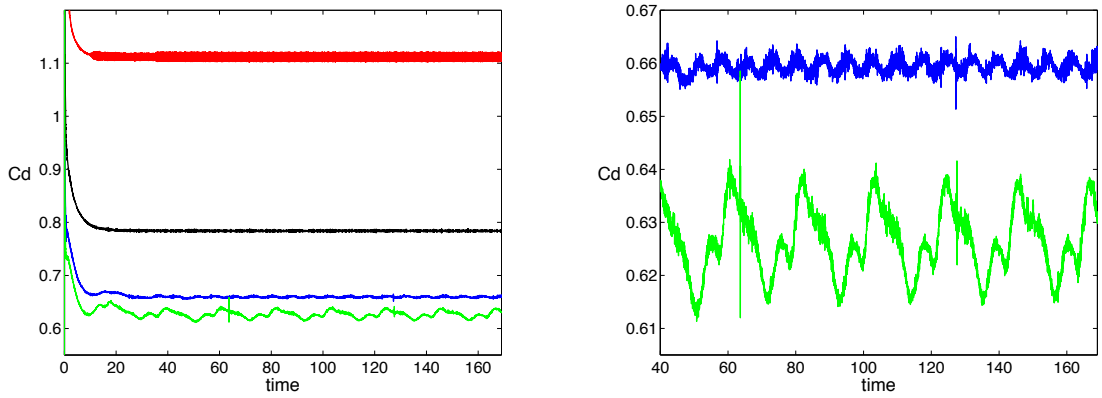


Figure 3.21: The left figure shows the drag coefficients for $Re = 100$ (red), $Re = 200$ (black), $Re = 300$ (blue) and $Re = 350$ (green). The right figure presents a close up of the oscillatory behavior for Reynolds numbers 300 and 350.

	$Re = 100$	$Re = 200$	$Re = 300$	$Re = 350$
Mittal *et al.* [126]	1.08	-	0.67	0.62
Marella *et al.* [127]	1.06	-	0.621	-
Le Clair *et al.* [128]	1.096	0.772	0.632	-
Johnson and Patel [129]	1.1	0.8	0.656	-
Present	1.112	0.784	0.659	0.627

Table 3.6: Average drag coefficients for example 3.4.2

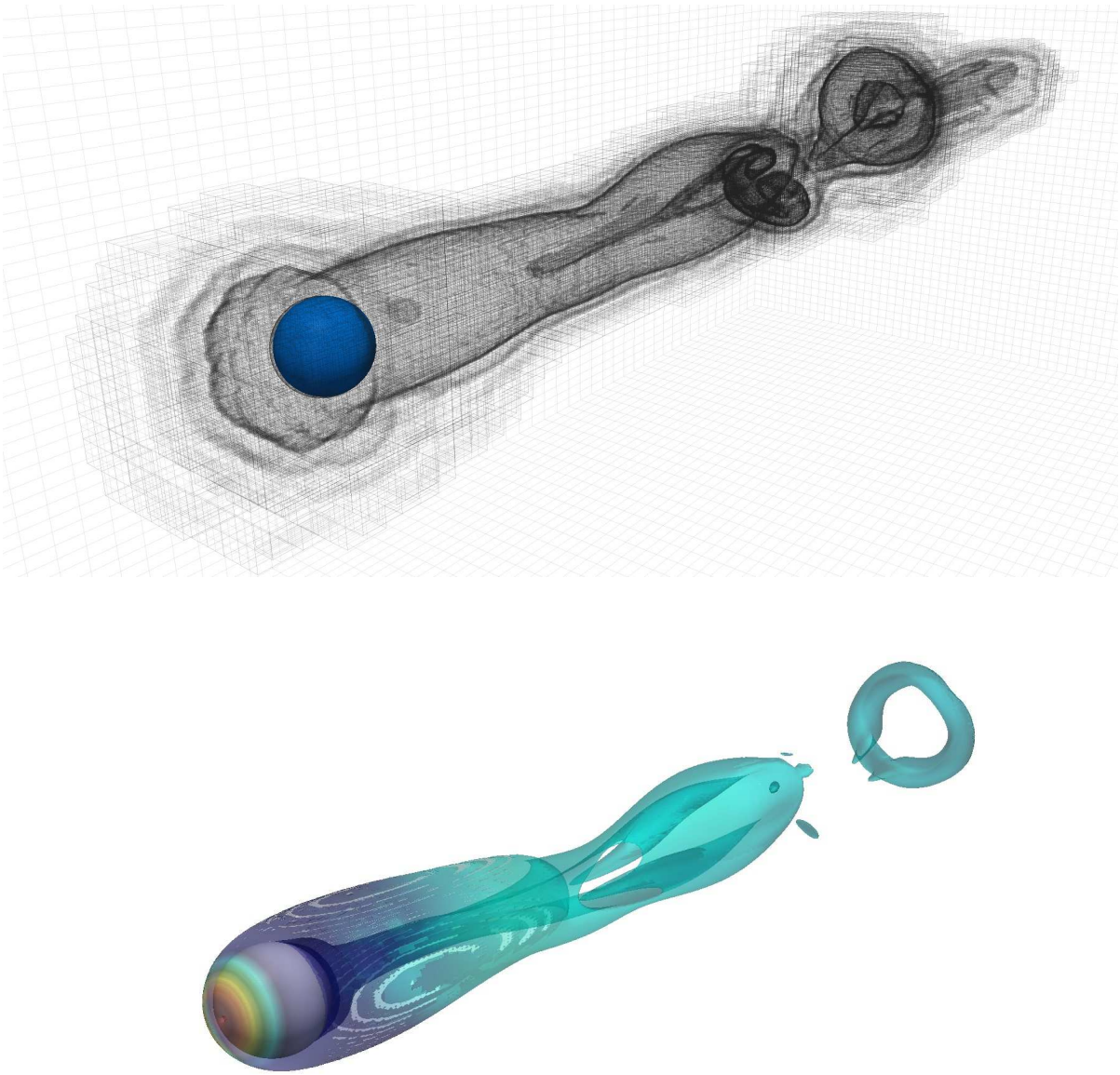


Figure 3.22: Visualization of the computational mesh (top) and of a vorticity contour colored with the pressure values (bottom) for $Re = 350$.

	$Re = 300$	$Re = 350$
Mitall *et al.* [126]	0.135	0.142
Marella *et al.* [127]	0.133	-
Johnson and Patel [129]	0.137	-
Bagchi *et al.* [130]	-	0.135
Present	0.137	0.141

Table 3.7: Strouhal frequencies for example 3.4.2

Flow through a porous medium: Darcy's law

Monophasic viscous flows through a porous medium have been shown to follow Darcy's law, which states that the flux q through the medium is proportional to the gradient of pressure, ∇P, i.e.,

$$q = \frac{-k}{\mu} \nabla P,$$

where k is the intrinsic permeability of the porous medium and μ is the viscosity of the fluid. In general, this relation is used experimentally to find the permeability by applying a pressure gradient and measuring the flow rate.

We propose to reproduce Darcy's law numerically. In order to do so, we choose a porous geometry in a domain $\Omega = [-8, 8] \times [-8, 8] \times [0, 32]$, shown in figure 3.23. We then enforce a pressure gradient through Dirichlet boundary conditions on the pressure on the x-axis walls and we measure the corresponding flux at the location $x = -6$. We impose a no-slip condition on the irregular geometry and on the y- and z-walls, and homogeneous Neumann boundary conditions on the x-walls for the velocity field. We repeat the process for a range of pressure values and observe the linear relationship between the pressure gradient and the flux for a viscosity $\mu = 20$ and a pressure gradient ranging from $3.1 \cdot 10^{-3}$ to 15.625. We also compute the Reynolds number for the various pressure gradients, which we define as $Re = \frac{2\rho q \tilde{r}}{\mu \mathcal{A}}$, where $\mathcal{A}$ is the area of the cross-section at $x = -6$, $\tilde{r}$ is the average hydraulic radius, and $\rho = 1$ is the density of the fluid. We

monitor the Reynolds number to make sure that we are running the system in a range of parameters to which Darcy's law applies (see figure 3.26).

All the simulations were done on an adaptive mesh of minimum level 3 and maximum level 8, resulting in $3,126,928$ cells. We use only 18.64% of the cells that the equivalent uniform mesh would have to represent very accurately this complex and space-filling geometry. Note that as the maximum level of the mesh increases, this ratio becomes more and more advantageous. A single time iteration completed in fifteen minutes. We ran each simulation long enough for the flow to reach the steady-state regime, as demonstrated in figure 3.24. The results are reported in figure 3.25 and 3.26.

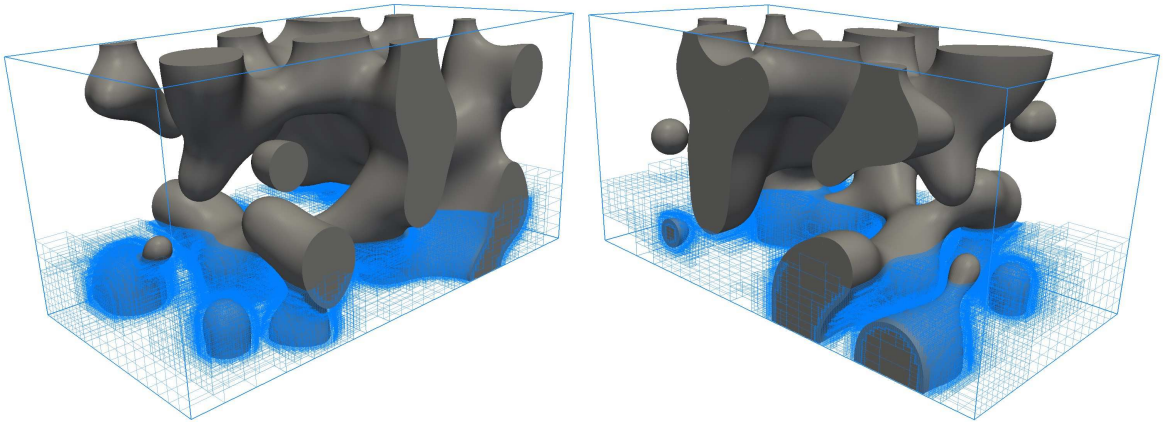


Figure 3.23: Representation of the geometry (grey) used for the three dimensional porous flow simulation 3.4.2 seen from two different angles. Part of the adaptive grid is depicted in blue.

Flow around a deformable object

We propose a simulation of the flow past a swimming great white shark. Note that the Reynolds number for this problem, of the order of 10^6, is far too large for the boundary layer to be resolved properly with our method on a single workstation. However, it still illustrates the ability of our solver to handle high Reynolds number flows around a

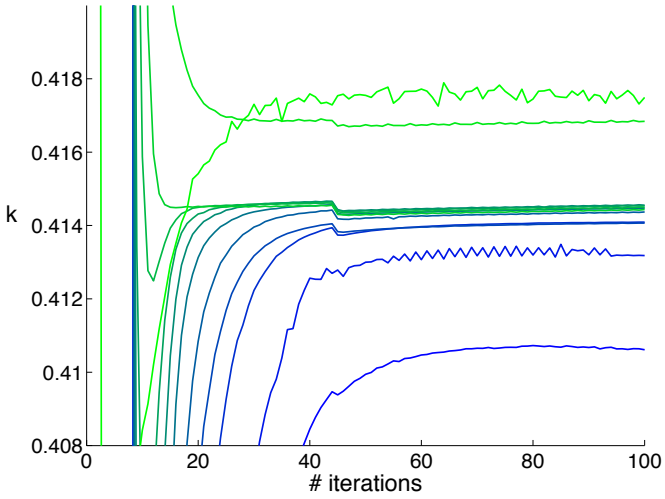


Figure 3.24: Convergence of the numerical solution of the Navier-Stokes equations towards a steady state for the porous flow example 3.4.2. The graph shows the evolution of the permeability coefficient k as a function of the number of iterations, with colors corresponding to various inlet pressures ranging from $P_{in} = 0.1$ for the bottom dark blue curve to $P_{in} = 500$ for the top light green curve. The imposed outlet pressure is $P_{out} = 0$.

deformable irregular geometry without loss of stability. The visual results are presented in figure 3.27.

3.5 Summary

We presented a method for the simulation of incompressible viscous flows on non-graded adaptive Cartesian meshes. The implicit discretization of the viscous term combined with a semi-Lagrangian approach for the advection term enables large time steps, while the discretization of the projection step is proven to be stable, regardless of the type of boundary conditions enforced or the ratio between adjacent cells. We validated the solver in both two and three spatial dimensions on benchmark cases, and demonstrated the strength of using an adaptive mesh in terms of efficiency. This work can be used as a building block for the numerical study of more complex flows, such as free surface flows

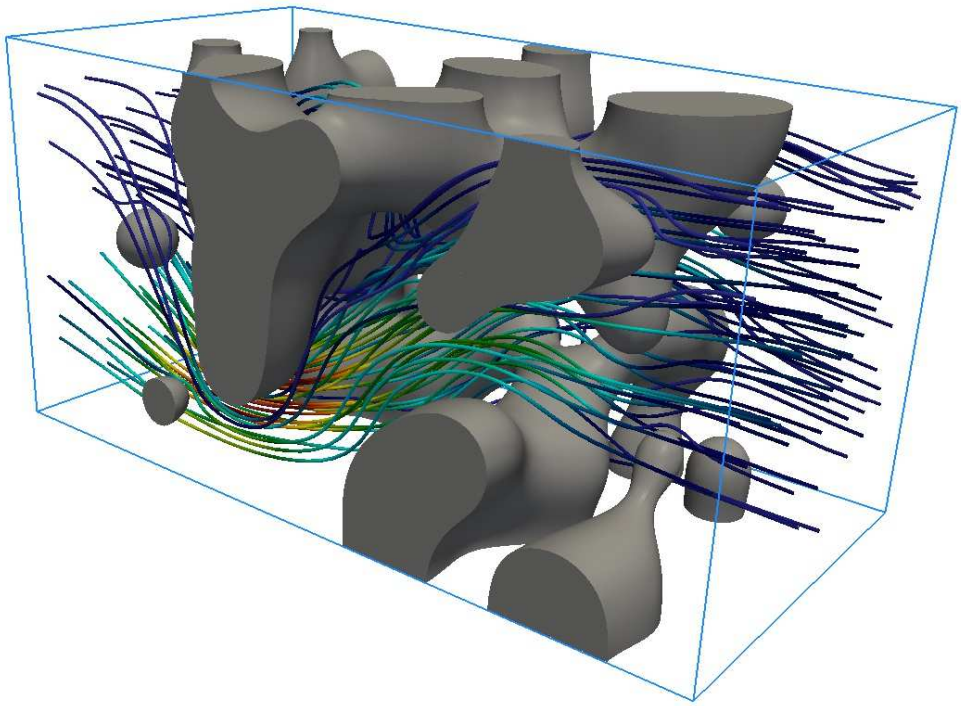


Figure 3.25: Representation of the streamlines through the porous geometry for example 3.4.2. The imposed pressure is 1 at the inlet and 0 at the outlet. The colors show the magnitude of the velocity field along the streamlines, ranging from 0 (in blue) to $8 \cdot 10^{-3}$ (in red).

P_{in}	k	Re
.1	0.4107	$1.09 \cdot 10^{-5}$
.2	0.4134	$2.20 \cdot 10^{-5}$
.5	0.4141	$5.50 \cdot 10^{-5}$
1	0.4141	$1.10 \cdot 10^{-4}$
2	0.4143	$2.20 \cdot 10^{-4}$
5	0.4143	$5.50 \cdot 10^{-4}$
10	0.4145	$1.10 \cdot 10^{-3}$
20	0.4145	$2.20 \cdot 10^{-3}$
50	0.4145	$5.50 \cdot 10^{-3}$
100	0.4144	$1.10 \cdot 10^{-2}$
200	0.4168	$2.21 \cdot 10^{-2}$
500	0.4175	$5.55 \cdot 10^{-2}$

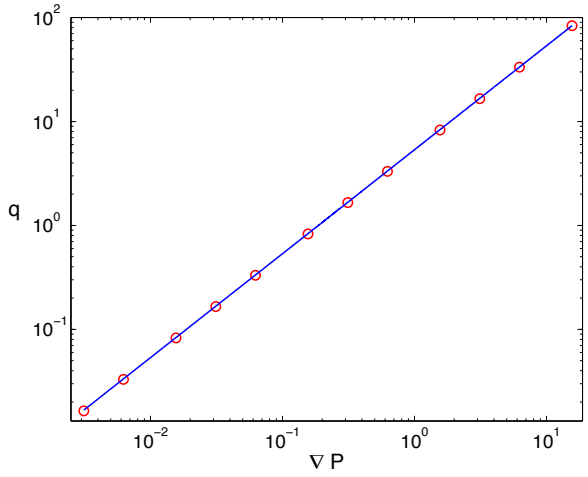


Figure 3.26: Left: values of the intrinsic permeability coefficient k and the computed Reynolds number Re for various imposed inlet pressures P_{in}, with the outlet pressure 0. Right: representation of the observed flux as a function of the imposed pressure gradient. The red circles are the experimental data and the blue line is the linear regression. The relation is clearly linear, and we find an intrinsic permeability coefficient of 0.414 for the chosen geometry.

or multiphase flows, or in the coupling with other physical phenomena.

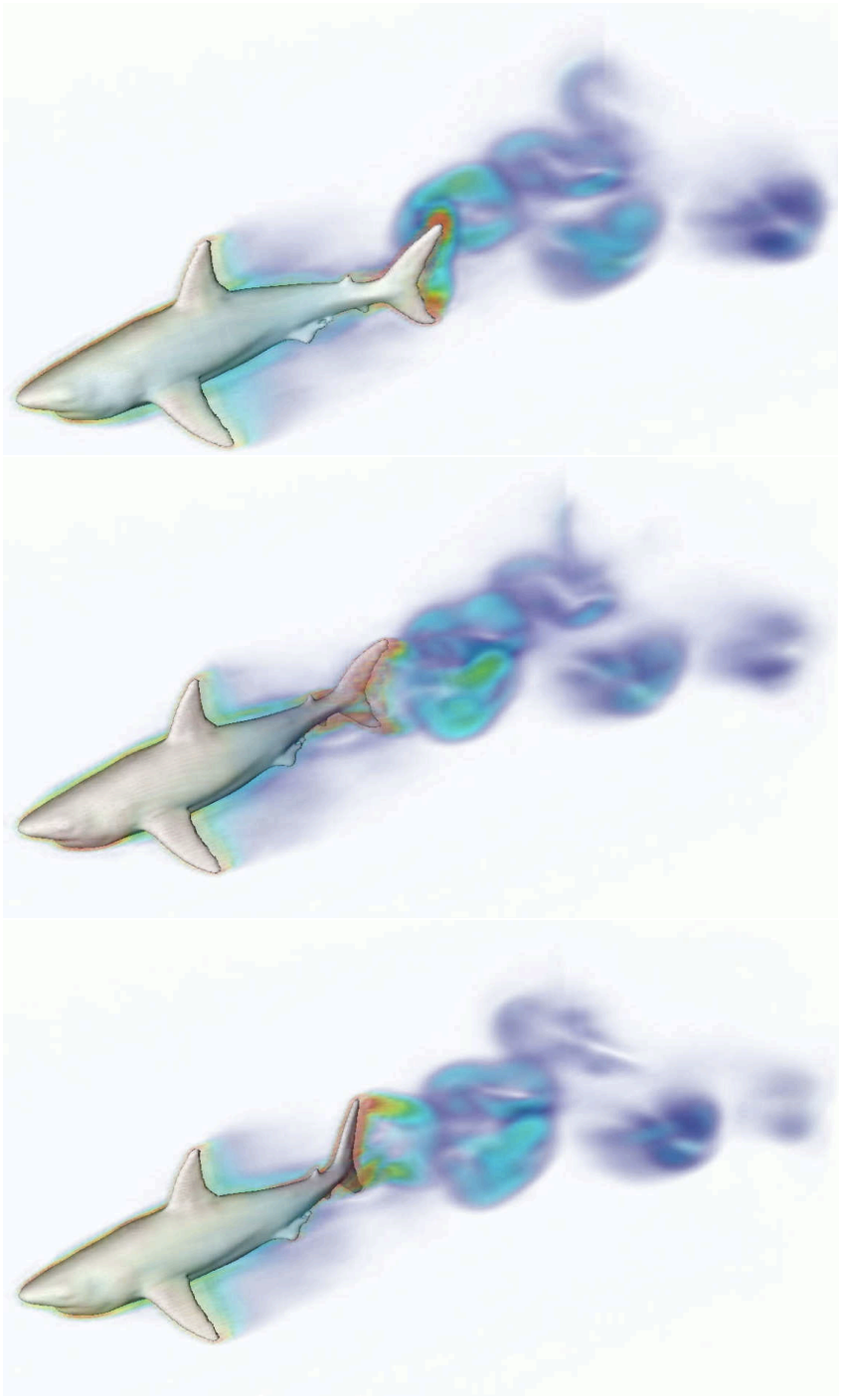


Figure 3.27: Vorticity generated by a swimming great white shark.

Chapter 4

Extension of the incompressible fluid solver to parallel environments

4.1 Introduction

In the last decade, the democratization of the access to supercomputers has prompted the development of massively parallel simulation techniques. The previously existing serial codes are progressively being adapted to exploit the hundreds of thousands of cores available through the main computing clusters. We propose a parallel implementation of the solver for the incompressible Navier-Stokes equations introduced in [2] based on the parallel level-set framework presented in [131].

Numerical simulations at the continuum scale are generally divided into two categories characterized by their meshing technique. On the one hand, the finite elements community relies on body-fitted unstructured meshes to represent irregular domains. Given a high quality mesh, the resulting solvers are fast and very accurate and this approach has been successfully applied to the simulation of incompressible viscous flows [76, 132, 133]. However, the mesh generation is very costly and impractical when tracking moving interfaces and fluid features. On the other hand, methods based on structured Cartesian grids render the mesh geometry mainly trivial but lead to a higher complexity for the implicit

representation of irregular interfaces. We focus here on the latter class of methods.

A common approach to represent an irregular interface in a implicit framework is to use Peskin's immersed boundary method [99, 134, 100] or its extension, the immersed interface method [135]. However, these methods introduce a smoothing of the interface through a delta formulation and therefore restrict the accuracy of the solution to first order in the L^∞-norm near the boundaries. We therefore opt for the sharp interface representation provided by the level-set function [4]. The level-set function can be used in several ways to implement the boundary conditions on an irregular interface, such as with the rasterization approach described in Losasso *et al.* [13] or the Heaviside formulation of Batty *et al.* [101]. However, these treatments were shown to be inconsistent in the L^∞-norm by Ng *et al.* [102] and we choose the finite-volume/cut-cell approach they suggested.

Fluid flows are by nature multiscale, thus limiting the scope of uniform Cartesian grids. A range of strategies have been proposed to leverage the spatial locality of the fluid information such as stretched grids [80, 81], nested grids [82, 83, 84, 85], chimera grids [136, 106] or unstructured meshes [86, 87, 88, 89, 90]. Another approach is to use a Quadtree [137] (in two spatial dimensions) or Octree [138] (in three spatial dimensions) data structures to store the mesh information [139, 140]. Popinet applied this idea combined with a non-compact finite volume discretizations on the Marker-And-Cell (MAC) configuration [96] to the simulation of incompressible fluid flows [12]. Losasso *et al.* also proposed a compact finite volume solver on Octree for inviscid free surface flows [13] while Min *et al.* presented a node-based second order viscous solver [98]. The present work is based on the work presented in [2] which solves the viscous Navier-Stokes equations implicitly on the MAC configuration using a Voronoi partition and where the advection part of the momentum equation is discretized along the characteristic with a BDF semi-Lagrangian scheme [141, 23].

The extension of [2] to parallel architectures relies on the existence of an efficient parallel Octree structure. Possible ways to implement parallel tree structures include the replication of the entire grid on each process. This approach, however, is not feasible when the grid size exceeds the memory of a single compute node, which must be considered a common scenario today. Using graph partitioners such as `parMETIS` [33] on a tree structure would discard the mathematical relations between neighbor and child elements that are implicit in the tree, and thus result in additional overhead. Another option, which we find preferable, is to exploit the tree's logical structure using space-filling curves [37]. This approach has been shown to lead to load balanced configurations with good information locality for a selection of space-filling curves including the Morton (or Z-ordering) curve and the Hilbert curve [39].

Space-filling curves have been used in several ways, for example augmented by hashing [38], tailored to PDE solves [142], or focusing on optimized traversals [143]. `Octor` [40] and `Dendro` [42] are two examples of parallel Octree libraries making use of this strategy that have been scaled to 62,000 [41] and 32,000 [43] cores, operating on parent-child pointers and a linearized octant storage, respectively. Extending the linearized storage strategy to a forest of interconnected Octrees [36, 31], the `p4est` library [44] provides a publicly available implementation of the parallel algorithms required to handle the parallel mesh, including an efficient 2:1 balancing algorithm [70]. `p4est` has been shown to scale up to over 458k cores [46], with applications using it successfully on 1.57M cores [47] and 3.14M cores [48].

The algorithms pertaining to the second order level-set method on Octree presented in [14] have been extended to the `p4est` data structure in [131] and shown to scale up to 4,096 cores. Starting from this existing basis for the level-set function procedures, we present the implementation of the algorithms pertaining to the simulation of incompressible fluid flows detailed in [2]. The Voronoi tesselation that we construct over the

adaptive tree mesh requires two layers of ghost cells, whose efficient parallel construction we describe in detail. We report on the scalability of the algorithms presented before illustrating the full capacities of the resulting solver.

4.2 The computational method

We now briefly present the mathematical approach employed to solve the incompressible Navier-Stokes equations on a Quadtree grid. The extension to Octree grids is straightforward. We use the method described in [2] and encourage the reader to refer to it for a more complete description.

4.2.1 Representation of the spatial information

The level-set method

A central desired feature of the proposed solver is to be able to handle complex, possibly deforming, sharp interfaces. The level-set framework, first introduced by [4] and extended to Quadtrees in [11], is the perfect tool for such a goal. The level-set representation of an arbitrary contour Γ separating a domain Ω into two subdomains Ω^- and Ω^- consists in defining a function ϕ, called the level-set function, such that $\Gamma = \{\underline{\boldsymbol{x}} \in \mathbb{R}^n | \phi(\underline{\boldsymbol{x}}) = 0\}$, $\Omega^- = \{\underline{\boldsymbol{x}} \in \mathbb{R}^n | \phi(\underline{\boldsymbol{x}}) < 0\}$ and $\Omega^+ = \{\underline{\boldsymbol{x}} \in \mathbb{R}^n | \phi(\underline{\boldsymbol{x}}) > 0\}$.
Among all the possible candidates that satisfy these critera, a signed distance function is the most convenient one. In order to transform any function into a signed distance function that shares the same zero contour, one can solve the reinitialization equation

$$\frac{\partial \phi}{\partial \tau} + \text{sign}(\phi)\left(|\nabla \phi| - 1\right) = 0,$$

where τ is a fictitious time. The finite difference discretization and its corresponding parallel implementation employed to solve this equation are presented respectively in [14] and [131].

Octrees and the `p4est` library

When dealing with physical problems that exhibit a wide range of length scales, uniform Cartesian meshes become impractical in that capturing the smallest length scales require a very high resolution. This is the case for high Reynolds number flows for which the boundary layers and wake vortices have a length scale significantly smaller than that of the far field flow. This observation naturally leads to the use of adaptive Cartesian grids, including Octrees grids.

The `p4est` library [44] is a collection of parallel algorithms that handles a linearized tree data structure and its manipulation methods and has shown to scale up to 458,752 cores [46]. In `p4est` the domain is first divided by a coarse grid, which we will refer to as the "macromesh", common to all the processes. While this macromesh can represent complex domains, for our purpose we will consider solely uniform Cartesian macromeshes in a brick layout. Such a layout can be constructed at no cost using predefined and self-contained functions. A collection of trees rooted in each cell of the macromesh is then constructed and partitioned and their associated expanded ghost layers are generated. The refinement and coarsening criteria necessary for the construction of the trees are provided to `p4est` through defining callback functions. We propose to use three criteria based on the physical characteristics at hand. Different combinations of these criteria is used depending on the circumstances.

The first criterion, presented in [14] and [131], captures the location of the interface.

Specifically, a cell C is marked for refinement if

$$\min_{v \in V(C)} |\phi(v)| \leq \frac{\text{Lip}(\phi) \cdot \text{diag}(C)}{2},$$

where $V(C)$ is the set of all the vertices of cell C, $\text{Lip}(\phi)$ is the Lipschitz constant of the level-set function ϕ, and $\text{diag}(C)$ is the length of the diagonal of cell C. Similarly, a cell is marked for coarsening if

$$\min_{v \in V(C)} |\phi(v)| > \frac{\text{Lip}(\phi) \cdot \text{diag}(C)}{2}.$$

The second criteria, introduced in [12] and used in [98] and [2], is based on the vorticity of the fluid. High vorticity correspond to small length scales and therefore necessitate a high mesh resolution. We mark a cell C for refinement if

$$\text{size}(C) \cdot \frac{\max_{v \in V(C)} \|\nabla \times \underline{\boldsymbol{u}}(v)\|_2}{\max_{\Omega} \|\underline{\boldsymbol{u}}\|_2} \geq \gamma,$$

where $\text{size}(C)$ is the length of cell C and γ is a parameter controlling the level of refinement. Analogously, a cell C is marked for coarsening if

$$\text{size}(C) \cdot \frac{\max_{v \in V(C)} \|\nabla \times \underline{\boldsymbol{u}}(v)\|_2}{\max_{\Omega} \|\underline{\boldsymbol{u}}\|_2} < \gamma.$$

Finally, we introduce a passive marker in the fluid for visualization in section 4.5.4. For enhanced graphical results, we propose to refine the mesh where the density of the marker exceeds a threshold. Given a density $\beta \in [0, 1]$, a cell C is marked for refinement if

$$\max_{v \in V(C)} \beta(v) \geq \delta,$$

where δ is a parameter controlling the level of refinement. Conversely, a cell is marked

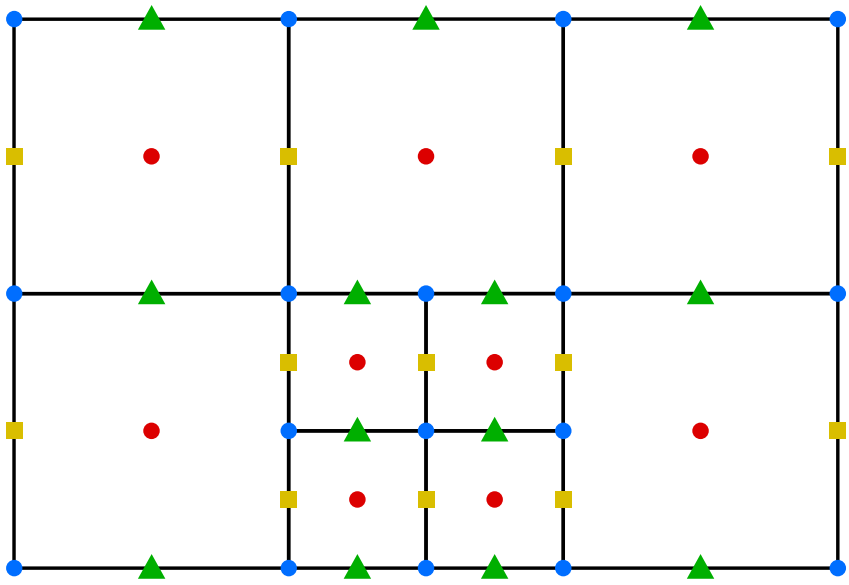


Figure 4.1: Representation of the Marker-And-Cell (MAC) data layout on a Quadtree structure with the location of the x-velocity (■), the y-velocity (▲), the Hodge variable (●) and the level-set values (●).

for coarsening if

$$\max_{v \in V(C)} \beta(v) < \delta.$$

The Marker-And-Cell layout

The standard data layout used to simulate incompressible viscous flows on uniform grids is the Marker-And-Cell (MAC)[96]. The analogous layout for Quadtrees is presented in figure 4.1 and leads to complications in the discretizations compared to uniform grids. However, second order accuracy is achievable and two possible corresponding discretizations are presented for the data located at the center of the cells (the leaves of the trees) and at their faces, respectively, in sections 4.2.4 and 4.2.3.

4.2.2 The projection method

Consider the incompressible Navier-Stokes equations for a fluid with velocity $\underline{\boldsymbol{u}}$, pressure p, density ρ and dynamic viscosity μ,

$$\rho\left(\frac{\partial \underline{\boldsymbol{u}}}{\partial t} + \underline{\boldsymbol{u}} \cdot \nabla \underline{\boldsymbol{u}}\right) = -\nabla p + \mu \Delta \underline{\boldsymbol{u}}, \tag{4.1}$$

$$\nabla \cdot \underline{\boldsymbol{u}} = 0. \tag{4.2}$$

The standard approach to solve this system is the projection method introduced by Chorin in 1967 [94]. We refer the reader to [95] for a review of the variation of the projection method. The system is decomposed into two distinct steps, identified as the viscosity step and the projection step. The first step consists in solving the momentum equation (4.1) without the pressure term,

$$\rho\left(\frac{\partial \underline{\boldsymbol{u}}}{\partial t} + \underline{\boldsymbol{u}} \cdot \nabla \underline{\boldsymbol{u}}\right) = \mu \Delta \underline{\boldsymbol{u}}, \tag{4.3}$$

to find an intermediate velocity field $\underline{\boldsymbol{u}}^*$. Since this field does not satisfy the incompressibility condition (4.2), it is then projected on the divergence-free subspace to obtain $\underline{\boldsymbol{u}}^{n+1}$:

$$\underline{\boldsymbol{u}}^{n+1} = \underline{\boldsymbol{u}}^* - \nabla \Phi. \tag{4.4}$$

where Φ is referred to as the Hodge variable. The following sections describe the discretization applied to solve these two steps.

4.2.3 Implicit discretization of the viscosity step

The viscosity step (4.3) contains two distinct terms, the advection term on the left hand side and the diffusion term on the right hand side. We now present their respective

discretizations.

Discretization of the advection term with a semi-Lagrangian approach

We discretize the advection part of the viscosity step using a semi-Lagrangian approach [141] combined with a second order backward difference discretization [23]. This method relies on the fact the solution of the advection equation

$$\frac{\partial \underline{\boldsymbol{u}}}{\partial t} + \underline{\boldsymbol{u}} \cdot \nabla \underline{\boldsymbol{u}} = 0 \tag{4.5}$$

is constant along the characteristics of the equation. In other words, along a characteristic of equation (4.5) parametrized by ($\mathbf{x}$(s), t(s)), one can write

$$\frac{d\underline{\boldsymbol{u}}}{ds} = 0. \tag{4.6}$$

Applying the second order backward difference formula to find $\underline{\boldsymbol{u}}^*$, located at (x^*, y^*), then leads to

$$\frac{\partial \underline{\boldsymbol{u}}}{\partial t} + \underline{\boldsymbol{u}} \cdot \nabla \underline{\boldsymbol{u}} \approx \frac{\alpha}{\Delta t_n} \underline{\boldsymbol{u}}^* + \left(\frac{\beta}{\Delta t_{n-1}} - \frac{\alpha}{\Delta t_n} \right) \underline{\boldsymbol{u}}_d^n - \frac{\beta}{\Delta t_{n-1}} \underline{\boldsymbol{u}}_d^{n-1}, \tag{4.7}$$

where $\alpha = \dfrac{2\Delta t_n + \Delta t_{n-1}}{\Delta t_n + \Delta t_{n-1}}$, $\beta = -\dfrac{\Delta t_n}{\Delta t_n + \Delta t_{n-1}}$, and $\underline{\boldsymbol{u}}_d^n$ and $\underline{\boldsymbol{u}}_d^{n-1}$ are the interpolations of the velocity field a time t_{n-1} and t_n respectively along the characteristic of equation (4.5) that passes through (x^*, y^*). We refer the reader to [2] for further details.

Implicit discretization of the Laplace operator

The information on the velocity field is located on the faces of the leaves of the mesh, and therefore that is where the Laplace operator in equation (4.3) must be discretized.

As explained in [2], this can be done by applying a finite volume approach to the Voronoi tessellation of the faces of the leaves. We present a summary of the approach and refer the reader to [2] for further details.

Given a set of points in space, called seeds, we define the Voronoi cell of a seed as the region of space that is closer to that seed than to any other seed. In the case of the x-component of the velocity field, located on the vertical faces of the mesh, the seeds correspond to the center of the faces of the leaves of the Quadtree grid. The union of all the Voronoi cells forms a tessellation of the domain, i.e. a non-overlapping gap-free tiling of the domain. And example of Voronoi tessellation of a Quadtree grid is presented in figure 4.2. The diffusion equation

$$\mu \Delta u = f \tag{4.8}$$

is then discretized on the Voronoi tessellation with a finite volume approach, where the control volume for each degree of freedom i is its Voronoi cell $\mathcal{C}_i$, as follows:

$$\int_{\mathcal{C}_i} \mu \Delta u = \int_{\partial \mathcal{C}_i} \mu \ \nabla u \cdot \underline{\boldsymbol{n}} \approx \sum_{j \in \text{ngbd}(i)} \mu \ s_{ij} \frac{u_j - u_i}{d_{ij}},$$

where ngbd(i) is the set of neighbors for i, $\underline{\boldsymbol{n}}$ is the normal to the face of $\mathcal{C}_i$ between i and j, s_{ij} is the length of that face and d_{ij} is the distance between i and j, as illustrated in figure 4.2. This discretization provides a second order accurate solution [144].

General discretization for the viscosity step

Combining the discretizations presented in the two previous sections, we obtain the general formula for any degree of freedom i and its associated Voronoi cell $\mathcal{C}_i$

$$\text{Vol}(\mathcal{C}_i)\rho \frac{\alpha}{\Delta t_n} u_i^* + \mu \sum_{j \in \text{ngbd}(i)} s_{ij} \frac{u_i^* - u_j^*}{d_{ij}} = \text{Vol}(\mathcal{C}_i)\rho \left[\left(\frac{\alpha}{\Delta t_n} + \frac{\beta}{\Delta t_{n-1}} \right) u_{i,d}^n - \frac{\beta}{\Delta t_{n-1}} u_{i,d}^{n-1} \right],$$

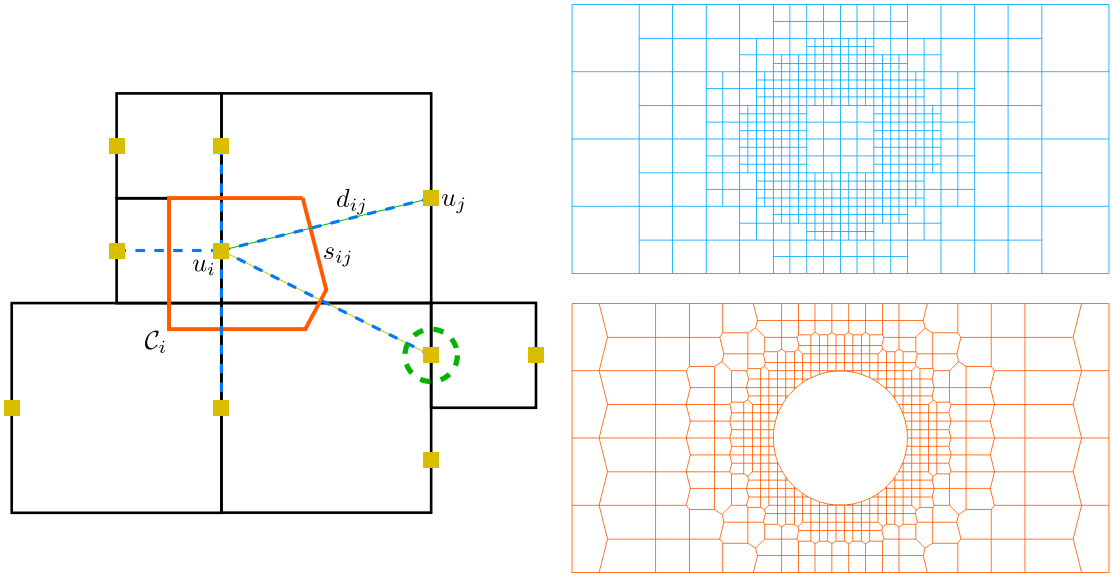


Figure 4.2: Left: nomenclature for the discretization of the Laplace operator on a Voronoi diagram. Right: example of a Quadtree mesh (top) and its Voronoi tessellation for the vertical faces (bottom).

with $\mathrm{Vol}(\mathcal{C}_i)$ the volume of $\mathcal{C}_i$. This produces a symmetric positive definite linear system that we solve using the biconjugate gradient stabilized iterative solver and the successive over-relaxation preconditioner provided by the PETSc library [75, 108, 109].

4.2.4 A stable projection

The projection step consists in solving the Poisson equation (4.4) with the data located at the center of the leaves of the tree. Stability and accuracy constraints result in the discretization presented in [49]. The method relies on a finite volume approach with a leaf being the control volumes for the degree of freedom located at its center. Using the notations defined in figure 4.3, we now explain the discretization of the flux of the Hodge variable Φ on the right face of $\mathcal{C}_2$. The first step is to define the average distance Δ between Φ_0 and its neighboring small leaves on the left side,

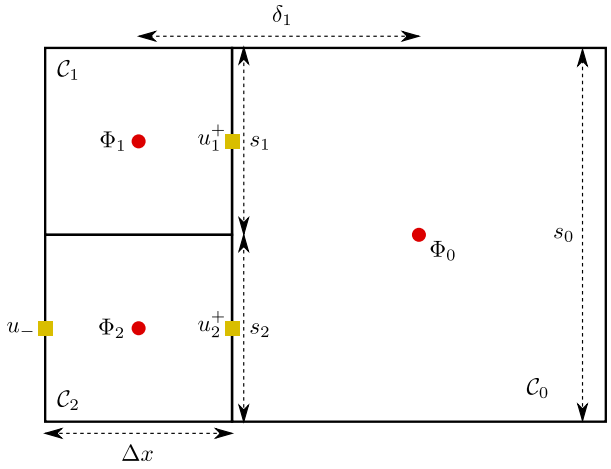


Figure 4.3: Nomencalture for the discretization of the Laplace operator with the data located at the center of the leaves.

$$\Delta = \sum_{i \in \mathcal{N}} \frac{s_i}{s_0} \delta_i,$$

where $\mathcal{N}$ is the set of leaves whose right neighbor leaf is $\mathcal{C}_0$. We then define the gradient of Φ on the right face of $\mathcal{C}_2$ as

$$\frac{\partial \Phi}{\partial x} = \sum_{i \in \mathcal{N}} \frac{s_i}{s_0} \frac{\Phi_0 - \Phi_i}{\Delta}.$$

Similarly, we define the divergence of u at the center of the leaf containing Φ_2 as

$$\nabla \cdot u = \frac{1}{\Delta x} \left(\sum_{i \in \mathcal{N}} \frac{s_i}{s_0} u_i^+ - u^- \right)$$

Both the divergence and the gradient operators involve the small leaves sharing a right neighbor leaf with $\mathcal{C}_0$. This discretization is second order accurate and ensures that the projection step is stable. The proofs of stability and second order accuracy can be found in [2]. The linear system resulting from this approach is symmetric positive definite, and we use the same combination of the biconjugate gradient stabilized iterative solver

preconditioned with successive over-relaxation provided by the Petsc library than for the viscosity step.

4.3 Parallel algorithms

In this section, we discuss the new parallel algorithms introduced in addition to the ones presented in [131]. Specifically, we describe how to extend the collection of off-processor neighbor octants, commonly called ghost cells, iteratively by any desired number of layers. In addition, we comment on the numbering scheme of face-centered variables.

4.3.1 Expansion of the ghost layer

The Voronoi and cell-centered finite volume operators constructed in sections 4.2.3 and 4.2.4 are compact, in that they only connect degrees of freedom located on the same or adjacent cells. However, in the case of the Voronoi based discretization, a neighboring degree of freedom can be on the edge of an adjacent cell and yet belong to a second-degree cell neighbor, as illustrated in figure 4.2. Furthermore, the backward difference formula in section 4.2.3 uses a wide neighborhood for the second order least-squares interpolation, as illustrated in figure 4.4. The locations for the upstream values $u_{i,d}^n$ and $u_{i,d}^{n-1}$ for a velocity degree of freedom u_i can also, depending on the time step, lie outside of the cells adjacent to u_i. For these reasons, the layer of ghost cells at inter-processor boundaries must be deeper than a single layer. The ability to construct deep ghost layers is a recent extension to `p4est` interface, which we briefly describe here.

The algorithm used by `p4est` to construct a single layer of ghosts ([44, Algorithm 19]) is able to maximize the overlap of computation and communication because each process can determine for itself which other processes are adjacent to it. This is because the

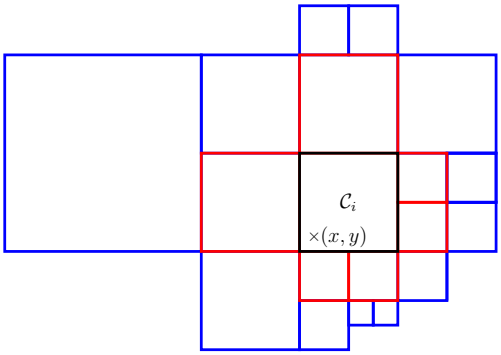


Figure 4.4: The stencil used to interpolate the velocity at (x, y) in cell $\mathcal{C}_i$ is not just $\text{ngbd}(\mathcal{C}_i)$ (red), but $\text{ngbd}^2(\mathcal{C}_i)$ (blue), a set of cells including second-degree (indirect) neighbors.

"shape" of each process's subdomain (determined by the interval of the space-filling curve assigned to it) is known to every other process. As a consequence, the communication pattern is symmetric and no sender-receiver handshake is required.

As a first extension, when creating the send buffers we remember their entries, since they identify the subset of local cells that are ghosts to one or more remote processes. We store these preimage cells or "mirrors" in ascending order with respect to the space filling curve, and create one separate index list per remote processor into this array. This data is accomodated inside the ghost layer data structure and proves useful for many purposes, the most common being the local processer needing to iterate through the preimage to define and fill send buffers with application-dependent numerical data.

The communication pattern of a deeper ghost layer, on the other hand, depends not just on the shapes of the subdomains, but the leaves within them, as illustrated in figure 4.5. Rather than complicating the existing ghost layer construction algorithm to accommodate deep ghost layers, a function that adds an additional layer to an existing ghost layer has been added to `p4est`. This function is called `p4est_ghost_expand`() and adds to both the ghosts and the preimages. Thus, as a second extension to the data structure, we also identify those local leaves that are on the inward-facing front of each

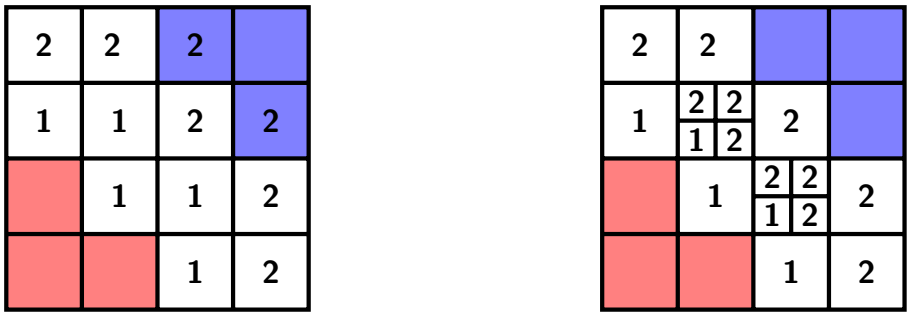


Figure 4.5: Two meshes with the same partition shapes, but with different two-deep ghost layers. In each we show the first and second layers of the ghost layer of process p (red). In the first, the second layer includes cells from process q (blue), but in the second it does not.

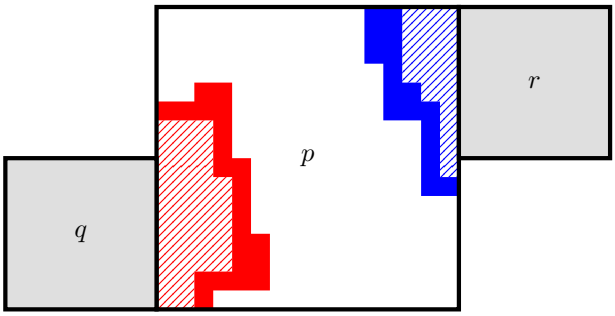


Figure 4.6: We show the preimages of process q's and process r's ghost layers in the leaves of process p. The solid red area represent the cells at the "front" of the preimage for q (`preimage_front`$[q]$ in algorithm 8), while the solid and dashed together form the whole preimage (`preimage`$[q]$).

preimage, in other words the most recently added mirrors. This is illustrated in figure 4.6.

When process p expands its portion of process q, it loops over the leaves in the front of the preimage for process q and adds any neighbors that are not already in the ghost layer. Sometimes this will include a leaf from a third process r: process p will also send such leaves to process q, because it may be that r is not yet represented in q's ghost layer, and so communication between q and r is not yet expected. The basic structure of this algorithm is outlined in algorithm 8.

```
1: for q ∈ ghost_neighbors do                    ▷ processes that contribute to ghost layer
2:     initialize empty sets send_forward[q], send_back[q], and new_front[q]
3: end for
4: for q ∈ ghost_neighbors do
5:     for l ∈ preimage_front[q] do
6:         for each neighbor n of l in local_leaves do                ▷ n found by search
7:             if n ∉ preimage[q] then
8:                 add n to send_forward[q], preimage[q], and new_front[q]
9:             end if
10:        end for
11:        for each neighbor n of l in ghost_layer do                 ▷ n found by search
12:            if n belongs to process r ≠ q then
13:                add n to send_forward[q] and (n, q) to send_back[r]
14:            end if
15:        end for
16:    end for
17:    replace preimage_front[q] with new_front[q]
18: end for
19: for q ∈ ghost_neighbors do
20:    send (send_forward[q], send_back[q]), receive (recv_forward[q], recv_back[q])
21:    add all of recv_forward[q] to ghost_layer
22:    for (l, r) ∈ recv_back[q] do
23:        if r ∉ ghost_neighbors or l ∉ preimage[r] then
24:            add l to preimage[r] and preimage_front[r]              ▷ new lists if
   r ∉ ghost_neighbors
25:        end if
26:    end for
27: end for
28: recompute ghost_neighbors from leaves in ghost_layer
```

Algorithm 8: Process p's algorithm for expanding other processes' ghost layers, and receiving expansions to its own ghost layer. Note that finding a neighbor of a leaf l entails a fixed number of binary searches through the `local_leaves`, which are sorted by the space-filling curve induced total ordering.

4.3.2 Indexing the faces

Although the `p4est` library provides a global numbering for the faces of the leaves, its numbering differs from our needs because it does not number the small faces on a coarse-fine interface, where we have degrees of freedom in our MAC scheme. Therefore, we implement a procedure to distribute the faces of the leaves across the processes and to generate a unique global index for each face. Since some faces are shared between two processes, we chose to attribute a shared face to the process with the smaller index. With this rule, each face belongs to a unique process and after broadcasting the local number of faces a global index can be generated for all the local faces. The second step is to update the remote index of the faces located in the ghost layer so that their global index can be constructed easily by simply adding the offset of the process each face belongs to. We do so in two steps, represented in figure 4.7. First, the indices of the ghost faces of the local leaves are synchronized, then the indices of the faces of the ghost layer of leaves are updated. This has some similarities to the two-pass node numbering from [68], here extended to two layers of ghosts. Algorithm 9 details the steps of our implementation and makes use of the `Notify` collective algorithm described in [70] to reverse the asymmetric communication pattern.

4.4 Scalability

In this section, we present an analysis of the scaling performance of our implementation. We define the parallel efficiency as $e = s \cdot (P_0/P)^\eta$ with $s = t_0/t_p$ the speed-up where η is the optimal parallel scaling coefficient ($\eta = 1$ for linear scaling), P_0 is the smaller number of processes with its associated runtime t_0 and P is the number of processes with its associated runtime t_p. All the results were obtained on the Stampede supercomputer at the Texas Advanced Computing Center (TACC), at The University of Texas

```
1: for l ∈ (local|ghost) leaves do
2:     for f ∈ remote_faces(l) do
3:         add proc(f) to receivers
4:         add f to buffer[proc(f)]
5:     end for
6: end for
7: Notify(receivers,senders)                    ▷ reverse communication pattern
8: for p ∈ receivers do                                          ▷ send requests
9:     MPI_Isend(buffer[p])                      ▷ send request to process p
10: end for
11: for p ∈ senders do                                  ▷ process remote requests
12:     MPI_Recv(req)                           ▷ receive request from process p
13:     assemble answer with local indices requested
14:     MPI_Isend(ans)                            ▷ send answer to process p
15: end for
16: for p ∈ receivers do                                        ▷ process answers
17:     MPI_Recv(p)                             ▷ receive answer from process p
18:     update faces information
19: end for
```

Algorithm 9: Communication algorithm to generate a global indexing of the faces. The `Notify` collective algorithm is used to reverse the communication pattern, described in more detail in [70].

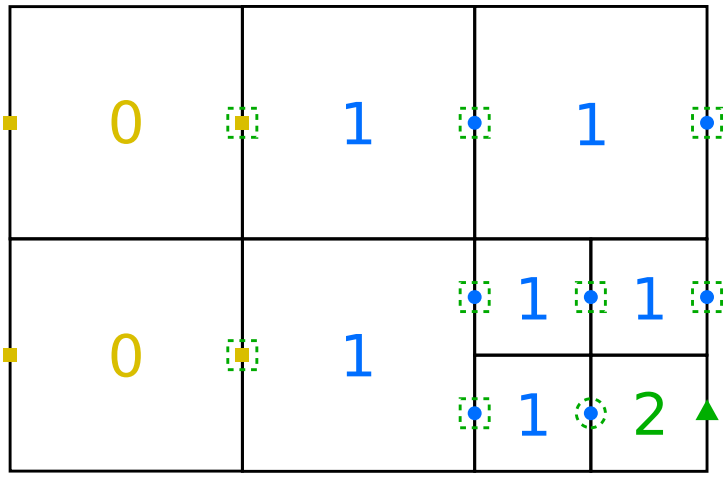


Figure 4.7: Illustration of the ghost layer of x-faces of depth 2 and of the global indexing procedure for process 2. The numbers in the leaves correspond to the indices of the processes owning them. After the first step, the remote index for the circled face is known to process 2, and after the second step the remote indices for the faces in a square are known to process 2. Note that a single step would not be sufficient for process 2 to gain knowledge of the remote index of the two faces belonging to process 0.

at Austin, and on the Comet supercomputer at the San Diego Supercomputer Center, at the University of California at San Diego. Those resources are available through the Extreme Science and Engineering Discovery Environment (XSEDE) [71]. The maximum number of processes available to us per run are 4,096 for Stampede and 1,728 for Comet.

4.4.1 Expansion of the ghost layer

We present both weak and strong scaling results for the algorithm use to expand the ghost layer of cells for each process in figure 4.8. The associated efficiency is presented in table 4.1. The strong scaling consists in choosing a problem and solving it with increasing number of processes. Ideally, for an algorithm with a workload increasing linearly with the problem size (i.e. with parallel scaling coefficient $\eta = 1$), doubling the amount of resources spent on solving a problem should half the runtime. However, in the case of the ghost layer expansion, and as explained in [46], the amount of work depends on the

size of the ghost layers. For a well behaved partition, we expect $O(N^{\frac{(d-1)}{d}})$ of the leaves to be in the ghost layer, where d is the number of spatial dimensions. We therefore consider a parallel scaling coefficient $\eta = 2/3$ to be optimal for a three dimensional problem, i.e. $O((N/P)^{2/3})$ is the ideal scaling, with P the number of processes and N the problem size. The results presented were obtained on Stampede for a mesh of level 9/13, corresponding to 588,548,472 leaves, and on Comet for a mesh of level 10/13, corresponding to 1,595,058,088 leaves. The computed parallel efficiency between the smallest and the largest run is 66% for Stampede and 59% for Comet.

The idea behind the weak scaling is to keep the problem size constant for each process while increasing the number of processes. The right graph of figure 4.9 presents the results obtained on Stampede for two problems of sizes 30,248 leaves per process and 473,768 leaves per process, and for a number of processes ranging from 27 to 4,096. The runtime increases by 16% between the smallest and the largest run for the small problem and by 6% for the large problem.

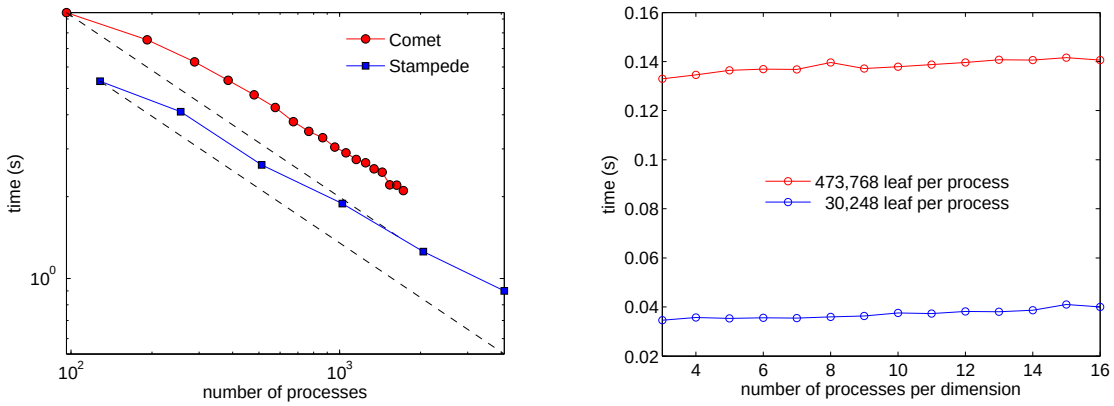


Figure 4.8: Scaling results for the expansion of the layer of ghost cells (see section 4.4.1). The strong scaling results are presented in the left figure together with the ideal reference scaling for a parallel scaling coefficient $\eta = 2/3$ (dashed lines) while the weak scaling results are shown on the right figure. The increases in runtime observed for the weak scaling are of 16% for the small problem and 6% for the large problem.

Stampede

Number of processes	128	256	512	1024	2048	4096
Efficiency	100%	79%	70%	69%	66%	66%

Comet

Number of processes	96	192	384	672	1152	1728
Efficiency	100%	82%	81%	71%	67%	59%

Table 4.1: Efficiency of the procedure for expanding the ghost layer of leaves.

4.4.2 Indexing the faces

The scaling procedure presented in the previous section is repeated for Algorithm 9 and the results are presented in figure 4.9. Even though the workload for this procedure increases slightly as the number of processes increases and the number of leaves in the ghost layers increases, we compare our results to and ideal linear scaling $\eta = 1$. The corresponding efficiency is computed in table 4.2. The parallel efficiency computed between the smallest and the largest run from the strong scaling results is 44% for Stampede and 70% for Comet. The weak scaling results show an increase in runtime of 71% for the small problem and of 14% for the large problem.

Stampede

Number of processes	128	256	512	1024	2048	4096
Efficiency	100%	94%	87%	76%	63%	44%

Comet

Number of processes	96	192	384	672	1152	1728
Efficiency	100%	96%	88%	82%	77%	70%

Table 4.2: Efficiency of Algorithm 9 producing a global index for the faces.

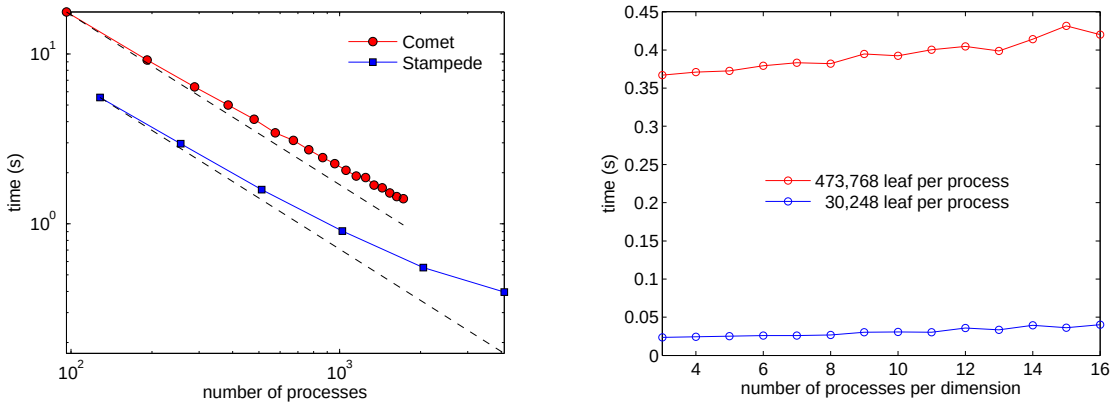


Figure 4.9: Scaling results for the indexing of the faces with Algorithm 9. The strong scaling results are presented in the left figure together with the reference ideal linear scaling (dash lines) while the weak scaling results are shown on the right figure. The strong scaling problem shown for Comet is three times larger than the one for Stampede. The increases in runtime observed for the weak scaling are of 71% for the small problem and 14% for the large problem.

4.4.3 Scalability of the full solver

Having presented the scaling results for the new algorithms introduced in this manuscript, we now focus on analyzing the performance of the main components of the proposed incompressible Navier-Stokes solver. The results are presented in figure 4.10 for the three principal parts of the solver and for the overall program. The first part is the viscosity solver introduced in section 4.2.3, the second part is the projection step described in section 4.2.4 and the third part is the re-meshing step. The problem used to obtain these results consists in solving the first two iterations of the flow past a sphere described in section 4.5.2 on a macromesh of size $8 \times 4 \times 4$ and for Octrees of level 6/9. This leads to approximately 35 million leaves. The results observed are very satisfying, particularly considering that for the largest run, with $4,096$ processes, each process has only around $8,500$ degrees of freedom. The parallel efficiency of the program is reported in table 4.3. Once more, the disparity between the two supercomputers can be partially

explained by the larger number of processes available per run on Stampede.

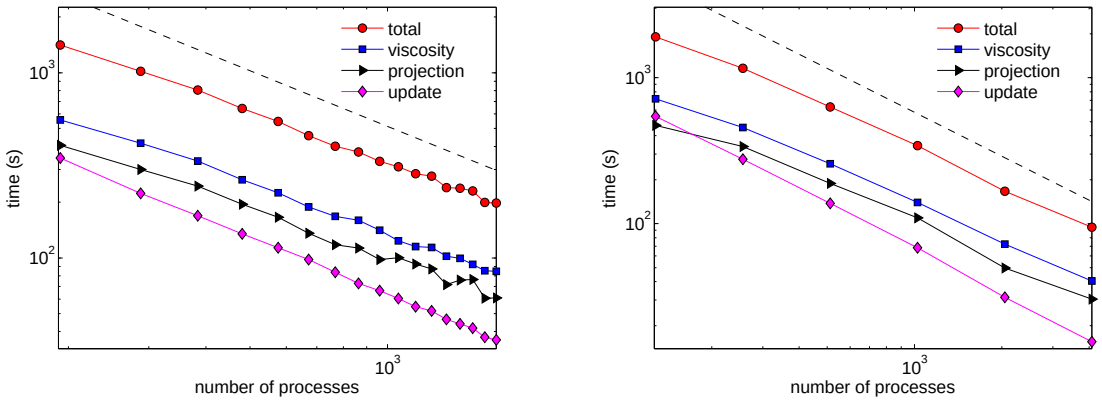


Figure 4.10: Scaling results for the main components of the complete solver and for the overall program on the Comet supercomputer (left) and on the Stampede supercomputer (right), compared with an ideal linear scaling (dashed lines).

Stampede

Number of processes	128	256	512	1024	2048	4096
Efficiency	100%	82%	75%	69%	71%	63%

Comet

Number of processes	192	384	576	864	1248	1728
Efficiency	100%	88%	87%	84%	79%	79%

Table 4.3: Efficiency of the full solver proposed for the incompressible Navier-Stokes equations on the Stampede and Comet supercomputers.

4.5 Numerical validation

In this section, we present a series of numerical examples to validate our implementation as well as to demonstrate the potential of our approach.

4.5.1 Validation with an analytical solution

Our first application aims at validating the implementation by monitoring the convergence of the solver using the analytical solution presented in [102]. Consider the irregular domain $\Omega = \{(x, y, z) | -\cos(x)\cos(y)\cos(z) \geq .4 \text{ and } \frac{\pi}{2} \leq x, y, z \leq \frac{3\pi}{2}\}$ and the exact solution

$$
\begin{aligned}
u(x, y, z) &= \cos(x)\sin(y)\sin(z)\cos(t), \\
v(x, y, z) &= \sin(x)\cos(y)\sin(z)\cos(t), \\
w(x, y, z) &= -2\sin(x)\sin(y)\cos(z)\cos(t), \\
p(x, y, z) &= 0.
\end{aligned}
$$

The exact velocity is prescribed at the interface and homogeneous Neumann boundary conditions are enforced for the Hodge variable. The appropriate forcing term is applied to the viscosity step. We take a final time of $\frac{\pi}{3}$ and monitor the error on the velocity field and on the Hodge variable as the mesh resolution increases. The successive resolutions are obtained by splitting every cell from the previous resolution. The results are presented in table 4.4 and indicate first order accuracy for the velocity field and second order accuracy for the Hodge variable in the L^∞ norm.

	u, v		w		Hodge variable	
level (min/max)	L^∞ error	order	L^∞ error	order	L^∞ error	order
4/6	$4.72 \cdot 10^{-3}$	-	$3.77 \cdot 10^{-3}$	-	$8.71 \cdot 10^{-4}$	-
5/7	$3.34 \cdot 10^{-3}$	0.50	$2.03 \cdot 10^{-3}$	0.89	$2.64 \cdot 10^{-4}$	1.72
6/8	$1.63 \cdot 10^{-3}$	1.03	$1.05 \cdot 10^{-3}$	0.95	$8.33 \cdot 10^{-5}$	1.66
7/9	$8.33 \cdot 10^{-4}$	0.97	$5.07 \cdot 10^{-4}$	1.05	$2.37 \cdot 10^{-5}$	1.81

Table 4.4: Convergence of the solver for the analytic solution presented in section 4.5.1. First order accuracy is observed for the velocity field and second order accuracy for the Hodge variable.

4.5.2 Vortex shedding of the flow past a sphere

Now that the accuracy of our solver have been validated on an analytical solution, we propose to compare its results to the standard problem of measuring the properties of a flow past a sphere. We consider a sphere of radius $r = 1$ and located at $(8, 0, 0)$ in the domain $\Omega = [0, 32] \times [-8, 8] \times [-8, 8]$. We impose an inflow velocity $u_0 = (1, 0, 0)$ on the $x = 0$ edge of the domain as well as on the the side walls, homogeneous Neumann boundary conditions on the velocity field at the outlet $x = 32$ and no-slip conditions on the sphere. The pressure is set to zero at the outlet and is subject to homogeneous Neumann boundary condition on the other walls as well as on the sphere. We set the density of the fluid to $\rho = 1$ and vary the viscosity μ according to the Reynolds number. The Octree mesh is refined around the sphere and according to the vorticity criteria of section 4.2.1, with a threshold of $\gamma = 0.01$. All the results were obtained with a macromesh $8 \times 4 \times 4$ and with trees of levels 4/7, leading to approximately 6 million leaves and corresponding to a uniform grid resolution of 268,435,456 cells. We set $\Delta t = 5\frac{\Delta x_{min}}{\max_\Omega \|\underline{\boldsymbol{u}}\|}$. We monitor the average drag coefficient C_D of the fully developed flow on the sphere, as well as the lift coefficient and the Strouhal frequency $St = \frac{2rf}{u_0}$ when applicable and when reference data is available for comparison. The drag coefficient is obtained by geometric integration [24] of the viscous and pressure forces on the sphere,

$$C_D = \frac{F_D}{\frac{1}{2}\rho u_0^2 \pi r^2} = \frac{\int_\Gamma (-p + 2\mu \underline{\underline{D}})\underline{\boldsymbol{n}}}{\frac{1}{2}\rho u_0^2 \pi r^2}, \tag{4.9}$$

where Γ is the surface of the sphere, $\underline{\underline{D}}$ is the symmetric stress tensor and $\underline{\boldsymbol{n}}$ is the outward normal to the sphere. The comparison of our results for a range of Reynolds numbers from 50 to 500 with the data from various publications is presented in tables 4.5 and 4.6 and drag coefficients are visualized in figure 4.11. Figure 4.12 shows a snapshot of the unsteady flow for $Re = 300$. The values we obtain are in general in very good agreement

with the existing literature.

	Re=50	Re=100	Re=150	Re=215	Re=250	
	C_D	C_D	C_D	C_D	C_D	C_L
Kim *et al.* [145]	-	1.09	-	-	0.70	0.059
Johnson *et al.* [129]	1.57	1.08	0.90	-	0.70	0.062
Constantinescu *et al.* [146]	-	-	-	-	0.70	0.062
Choi *et al.* [147]	-	1.09	-	-	0.70	0.052
Bagchi *et al.* [130]	1.57	1.09	-	-	0.70	-
Marella *et al.* [127]	1.56	1.06	0.85	0.70	-	-
Guittet *et al.* [2]	-	1.11	-	-	0.784	-
Present	1.63	1.13	0.92	0.78	0.74	0.063

Table 4.5: Drag coefficient on a sphere for steady flows.

	Re=300			Re=350		Re=500	
	C_D	C_L	St	C_D	St	C_D	St
Kim *et al.* [145]	0.657	0.067	0.134	-	-	-	-
Johnson *et al.* [129]	0.656	0.069	0.137	-	-	-	-
Constantinescu *et al.* [146]	0.655	0.065	0.136	-	-	-	-
Choi *et al.* [147]	0.658	0.068	0.134	-	-	-	-
Marella *et al.* [127]	0.621	-	0.133	-	-	-	-
Bagchi *et al.* [130]	-	-	-	0.62	0.135	0.56	0.175
Mittal *et al.* [126]	-	-	-	-	0.142	-	-
Guittet *et al.* [2]	0.659	-	0.137	0.627	0.141	-	-
Present	0.695	0.068	0.134	0.66	0.128	0.59	0.153

Table 4.6: Drag coefficient and Strouhal frequency for a sphere in a unsteady flow.

4.5.3 Oscillating sphere in a viscous fluid

As stated in [2], the solver presented in this article is able to handle deforming and moving geometries. We propose to illustrate this capacity by computing the drag coefficient of a viscous fluid on an oscillating sphere. We consider a sphere of radius $r = 0.1$ in a domain $\Omega = [-1, 1]^3$. We enforce no-slip boundary conditions on the sphere and

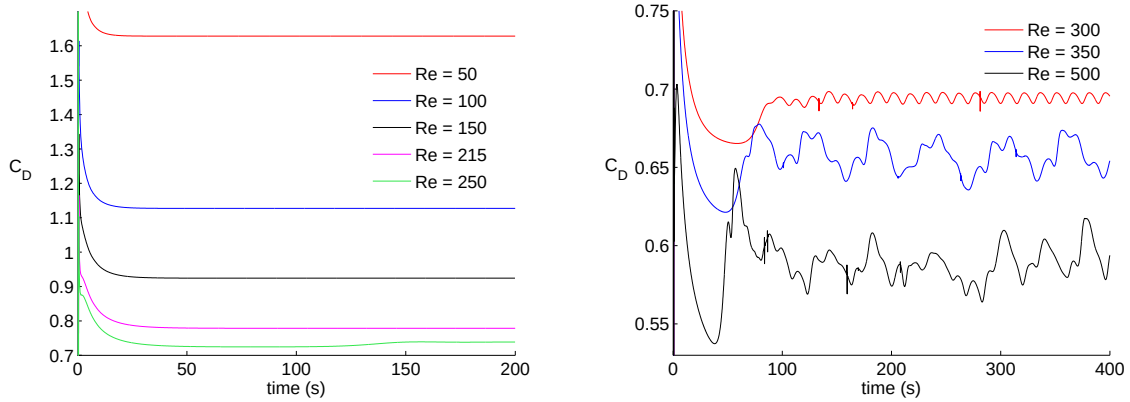


Figure 4.11: Visualization of the drag coefficient on a sphere for steady (left) and unsteady (right) regimes corresponding to a Reynolds number ranging from 50 to 500.

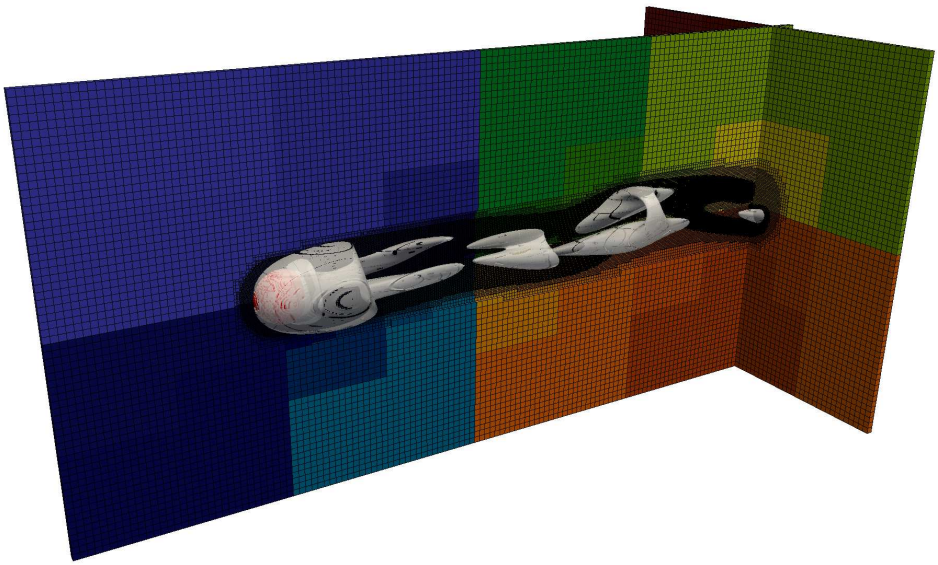


Figure 4.12: Visualization of the unsteady flow past a sphere for $Re = 300$. The trees are level 4/7 rooted in a 8x4x4 macromesh, leading to approximately 6 million leaves. The snapshot is taken at time $t = 221$ seconds, corresponding to 1600 time iterations. The colors correspond to the process ranks and the surface is an isocontour of the Q-criterion for $Q = 0.006$. This simulation was ran on the Stampede supercomputer with 1024 processes.

on the edges of the domain, and we impose an oscillatory motion to the sphere in the x-direction described by

$$x(t) = X_0 \cos(2\pi f_0 t),$$

where f_0 and $X_0 = r/4$ are the frequency and the amplitude of the oscillation, respectively. We choose $\mu = 1$, $\rho = 1$, and we set the Strouhal frequency to $S_t = \frac{2rf_0}{u_0} = 1.5$ and define the Reynolds number as $Re = \frac{2r\rho u_0}{\mu}$. For this example, we choose the fixed time step $\Delta t = \frac{T}{200} = \frac{1}{200 f_0}$. We then computed the drag coefficient on the sphere according to equation (4.9) for various Reynolds numbers. The results are presented in figure 4.13 for three periods of oscillation. As expected, we observe that the amplitude of the drag coefficient increases as the Reynolds number decreases. Furthermore, we observe a lag in the response as the Reynolds number decreases. Indeed, as the viscous forces become more important, the information takes longer to propagate in the fluid. In contrast, the forces in a system dominated by inertia come mainly from the pressure term and the incompressibility condition enforces instantaneous propagation of the information. Figure 4.14 shows some visual representations of the simulation for $Re = 80$. The simulations completed in around 4 hours on the Stampede supercomputer and with 512 processes for a mesh of resolution 5/10 rooted in a single macromesh cell, resulting in a number of leaves ranging between 2 and 5 million.

4.5.4 Transport of a scalar quantity in a flow

We now propose to add a smoke marker to the fluid. For the two examples in this section, we refine the Octree where any of the three criteria (interface, vorticity and marker) presented in section 4.2.1 is satisfied. For the first smoke simulation, the marker is passively advected in a flow past a sphere. The simulation setup is the same as the one in section 4.5.2 and we set the Reynolds number to $Re = 5,000$ and the level of each tree

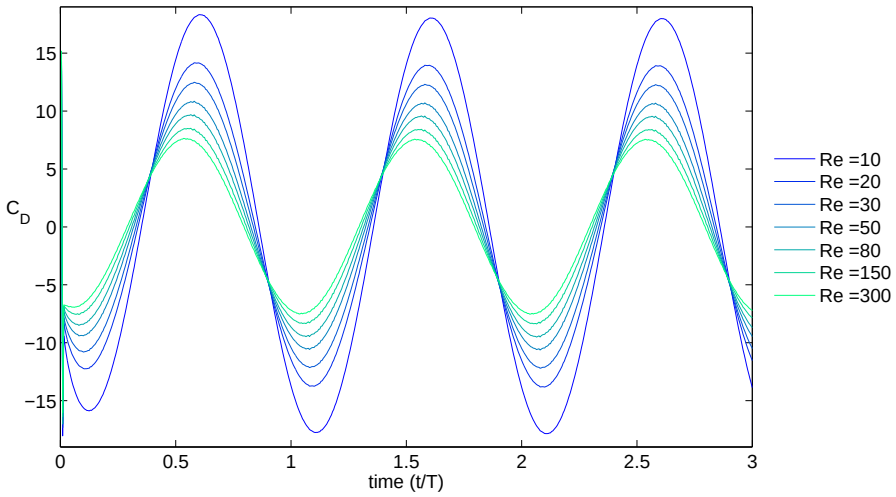


Figure 4.13: Drag coefficient for a periodically oscillating sphere and for a range of Reynolds numbers. The drag increases and the peaks shift as the Reynolds number decreases and the viscous forces become dominant.

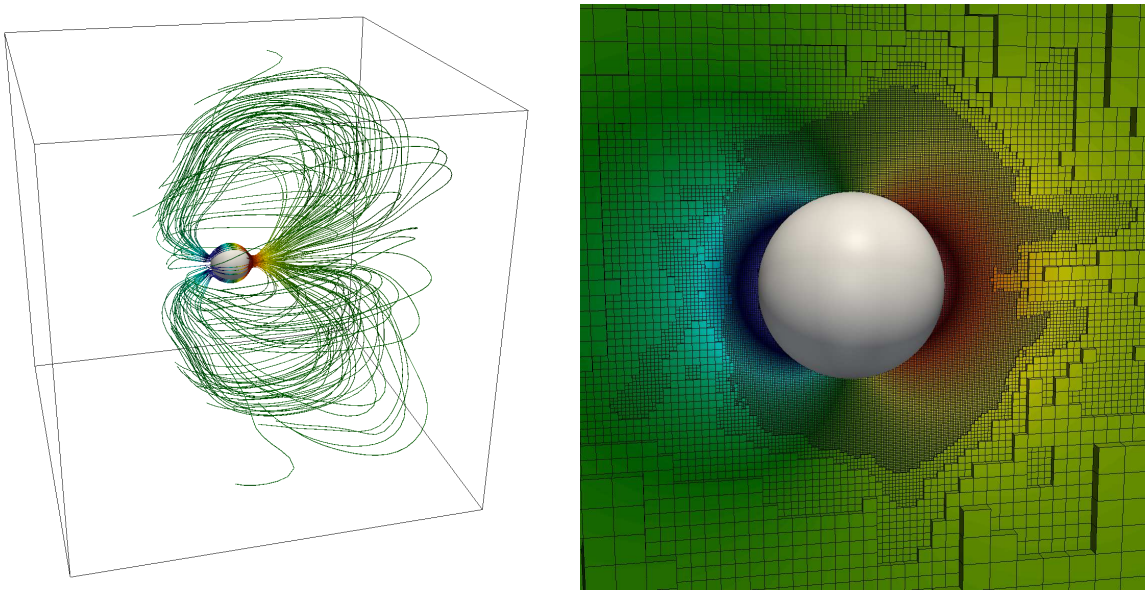


Figure 4.14: Left: visualization of the computational domain and of the streamlines colored with the pressure for the oscillating sphere 4.5.3. Right: slice of the Octree grid colored with the pressure.

to 2/8, with an $8 \times 4 \times 4$ macromesh. This leads to around 34 million leaves for a fully developed flow and with the marker refinement threshold set to $\delta = 0.05$. Figure 4.15 presents a visualization after 48 seconds, corresponding to 857 time steps. The entire simulation took 20 hours on 1,024 cores of the Stampede supercomputer.

The second example with a smoke marker involves gravity. Given a fluid of density $\rho = 1$

Figure 4.15: Visualization of a passively advected smoke in a flow past a sphere and for a reynolds number $Re = 5,000$.

in a box of dimension $\Omega = [0, 1]^3$, we initialize the simulation with a ball of smoke with concentration $\beta = 1$ located at $(0.5, 0.5, 0.75)$ and with radius 0.1. The action of gravity is then included by simply adding the force term $\beta(t, \underline{\boldsymbol{x}})\underline{\boldsymbol{g}}$ to the momentum equation 4.1. We choose a $2 \times 2 \times 2$ macromesh and trees with resolution 4/8, leading to 20 million leaves. The simulation took 20 hours on 256 cores of the Stampede supercomputer for 603 time iterations. Figure 4.16 shows three snapshots of the simulation.

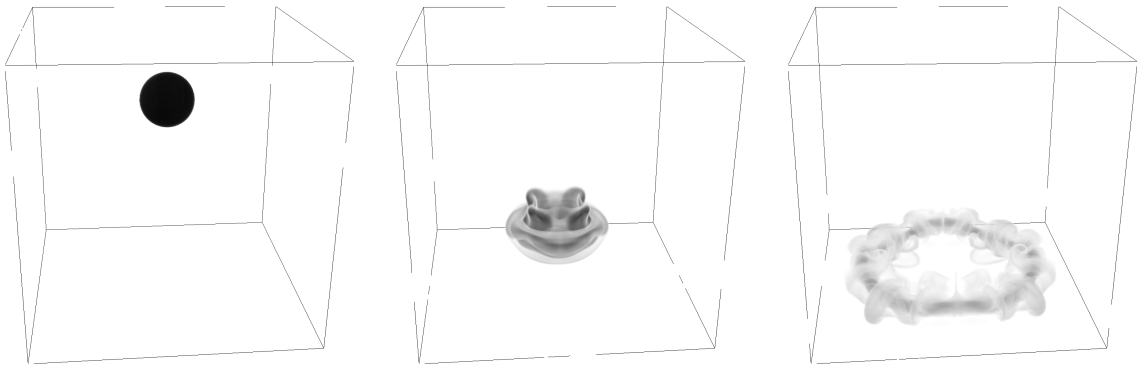


Figure 4.16: Visualization of a drop of high density smoke falling under gravity after 0 second, 0.75 second and 1.55 second.

4.6 Summary

We have described the implementation of a Navier-Stokes solver for simulating incompressible flows in irregular domains. The strategy is based on the discretizations on adaptive Cartesian grids using a forest of Octrees. We have introduced an algorithm for defining a unique indexing for the degrees of freedom located at the cells' faces in a standard MAC arrangement so that distributed machines can be readily considered. Scaling analyses have illustrated the strong and weak scalings of this solver. Future work will explore the extension to free surface and multiphase flows in irregular geometries.

Chapter 5

The Voronoi Interface Method for discontinuous elliptic problems

5.1 Introduction

We focus on the class of Elliptic problems that can be written as:

$$\begin{aligned} \underline{\boldsymbol{\nabla}} \cdot (\beta \underline{\boldsymbol{\nabla}} u) + ku &= f \qquad \text{in } \Omega^- \cup \Omega^+, \\ [u] &= g \qquad \text{on } \Gamma, \\ [\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma] &= h \qquad \text{on } \Gamma, \end{aligned} \tag{5.1}$$

where the computational domain Ω is composed of two subdomains, Ω^- and Ω^+, separated by a co-dimension one interface Γ (see figure 5.1), with $\underline{\boldsymbol{n}}_\Gamma$ the outward normal. Here, $\beta = \beta(\underline{\boldsymbol{x}})$, with $\underline{\boldsymbol{x}} \in \mathbb{R}^n$ ($n \in \mathbb{N}$), is bounded from below by a positive constant and $[q] = q_\Gamma^+ - q_\Gamma^-$ indicates a discontinuity in the quantity q across Γ, f is in L^2, g, h and k are given. Note that this general formulation includes possible discontinuities in the diffusion coefficient β and in the gradient of the solution $\underline{\boldsymbol{\nabla}} u$. Dirichlet or Neumann boundary conditions are applied on the boundary of Ω, denoted by $\partial\Omega$. This class of equations, where some or all of the jump conditions are non-zero, is a corner stone in the modeling

of the dynamics of important physical and biological phenomena as diverse as multi-phase flows with and without phase change, biomolecules' electrostatics, electrokinetics (Poisson-Nerntz-Planck) models with source term or electroporation models.

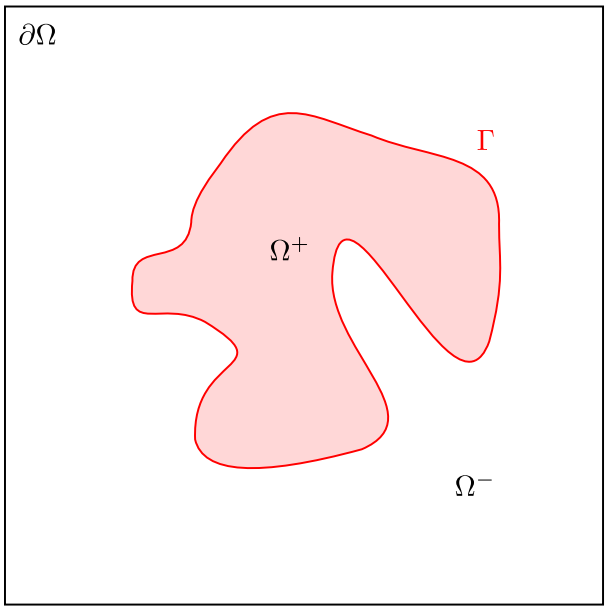


Figure 5.1: Geometry of the problem.

Given the importance of this class of equations, several approaches have been pursued to computationally approximate their solutions, each with their own pros and cons. The finite element method (FEM) is one of the earliest approaches to solve this problem [148, 149, 150, 151, 152, 153] and has the advantage of providing a simple discretization formalism that guarantees the symmetry and definite positiveness of the corresponding linear system, even in the case of unstructured grids. It also provides a framework where *a priori* error estimates can be used to best adapt the mesh in order to capture small scale details. However, the FEM is based on the generation of meshes that must conform to the irregular domain's boundary and must satisfy some restrictive quality criteria, a task that is difficult, especially in three spatial dimensions. The difficulty is exacerbated when the domain's boundary evolve during the course of a computation, as it is the case

for most of the applications modeled by these equations. Mesh generation is the focus of intense research [154], as the creation of unwanted sliver elements can deteriorate the accuracy of the solution.

Methods based on *capturing* the jump conditions do not depend on the generation of a mesh that conforms to the domain's boundary, hence avoiding the mesh generation difficulty altogether. However, they must impose the boundary conditions implicitly, which is a non-trivial task. A popular approach is the Immersed Interface Method (IIM) of Leveque and Li [135], and the more recent development of Immersed Finite Element Method (IFEM) and Immersed Finite Volume Method (IFVM) [155, 156, 157]. The basis of the IIM is to use Taylor expansions of the solution on each side of the interface and modify the stencils local to the interface in order to impose the jump conditions. As such, solutions can be obtained on simple Cartesian grids and the solution is second-order accurate in the L^∞ norm. The corresponding linear system, however, is asymmetric unless the coefficient β has no jump across the interface. Another difficulty is the need to approximate surface derivatives along Γ as well as the evaluation of high-order jump conditions. These difficulties have been addressed in the Piecewise-polynomial Interface Method of Chen and Strain [158] and several other approaches have improved the efficacy of the IMM [159, 160, 161, 162, 163, 164, 165]. We note also that the earliest approach on Cartesian grid is that of Mayo [166], who derived an integral equation to solve the Poisson and the bi-harmonic problems with piecewise constant coefficients on irregular domains; the solution is second-order accurate in the L^∞ norm. We also refer the interested researcher to the matched interface and boundary (MIB) method [167, 168].

The finite element community has also proposed embedded interface approaches, including discontinuous Galerkin and the eXtended Finite Element Method (XFEM) [169, 170, 171, 172, 173, 174, 175, 176, 177, 178]. The basic idea is to introduce additional

degrees of freedom[1] near the interface and augment the standard basis functions on these elements with basis functions that are combined with a Heaviside function in order to help capture the jump conditions.

In [179], the authors introduce a second-order accurate discretization in the case of, possibly adaptive, Cartesian meshes using a cut-cell approach. The jump condition is imposed by determining the fluxes on both side of the interface, which are constructed from a combination of least squares and quadratic approximations. In [180], the authors also use a cut cell approach but impose the jump with the help of a compact 27-point stencil.

The Ghost Fluid Method (GFM), originally introduced to approximate two-phase compressible flows [181], has been applied to the system problem 5.1 in [182]. The basic idea is to consider fictitious domains and ghost values that capture the jump conditions in the discretization at grid nodes near the interface. An advantage of this approach is that only the right-hand-side of the linear system is affected by the jump conditions. However, in order to propose a dimension-by-dimension approach, the projection of the normal jump conditions must be projected onto the Cartesian directions. As a consequence, the tangential component of the jump is ignored. Nonetheless, the method has been shown to be convergent with first-order accuracy [183]. The GFM was also shown to produce symmetric positive definite second-order accuracy [104] and even fourth-order accuracy [111], but for a different class of problem, namely for solving Elliptic problems on irregular domains with Dirichlet boundary conditions. In fact, symmetric positive definite second-order accurate solutions can also be obtained in the case where Neumann or Robin boundary conditions are imposed on irregular domains [77, 102, 184, 185]. These methods can be trivially extended to the case of adaptive Cartesian grids and we refer the interested readers to the review of Gibou, Min and Fedkiw [186] for more

[1]We understand by degrees of freedom the set of locations at which the solution is sampled.

details. In the case of jump conditions, Coco and Russo [187] have also used a fictitious domain approach, where a relaxation scheme is used to impose the boundary condition; the solution is second-order accurate. The same authors have also introduced a method to consider Dirichlet, Neumann and Robin boundary conditions on an irregular interface [188]. Latige *et al.* [189] have also presented a method based on fictitious domains using a piecewise polynomial representation of the solution on a dual grid, also obtaining second-order accurate solutions. Finally, [190] have applied the ghost fluid idea in a variational framework.

Related ideas are used in methods combining fictitious domains and variational formulations [191, 192], dubbed virtual nodes approaches. Some of these approaches can be considered similar to XFEM methods [193, 194, 195], while others are different and offer advantages when considering under-resolved, possibly non-smooth, interfaces [191, 196, 197]. This philosophy has been used in [198], which introduces a virtual node algorithm for solving Elliptic problems on irregular geometries with jump conditions as well as Dirichlet or Neumann boundary conditions imposed on Γ. The solutions are second-order accurate in the L^∞ norm and the approach provides a unifying treatment for Dirichlet, Neumann and jump boundary conditions; however sacrificing simplicity.

Rather recently, Cisternino and Weynans [199] introduced a second-order accurate method that uses additional degrees of freedom on the domain's boundary and use them to discretize the Poisson operator with jump conditions in a dimension-by-dimension framework. The authors also present how to carefully approximate the gradients. The method produces second-order accurate solutions and a nonsymmetric linear system in part because of the need to change the size of the stencil for nodes adjacent to the domain's boundary.

We introduce a capturing computational approach, the Voronoi Interface Method, that produces second-order accurate solutions in the L^∞ norm. This approach is based on

building a Voronoi diagram local to the interface, which enables the direct discretization of the jump conditions in the normal direction. The linear system is symmetric positive definite and the jump conditions only influence the right-hand-side. The construction of the local Voronoi mesh is a straightforward and parallelizable process and can be built with existing libraries that are freely available. In the present work, we use the excellent `Voro++` library in three spatial dimensions [107]. This method is different from body-fitted methods in that it relies on the post-processing of an existing background mesh, thus avoiding the standard difficulties associated with body-fitted approaches. We note that previous works have developed solvers for the Poisson equation on Voronoi diagrams (see [200, 201] and the references therein); however discontinuities across an irregular interface were not considered.

5.2 The geometrical tools

5.2.1 The level-set method

The level-set method [4] is a powerful way of representing irregular interfaces as the zero contour of a continuous function. This representation is convenient in that it can be applied to the case of moving boundaries that can change their topology. It also provides a framework that lends itself to design *sharp* discretizations.

We use the level-set set framework to capture the irregular interface on which the discontinuities are enforced. We define a level-set function ϕ on the domain Ω such that the irregular interface Γ is described by $\Gamma = \{\underline{\boldsymbol{x}} \in \mathbb{R}^n \in \Omega \,|\, \phi(\underline{\boldsymbol{x}}) = 0\}$ and ϕ is negative on one side of the interface and positive on the other side, as pictured in figure 5.1. Even though infinitely many functions satisfy this criteria, it is convenient to work with a signed distance function to the irregular interface, i.e. a function that is negative on one

side of the interface, positive on the other side and such that its magnitude at every point is the distance from the point to the interface. Constructing a signed distance function from an arbitrary function can be done for example by following the procedure explained in [98]. The normal to the interface is then obtained as

$$\underline{\boldsymbol{n}} = \frac{\underline{\boldsymbol{\nabla}}\phi}{\|\underline{\boldsymbol{\nabla}}\phi\|},$$

and the curvature as

$$\kappa = \underline{\boldsymbol{\nabla}} \cdot \underline{\boldsymbol{n}}.$$

Note that if the level-set function is a signed distance function, $\|\underline{\boldsymbol{\nabla}}\phi\| = 1$, and the projection onto Γ of any given point $\underline{\boldsymbol{x}}$ is easily computed as:

$$\underline{\boldsymbol{x}}_\Gamma = \underline{\boldsymbol{x}} - \phi(\underline{\boldsymbol{x}})\underline{\boldsymbol{\nabla}}\phi(\underline{\boldsymbol{x}}). \tag{5.2}$$

5.2.2 Voronoi diagrams

The solver we present is based on Voronoi diagrams, which can be generated locally with existing procedures and freely available libraries. In this work, we use the excellent `Voro++` library [107]. For the sake of clarity, we introduce the Voronoi diagram: given a set of points, which we call seeds, the Voronoi cell of a given seed consists of all the points of the domain that are closer to that seed than to any other seed. Hence, the collection of all the Voronoi cells of a set of seeds is a tessellation of the domain, i.e. a tiling that fills the domain and does not contain any overlaps.

Given a computational mesh, which we consider to be uniform in this section for clarity, we propose to modify the mesh so that the irregular interface coincides with the

edges of the new mesh and the degrees of freedom close to the interface are all located at the same distance from the interface.

The procedure is illustrated in figure 5.2. Starting from a uniform grid, we find the projection of the degrees of freedom whose control volume is crossed by the irregular interface onto the interface using (5.2), and we remove those degrees of freedom from the original list of unknowns. If a projected point is within `diag`/5 of a previously computed projected point, where `diag` is the length of the diagonal of the smallest grid cell, we skip this points. Otherwise, we add two new degrees of freedom located at a distance d_Γ of the interface in the normal direction on either side of the interface. We repeat this procedure for all projected points. The new set of degrees of freedom is therefore made up of the original degrees of freedom whose control volume is not crossed by the interface and of the new degrees of freedom added next to the interface. This constitutes the set of seeds for the Voronoi diagram computational mesh on which we perform the computations. Each Voronoi cell can then be generated independently based on the local neighborhood of each degree of freedom, making the generation of the Voronoi mesh embarrassingly parallel.

Note that all the new degrees of freedom are placed at the same distance d_Γ from the interface. This is a free parameter of our method, and experimenting with various reasonable values shows little impact on the numerical results. We choose $d_\Gamma = \texttt{diag}/5$. This simple procedure will be shown in sections 5.4.1 and 5.4.2 to be sufficient to construct second-order accurate solutions in the L^∞ norm.

5.2.3 Smoothing the mesh

The algorithm described in the previous section can lead to undesirable geometrical configurations in the case when the interface is not sufficiently resolved. Figure 5.3

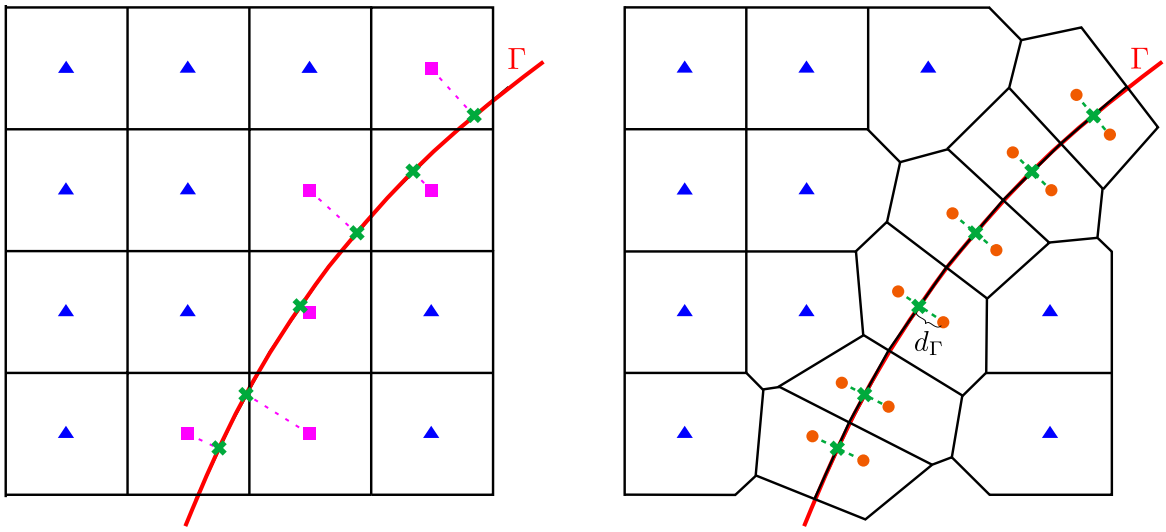


Figure 5.2: Illustration of the procedure for generating a Voronoi diagram based computational mesh. The left figure presents the original uniform mesh and the right figure shows the final computational mesh. The purple square degrees of freedom have been removed and the orange dots degrees of freedom have been added close to the interface Γ.

presents one such configuration. The control volumes of some of the degrees of freedom are connected by a face that is not capturing properly the interface.

It is possible to remediate this issue by modifying the Voronoi partition in a post-processing step. The control volume of any degree of freedom that has been added next to the interface should be connected to exactly one control volume associated to a degree of freedom on the other side of the irregular interface. Consequently, if more than one neighbor is found across the interface, we disconnect the undesired ones by removing the connecting edge as shown in figure 5.3. The edge and its two associated vertices are then replaced by a single vertex located in the middle of the removed edge. The control volume of all the degrees of freedom of the resulting mesh are connected to at most one control volume on the other side of the interface and the interface is captured properly.

However, this procedure alters the mesh which is no longer a Voronoi diagram, and

the edge between two degrees of freedom is not guaranteed to be orthogonal to the line connecting the two degrees of freedom. The impact of this post-processing algorithm is analyzed in sections 5.4.1 and 5.4.2 and does not seem to improve the method, we therefore recommend not using it.

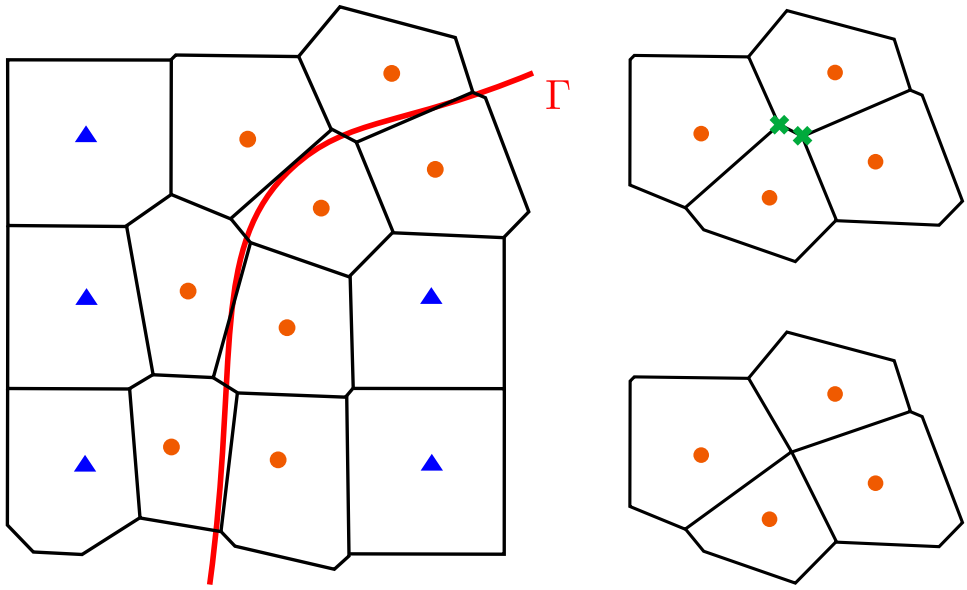


Figure 5.3: Example of a Voronoi mesh where a degree of freedom is connected to more than one other degree of freedom located on the other side of the irregular interface. The smoothing procedure is illustrated on the right.

5.2.4 Interpolating back to the original mesh

In general, if solving a diffusion equation with discontinuities is part of a larger solver, it is necessary to interpolate the solution from the Voronoi mesh back to the original mesh. This is an easy task given some basic bookkeeping information linking the original degrees of freedom to the ones generated for the Voronoi mesh. With this information, the solution on the Voronoi mesh can be accessed for the same cost than accessing the data on the original mesh. The algorithm to interpolate the solution at a given point

(x, y) from the Voronoi mesh is then as follows:

1. locate the cell of the original mesh containing (x, y),
2. using the bookkeeping information, identify the Voronoi degree of freedom $v(x, y)$ closest to (x, y),
3. find the two neighbors of $v(x, y)$ closest to (x, y) and on the same side of the interface,
4. compute the multilinear interpolation of the solution using those three degrees of freedom and evaluate it at (x, y).

This simple procedure produces a second-order interpolation at any given point (x, y).

5.3 Solving a Poisson equation on Voronoi diagrams

We discretize equation (5.1) with a finite volume approach on the Voronoi diagram introduced in section 5.2.2. We use the notations from figure 5.4. We consider the degree of freedom i with the set of Voronoi neighbors $\{j\}$. Applying a finite volume approach to the problem at i, we can write

$$\begin{aligned}\int_{\mathcal{C}} \underline{\boldsymbol{\nabla}} \cdot (\beta \underline{\boldsymbol{\nabla}} u)\, dV &= \int_{\partial \mathcal{C}} (\beta \underline{\boldsymbol{\nabla}} u) \cdot \underline{\boldsymbol{n}}_{\mathcal{C}}\, dl \\ &\approx \sum_j s_{ij} \beta_i \frac{u_{ij} - u_i}{d_{ij}/2},\end{aligned}$$

where $\underline{\boldsymbol{n}}_{\mathcal{C}}$ is the outer normal to the face of $\mathcal{C}$ connecting i and j, s_{ij} is the length of that face (or area of the surface in three spatial dimensions) and d_{ij} is the distance between the degrees of freedom i and j. For the case when i and j are on either side of the

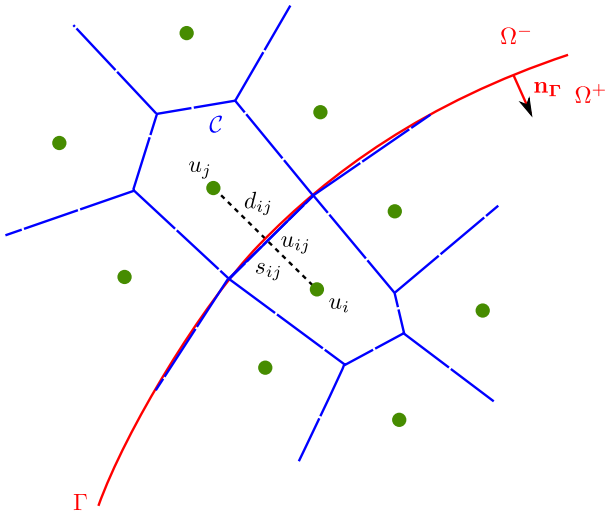


Figure 5.4: Nomenclature for the finite volume disretization for the degree of freedom i. For each neighboring degree of freedom j, we call d_{ij} the distance between i and j, s_{ij} the length of the edge (or surface of the polygon in three spatial dimensions) connecting i and j, and u_{ij} the value of u at the middle of the segment $[i, j]$. Note that by construction u_{ij} can be considered to be exactly on the irregular interface Γ, in which case we define u_{ij}^+ and u_{ij}^-.

interface with $\phi_i > 0$, where ϕ_i is the value of the level-set function at the degree of freedom i, we can match the flux at the irregular interface as follows,

$$s_{ij}\beta_i \frac{u_{ij}^+ - u_i}{d_{ij}/2} = s_{ij}\beta_j \frac{u_j - u_{ij}^-}{d_{ij}/2} - s_{ij}\left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma\right],$$

We also know that $u_{ij}^+ = u_{ij}^- + [u]$. Injecting this into the previous expression gives

$$\begin{aligned}
& s_{ij}\beta_i \frac{u_{ij}^+ - u_i}{d_{ij}/2} = s_{ij}\beta_j \frac{u_j - u_{ij}^+ + [u]}{d_{ij}/2} - s_{ij}\left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma\right] \\
\Leftrightarrow \quad & u_{ij}^+(\beta_i + \beta_j) = \beta_j u_j + \beta_i u_i + \beta_j [u] - \frac{d_{ij}}{2}\left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma\right] \\
\Leftrightarrow \quad & u_{ij}^+ = \frac{1}{\beta_i + \beta_j}\left(\beta_j u_j + \beta_i u_i + \beta_j [u] - \frac{d_{ij}}{2}\left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma\right]\right).
\end{aligned}$$

In the case when i and j are on the same side of the interface, the derivation is the same but the contributions from the discontinuities vanish. The contribution from the interaction between the degrees of freedom i and j, in the case when $\phi_i > 0$, to the finite volume discretization of (5.1) can then be written

$$\begin{aligned}
\beta_i s_{ij} \frac{u_{ij}^+ - u_i}{d_{ij}/2} &= \frac{2\beta_i}{\beta_i + \beta_j} \frac{s_{ij}}{d_{ij}} \left(\beta_j u_j + \beta_i u_i - (\beta_i + \beta_j) u_i + \beta_j [u] - \frac{d_{ij}}{2} \left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma \right] \right) \\
&= \frac{2\beta_i \beta_j}{\beta_i + \beta_j} s_{ij} \frac{u_j - u_i}{d_{ij}} - \frac{2\beta_i \beta_j}{\beta_i + \beta_j} \frac{s_{ij}}{d_{ij}} \left(-[u] + \frac{d_{ij}}{2\beta_j} \left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma \right] \right) \\
&= \tilde{\beta}_{ij} s_{ij} \frac{u_j - u_i}{d_{ij}} - \tilde{\beta}_{ij} \frac{s_{ij}}{d_{ij}} \left(-[u] + \frac{d_{ij}}{2\beta_j} \left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma \right] \right),
\end{aligned}$$

where $\tilde{\beta}_{ij}$ is the harmonic mean between β_i and β_j, i.e.

$$\tilde{\beta}_{ij} = \frac{|\phi_i| + |\phi_j|}{|\phi_i|/\beta_i + |\phi_j|/\beta_j} = \frac{2\beta_i \beta_j}{\beta_i + \beta_j}.$$

The contribution of the interaction between the degrees of freedom i and j to the linear system is therefore

$$\tilde{\beta}_{ij} s_{ij} \frac{u_j - u_i}{d_{ij}},$$

while the contribution to the right-hand side is

$$\tilde{\beta}_{ij} \frac{s_{ij}}{d_{ij}} \left(-[u] + \frac{d_{ij}}{2\beta_j} \left[\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma \right] \right).$$

Note that we made use of the fact that ϕ is a distance function and u_{ij} is midway between i and j to simplify the expression. Similarly, we can derive the contribution of the interaction between i and j for the case when $\phi_i < 0$ and obtain the general

expression for the interaction between any degrees of freedom i and j

$$\tilde{\beta}_{ij} s_{ij} \frac{u_j - u_i}{d_{ij}} + \texttt{Vol}(\mathcal{C}) \cdot k_i \cdot u_i = \tilde{\beta}_{ij} \frac{s_{ij}}{d_{ij}} \left(-\text{sign}(\phi_i)[u] + \frac{d_{ij}}{2\beta_j} [\beta \underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}_\Gamma] \right) + \texttt{Vol}(\mathcal{C}) \cdot f_i,$$

where $\texttt{Vol}(\mathcal{C})$ is the volume of the Voronoi cell associated to the degree of freedom i and k_i and f_i are the respective values of k and f at the degree of freedom i. This discretization is entirely implicit and leads to a symmetric positive definite matrix. The discontinuities contribute only to the right-hand side of the linear system. Note that this formulation is identical to the Ghost Fluid Method of [182] in the case where the irregular interface is orthogonal to the flux between the two degrees of freedom and located midway. In fact, the Voronoi Interface Method can be interpreted as a Ghost Fluid Method where the flux between to degrees of freedom is guaranteed to be orthogonal to the face connecting their respective control volumes. We solve the linear system with the Conjugate Gradient iterative solver provided by the Petsc libraries [108, 75] and preconditioned with the Hypre multigrid [202]. We enforce Dirichlet boundary conditions on $\partial\Omega$.

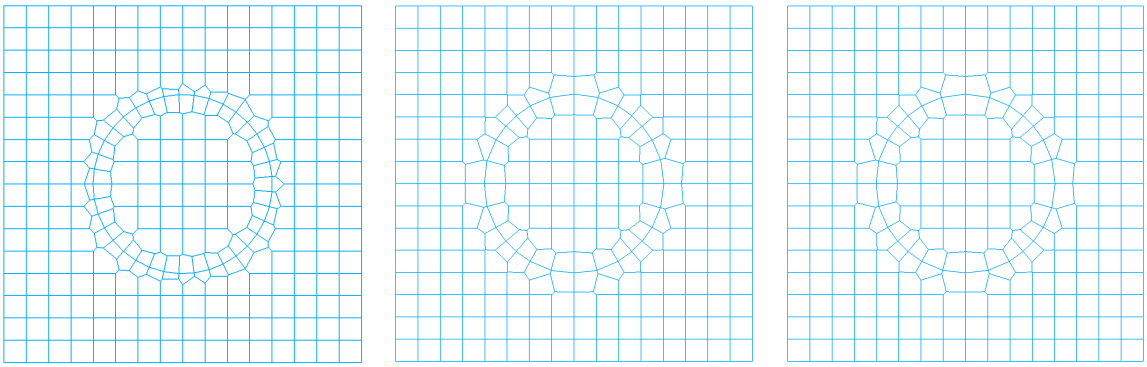
5.4 Numerical validation on uniform meshes

In this section, we validate the addition of the degrees of freedom along the irregular interface and analyze the convergence of our method on various examples. In order to demonstrate that our solver captures the discontinuities properly, all the results in this section are presented on meshes that are uniform away from the interface. Doing so, we make sure that the error on the interface dominates the overall error. Since the degrees of freedom are the seeds of the Voronoi cells, it is convenient to compute the gradient of the solution at every point located in the middle of two degrees of freedom, i.e. $\underline{\boldsymbol{\nabla}} u_{ij} \cdot \underline{\boldsymbol{n}}_{ij} = (u_j - u_i)/d_{ij}$ where $\underline{\boldsymbol{n}}_{ij}$ is the normal to the edge of the Voronoi cell

connecting the degrees of freedom i and j. The errors presented are normalized.

5.4.1 Validation of the construction of the Voronoi diagrams close to the interface

In this first example, we are interested in the influence of the quality of the mesh close to the interface. We consider three different possibilities, represented in figure 5.5 for a circular irregular interface described by $\phi(x,y) = -\sqrt{x^2+y^2} + r_0$, with $r_0 = 0.5$, in a domain $\Omega = [-1,1]^2$.



explicit non-smoothed smoothed

Figure 5.5: Visualization of the three different meshes on a resolution $2^4 \times 2^4$ for section 5.4.1.

For the first case, we place the new degrees of freedom at regular intervals on the irregular interface, making use of the explicit parametric expression available for a circle. We choose to place $N = 1.5\lfloor \frac{2\pi r_0}{\min(x_{min}, y_{min})} \rfloor$ new degrees of freedom on either side of the interface, at a distance $\texttt{diag}/5$ from the interface with $\texttt{diag} = \sqrt{x_{min}^2 + y_{min}^2}$. For the second case, we place the new degrees of freedom according to the procedure described in section 5.2.2, at a distance $\frac{\texttt{diag}}{5}$ from the interface. Finally, for the third case, we start from the partition obtained with the second case and modify it according to the procedure described in section 5.2.3 to obtain a smoothed mesh.

We monitor the convergence of our method on these three meshes for the following solution taken from [161],

$$u(x,y) = \begin{cases} 1 + \log(2\sqrt{x^2+y^2}) & \text{if } \phi(x,y) < 0, \\ 1 & \text{if } \phi(x,y) > 0, \end{cases}$$

and $\beta(x,y) = 1$. Note that for this case we have $[u] = 0$ and $[\underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}] = 2$, with continuous β and a discontinuity in the flux across the interface. A representation of the solution is given in figure 5.6 together with a visualization of the localization of the error. We report the convergence of the solver on this example for the three different meshes in tables 5.1 and 5.2. We observe second-order convergence for the solution and first-order convergence for the gradient of the solution, and very similar errors for all three meshes. We conclude that smoothing the mesh obtained with the procedure explained in section 5.2.3 does not seem to improve the accuracy of the solver.

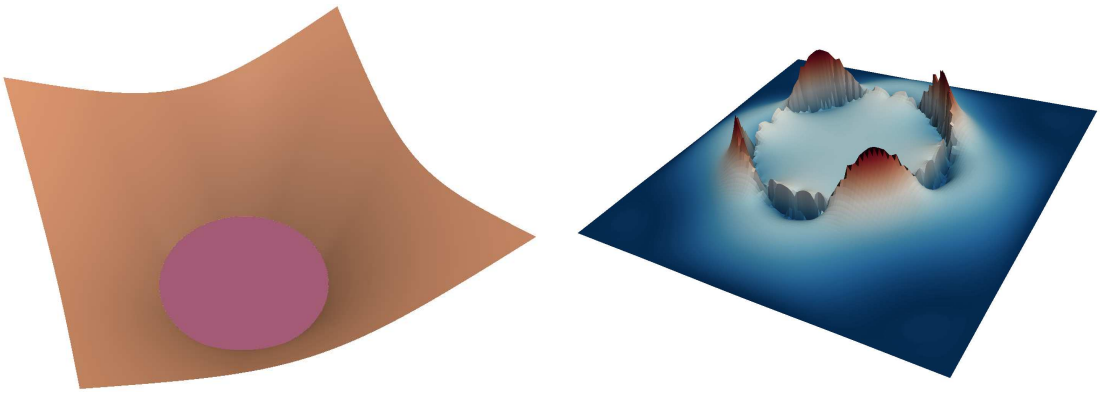


Figure 5.6: Left: representation of the solution for example 5.4.1. Right: visualization of the localization of the error on a non-smoothed mesh (case 2) of resolution $2^7 \times 2^7$.

	explicit		non-smoothed		smoothed	
resolution	solution	order	solution	order	solution	order
2^3	$3.66 \cdot 10^{-3}$	-	$1.27 \cdot 10^{-2}$	-	$1.20 \cdot 10^{-2}$	-
2^4	$1.79 \cdot 10^{-3}$	1.03	$2.34 \cdot 10^{-3}$	2.44	$2.20 \cdot 10^{-3}$	2.45
2^5	$5.77 \cdot 10^{-4}$	1.63	$6.17 \cdot 10^{-4}$	1.92	$6.02 \cdot 10^{-4}$	1.87
2^6	$1.56 \cdot 10^{-4}$	1.89	$1.62 \cdot 10^{-4}$	1.93	$1.61 \cdot 10^{-4}$	1.91
2^7	$4.32 \cdot 10^{-5}$	1.86	$4.45 \cdot 10^{-5}$	1.87	$4.23 \cdot 10^{-5}$	1.86
2^8	$1.13 \cdot 10^{-5}$	1.94	$1.14 \cdot 10^{-5}$	1.97	$1.14 \cdot 10^{-5}$	1.96
2^9	$2.83 \cdot 10^{-6}$	1.99	$2.96 \cdot 10^{-6}$	1.95	$2.95 \cdot 10^{-6}$	1.94
2^{10}	$7.19 \cdot 10^{-7}$	1.98	$7.46 \cdot 10^{-7}$	1.99	$7.45 \cdot 10^{-7}$	1.98

Table 5.1: Convergence of the error on the solution in the L^∞ norm for example 5.4.1. The first case corresponds to the degrees of freedom placed along the interface using the explicit parametrization, the second case corresponds to the mesh obtained following the method described in section 5.2.2, and the third case is the smoothed version of case 2.

	explicit		non-smoothed		smoothed	
resolution	gradient	order	gradient	order	gradient	order
2^3	$4.60 \cdot 10^{-2}$	-	$4.71 \cdot 10^{-2}$	-	$4.24 \cdot 10^{-2}$	-
2^4	$3.28 \cdot 10^{-2}$	0.49	$2.40 \cdot 10^{-2}$	0.98	$2.36 \cdot 10^{-2}$	0.84
2^5	$1.79 \cdot 10^{-2}$	0.87	$1.23 \cdot 10^{-2}$	0.96	$1.23 \cdot 10^{-2}$	0.94
2^6	$9.56 \cdot 10^{-3}$	0.91	$8.34 \cdot 10^{-3}$	0.56	$8.35 \cdot 10^{-3}$	0.56
2^7	$5.15 \cdot 10^{-3}$	0.89	$3.94 \cdot 10^{-3}$	1.08	$3.94 \cdot 10^{-3}$	1.08
2^8	$2.56 \cdot 10^{-3}$	1.01	$2.12 \cdot 10^{-3}$	0.90	$2.12 \cdot 10^{-3}$	0.90
2^9	$1.30 \cdot 10^{-3}$	0.98	$1.12 \cdot 10^{-3}$	0.92	$1.12 \cdot 10^{-3}$	0.92
2^{10}	$6.61 \cdot 10^{-4}$	0.98	$5.61 \cdot 10^{-4}$	1.00	$5.61 \cdot 10^{-4}$	1.00

Table 5.2: Convergence of the error on the gradient of the solution in the L^∞ norm for example 5.4.1. The first case corresponds to the degrees of freedom placed along the interface using the explicit parametrization, the second case corresponds to the mesh obtained following the method described in section 5.2.2, and the third case corresponds to its smoothed version.

5.4.2 Influence of the smoothing of the mesh

We further consider the influence of the smoothing procedure described in section 5.2.3. This time, we consider an interface described by $\phi(x,y) = -\sqrt{x^2+y^2} + r_0 + r_1\cos(5\theta)$, with $r_0 = 0.5$, $r_1 = 0.15$ and θ the angle between (x,y) and $(1,0)$, in a domain $\Omega = [-1,1]^2$. Since we do not have an explicit parametrization of the interface that would enable to place the degrees of freedom at regular intervals, we only consider the mesh generated form the procedure described in section 5.2.2 and its smoothed version obtained by applying the procedure described in section 5.2.3. Figure 5.7 gives a visualization of the meshes obtained.

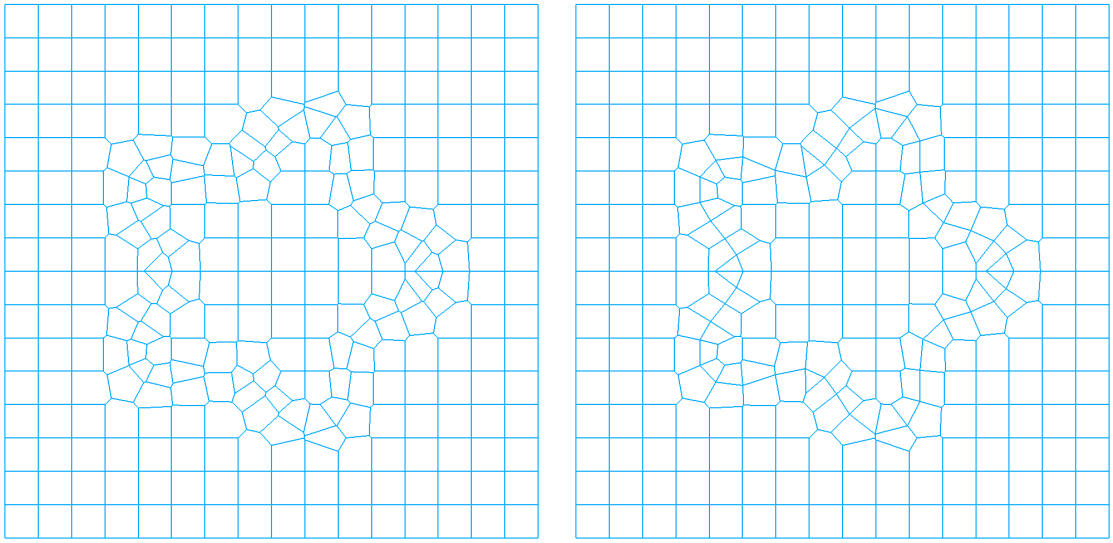


Figure 5.7: Visualization of the non-smoothed (left) and smoothed (right) meshes on a resolution $2^4 \times 2^4$ for section 5.4.2.

For this section, we choose to work with the exact solution taken from [162]

$$u(x,y) = \begin{cases} 0 & \text{if } \phi(x,y) < 0, \\ e^x\cos(y) & \text{if } \phi(x,y) > 0, \end{cases}$$

with $\beta^- = \beta^+ = 1$. The solution is represented in figure 5.8. We monitor the convergence of the solver in table 5.3 and observe second-order convergence for the solution and first-order convergence for the gradient of the solution in both cases. Given that the smoothing algorithm requires additional processing and does not seem to improve the accuracy (in fact, we notice for this particular example that the non-smoothed results are more accurate), we choose to work with the non-smoothed mesh constructed as described in section 5.2.2 for the remaining of this article.

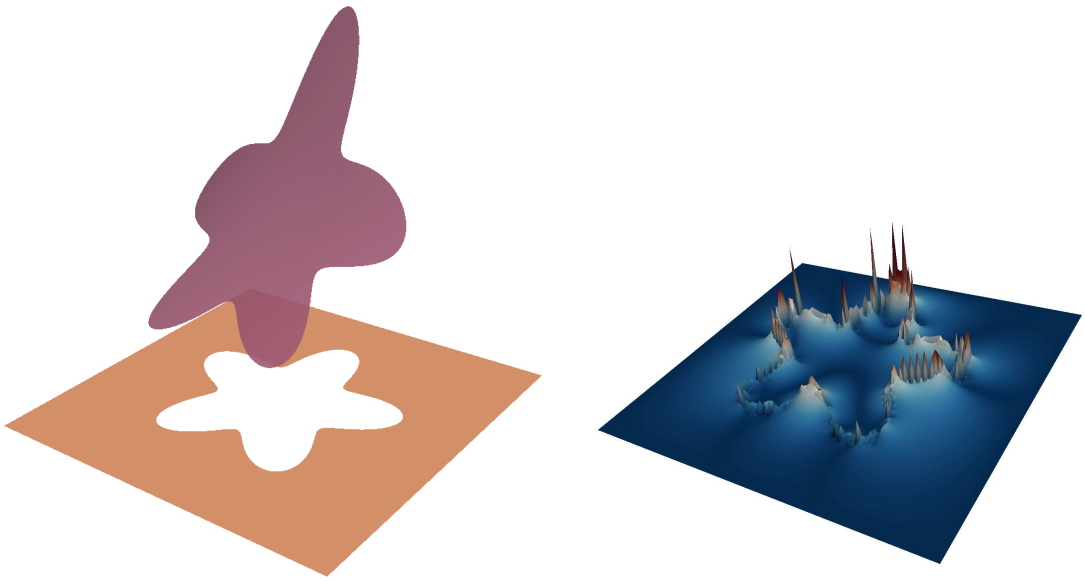


Figure 5.8: Left: representation of the solution for example 5.4.2. Right: visualization of the error on a non-smoothed mesh and for a resolution $2^7 \times 2^7$.

5.4.3 Example with a discontinuity in the diffusion coefficient

We now consider the exact solution

$$u(x,y) = \begin{cases} \frac{x(\rho+1)-x(\rho-1)r_0^2/r^2}{\rho+1+r_0^2(\rho-1)} & \text{if } \phi(x,y) < 0, \\ \\ \frac{2x}{\rho+1+r_0^2(\rho-1)} & \text{if } \phi(x,y) > 0, \end{cases}$$

	non-smoothed				smoothed			
res.	solution	order	gradient	order	solution	order	gradient	order
2^3	$2.39 \cdot 10^{-3}$	-	$3.01 \cdot 10^{-2}$	-	$2.00 \cdot 10^{-2}$	-	$1.06 \cdot 10^{-1}$	-
2^4	$1.06 \cdot 10^{-3}$	1.17	$5.08 \cdot 10^{-1}$	-4.07	$1.44 \cdot 10^{-2}$	0.47	$5.27 \cdot 10^{-1}$	-2.32
2^5	$3.43 \cdot 10^{-4}$	1.63	$8.42 \cdot 10^{-3}$	5.91	$4.21 \cdot 10^{-3}$	1.78	$2.84 \cdot 10^{-2}$	4.21
2^6	$6.82 \cdot 10^{-5}$	2.33	$3.95 \cdot 10^{-3}$	1.09	$1.62 \cdot 10^{-3}$	1.38	$1.66 \cdot 10^{-2}$	0.78
2^7	$2.79 \cdot 10^{-5}$	1.29	$2.93 \cdot 10^{-3}$	0.43	$4.55 \cdot 10^{-4}$	1.83	$8.03 \cdot 10^{-3}$	1.04
2^8	$7.35 \cdot 10^{-6}$	1.92	$1.40 \cdot 10^{-3}$	1.06	$1.11 \cdot 10^{-4}$	2.03	$2.66 \cdot 10^{-3}$	1.59
2^9	$1.89 \cdot 10^{-6}$	1.96	$6.31 \cdot 10^{-4}$	1.15	$2.76 \cdot 10^{-5}$	2.01	$9.62 \cdot 10^{-4}$	1.47
2^{10}	$4.75 \cdot 10^{-7}$	1.99	$3.28 \cdot 10^{-4}$	0.94	$6.86 \cdot 10^{-6}$	2.01	$3.34 \cdot 10^{-4}$	1.53

Table 5.3: Convergence of the error on the solution and its gradient in the L^∞ norm for example 5.4.2.

with $r_0 = .5$, $r = \sqrt{x^2+y^2}$, $\phi(x,y) = -r^2 + r_0^2$ and $\rho = \beta^+/\beta^-$ in the domain $\Omega = [-1,1]^2$. This corresponds to the example 7.3 from [161]. In this case, u is continuous, but the diffusion coefficient β experiences a large jump across the irregular interface Γ. We also have $[\underline{\boldsymbol{\nabla}} u \cdot \underline{\boldsymbol{n}}] \neq 0$. A visualization of the solution is given in figure 5.9. The gradient of the solution is given by

$$\underline{\boldsymbol{\nabla}} u(x,y) = \begin{cases} \frac{1}{\rho+1+r_0^2(\rho-1)} \begin{pmatrix} \rho + 1 - r_0^2(\rho-1)\frac{y^2-x^2}{(x^2+y^2)^2} \\ r_0^2(\rho-1)\frac{2xy}{(x^2+y^2)^2} \end{pmatrix} & \text{if } \phi(x,y) < 0, \\ \frac{1}{\rho+1+r_0^2(\rho-1)} \begin{pmatrix} 2 \\ 0 \end{pmatrix} & \text{if } \phi(x,y) > 0. \end{cases}$$

The errors on the solution and its gradient are monitored in table 5.4 which shows second-order convergence on the solution and first-order convergence on its gradient. Figure 5.10 provides a visualization of the localization of the error. We also monitor the evolution of the 1-norm condition number of the matrix of the linear system as the mesh is refined in and present the results in table 5.5. The condition number depends on the mesh resolution and on the diffusion coefficient. When the diffusion coefficient is large,

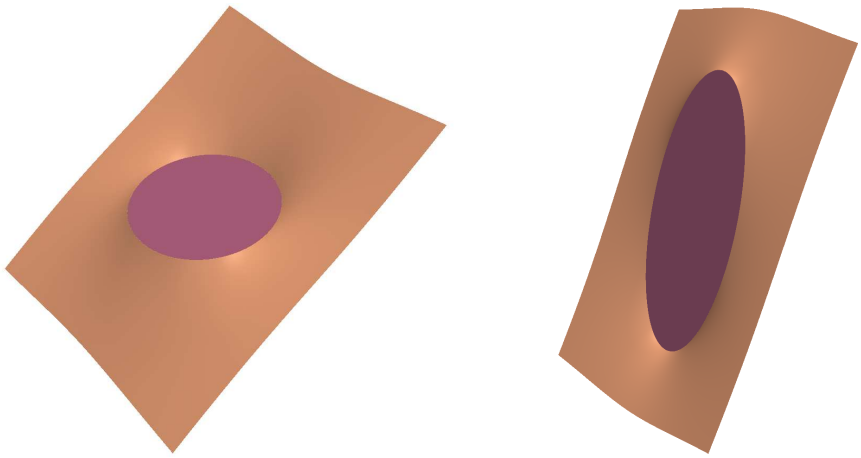


Figure 5.9: Two examples of solutions for example 5.4.3. Left: $\beta^- = 1$ and $\beta^+ = 10$. Right: $\beta^- = 10$ and $\beta^+ = 1$.

the condition number gets large rapidly. However, the discontinuities at the interface are entirely captured by the right hand side and therefore do not affect the conditioning of the matrix.

	$\beta^- = 1$, $\beta^+ = 10^5$				$\beta^- = 10^5$, $\beta^+ = 1$			
res.	solution	order	gradient	order	solution	order	gradient	order
2^3	$1.10 \cdot 10^{-2}$	-	$3.84 \cdot 10^{-2}$	-	$1.74 \cdot 10^{-2}$	-	$1.14 \cdot 10^{-1}$	-
2^4	$3.42 \cdot 10^{-3}$	1.68	$1.93 \cdot 10^{-2}$	1.00	$4.70 \cdot 10^{-3}$	1.89	$5.65 \cdot 10^{-2}$	1.01
2^5	$1.39 \cdot 10^{-3}$	1.30	$1.17 \cdot 10^{-2}$	0.72	$1.26 \cdot 10^{-3}$	1.90	$2.54 \cdot 10^{-2}$	1.15
2^6	$3.82 \cdot 10^{-4}$	1.86	$6.19 \cdot 10^{-3}$	0.92	$3.39 \cdot 10^{-4}$	1.90	$1.77 \cdot 10^{-2}$	0.53
2^7	$1.34 \cdot 10^{-4}$	1.51	$3.59 \cdot 10^{-3}$	0.79	$1.05 \cdot 10^{-4}$	1.69	$8.06 \cdot 10^{-3}$	1.13
2^8	$3.43 \cdot 10^{-5}$	1.97	$1.84 \cdot 10^{-3}$	0.96	$2.67 \cdot 10^{-5}$	1.98	$4.87 \cdot 10^{-3}$	0.73
2^9	$9.76 \cdot 10^{-6}$	1.81	$1.05 \cdot 10^{-3}$	0.82	$7.55 \cdot 10^{-6}$	1.82	$2.40 \cdot 10^{-3}$	1.02
2^{10}	$2.58 \cdot 10^{-6}$	1.92	$5.41 \cdot 10^{-4}$	0.95	$1.96 \cdot 10^{-6}$	1.95	$1.25 \cdot 10^{-3}$	0.94

Table 5.4: Convergence on the solution and its gradient for example 5.4.3, for two different combinations of diffusion coefficients.

resolution	$\beta^- = 1, \beta^+ = 10^5$	$\beta^- = 10^5, \beta^+ = 1$	$\beta^- = \beta^+ = 10^5$	$\beta^- = \beta^+ = 1$
2^3	$2.38 \cdot 10^6$	$2.02 \cdot 10^6$	$5.59 \cdot 10^6$	$1.72 \cdot 10^2$
2^4	$1.17 \cdot 10^7$	$5.09 \cdot 10^6$	$1.23 \cdot 10^7$	$8.11 \cdot 10^2$
2^5	$4.86 \cdot 10^7$	$1.57 \cdot 10^7$	$2.69 \cdot 10^7$	$3.66 \cdot 10^3$
2^6	$2.09 \cdot 10^8$	$7.18 \cdot 10^7$	$5.72 \cdot 10^7$	$1.58 \cdot 10^4$
2^7	$8.52 \cdot 10^8$	$3.14 \cdot 10^8$	$1.19 \cdot 10^8$	$6.59 \cdot 10^4$
2^8	$3.86 \cdot 10^9$	$1.32 \cdot 10^9$	$2.42 \cdot 10^8$	$2.70 \cdot 10^5$
2^9	$1.48 \cdot 10^{10}$	$5.43 \cdot 10^9$	$4.89 \cdot 10^8$	$1.09 \cdot 10^6$
2^{10}	$6.59 \cdot 10^{10}$	$2.20 \cdot 10^{10}$	$9.84 \cdot 10^8$	$4.39 \cdot 10^6$

Table 5.5: Evolution of the condition number as the mesh is refined for example 5.4.3. The condition number depends solely on the diffusion coefficient β and on the resolution on the mesh since the discontinuities are captured by the right hand side of the linear system.

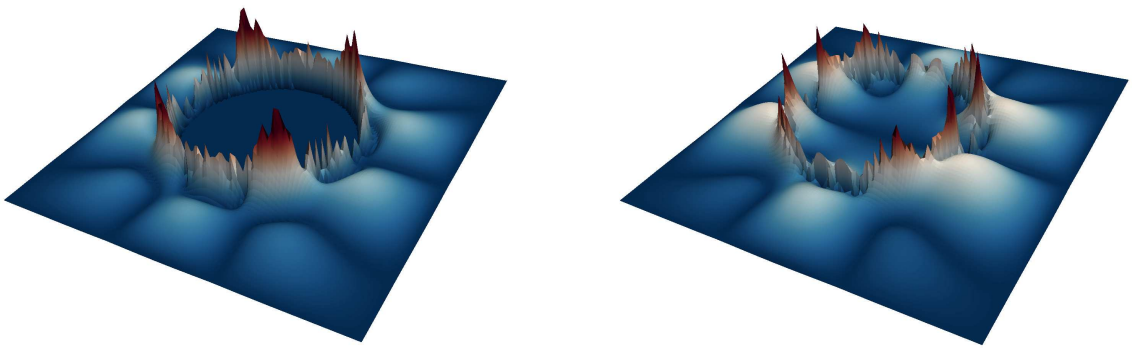


Figure 5.10: Visualization of the localization of the error in the L^∞ norm on a grid of resolution $2^7 \times 2^7$ for example 5.4.3. Left: $\beta^- = 1$ and $\beta^+ = 10^5$. Right: $\beta^- = 10^5$ and $\beta^+ = 1$.

5.4.4 A complete example

This example is meant to test our method to its full capacity, with discontinuities in all four quantities (the solution, its gradient, the diffusion coefficient and the flux across the interface), and with a complex irregular interface. We choose the exact solution

$$u(x,y) = \begin{cases} e^x & \text{if } \phi(x,y) < 0, \\ \cos(x)\sin(y) & \text{if } \phi(x,y) > 0, \end{cases}$$

in a domain $\Omega = [-1,1]^2$ with $\phi(x,y) = -\sqrt{x^2+y^2} + r_0 + r_1\cos(n\theta)$, where $r_0 = 0.5$, $r_1 = 0.15$ and $n = 5$, and we define the diffusion coefficient as

$$\beta(x,y) = \begin{cases} y^2\ln(x+2) + 4 & \text{if } \phi(x,y) < 0, \\ e^{-y} & \text{if } \phi(x,y) > 0. \end{cases}$$

Note that with our method, each degree of freedom has a control volume that is entirely on one side of the irregular interface, and therefore we can easily define a forcing term for any analytical solution. The exact solution and the diffusion coefficient are represented in figure 5.11. The convergence is summarized in table 5.6 and once again indicates second-order convergence for the solution and first-order convergence for the gradient of the solution. A visualization of the localization of the error on the solution is given in figure 5.12.

Figure 5.13 presents the percentage of the runtime consumed by the four principal components of the algorithm, i.e constructing the Voronoi mesh, assembling the matrix, computing the right hand side and solving the linear system. For coarse grids, the bottleneck of the computation is the construction of the Voronoi mesh, but for high resolution we observe that inverting the linear system is the costliest. These results

correspond to our implementation in the absence of parallelization. The finest resolution of 1024×1024 takes $18s$ on a single core of an Intel i7-2600 3.40GHz cpu.

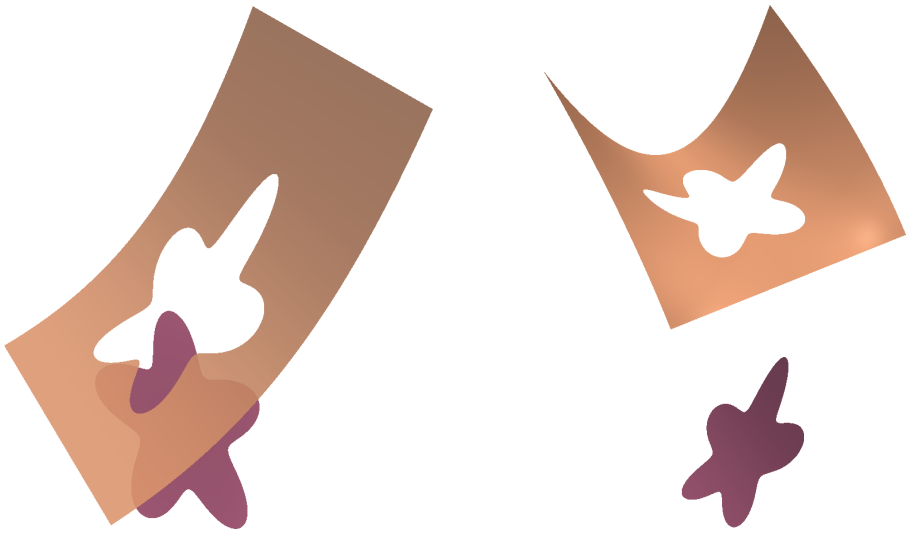


Figure 5.11: Left: visualization of the solution u for example 5.4.4. Right: visualization of the diffusion coefficient β.

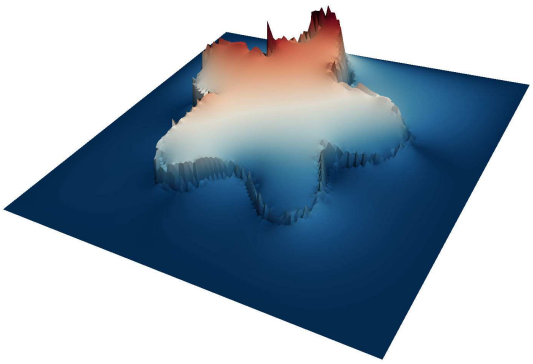


Figure 5.12: Visualization of the localization of the error for example 5.4.4 on a resolution of $2^7 \times 2^7$.

resolution	solution	order	gradient	order
2^3	$3.97 \cdot 10^{-3}$	-	$4.37 \cdot 10^{-1}$	-
2^4	$9.98 \cdot 10^{-4}$	1.99	$5.01 \cdot 10^{-1}$	-0.20
2^5	$2.90 \cdot 10^{-4}$	1.78	$3.67 \cdot 10^{-3}$	7.09
2^6	$8.84 \cdot 10^{-5}$	1.71	$1.73 \cdot 10^{-3}$	1.09
2^7	$2.06 \cdot 10^{-5}$	2.10	$9.84 \cdot 10^{-4}$	0.81
2^8	$5.22 \cdot 10^{-6}$	1.98	$4.77 \cdot 10^{-4}$	1.05
2^9	$1.33 \cdot 10^{-6}$	1.97	$2.45 \cdot 10^{-4}$	0.96
2^{10}	$3.39 \cdot 10^{-7}$	1.97	$1.23 \cdot 10^{-4}$	0.99

Table 5.6: Convergence on the solution and its gradient for example 5.4.4.

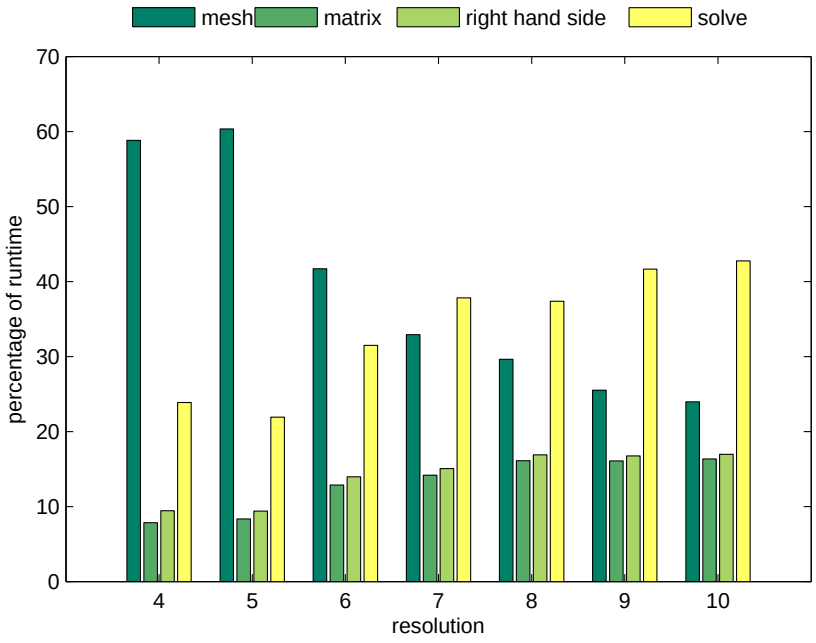


Figure 5.13: Representation of the computation time consumed by the four main sections of our implementation, constructing the mesh, assembling the matrix, computating the right hand side and solving the linear system for example 5.4.4. For coarse grids, building the Voronoi partition takes the most time, but as the resolution of the grid increases, the inversion of the linear system becomes the costliest.

5.4.5 Adding subdomains

We now propose an example with multiple subdomains. Note that the method we propose leads naturally to a linear system with N rows, the number of degrees of freedom. This number increases slightly as the number of subdomains increases and additional degrees of freedom are added next to the irregular interfaces.

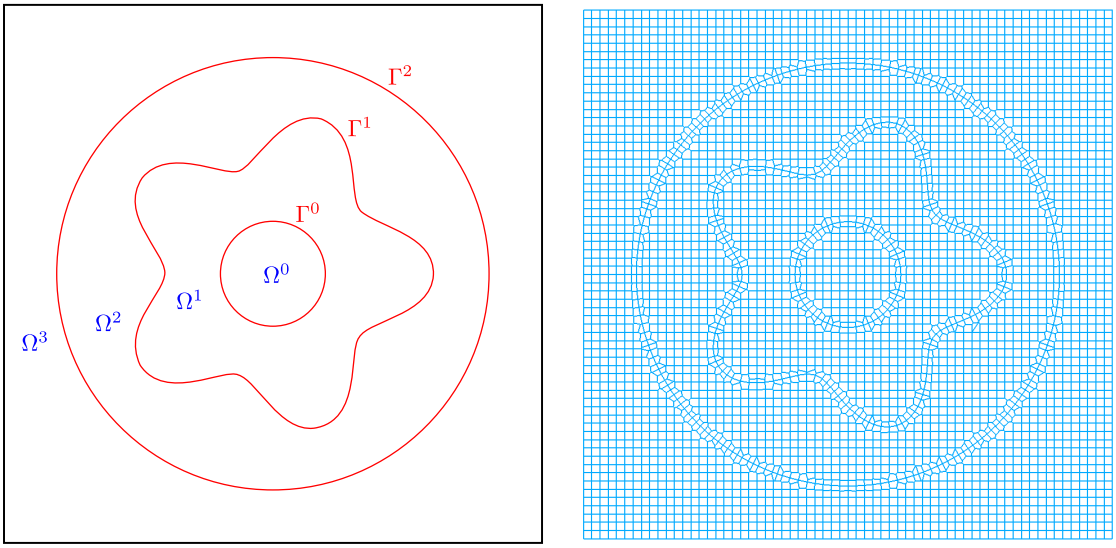


Figure 5.14: Illustration of the division of the computational domain into four subdomains, together with the voronoi mesh generated, for example 5.4.5.

For simplicity, we choose to work with non-intersecting irregular interfaces, but our method is suited for any general configuration. We divide the computational domain $\Omega = [-1, 1]^2$ in 4 subdomains represented in figure 5.14 and separated by the three contours defined by

$$\Gamma^0 = \left\{(x, y),\ \phi^0(x, y) = \sqrt{x^2 + y^2} - 0.2\right\},$$
$$\Gamma^1 = \left\{(x, y),\ \phi^1(x, y) = \sqrt{x^2 + y^2} - 0.5 + 0.1\cos(5\theta)\right\},$$
$$\Gamma^2 = \left\{(x, y),\ \phi^2(x, y) = \sqrt{x^2 + y^2} - 0.8\right\},$$

where θ is the angle between (x, y) and the x-axis. We choose the exact solution

$$u(x,y) = \begin{cases} e^x + 1.3 & \text{if } (x,y) \in \Omega^0, \\ \cos(y) + 1.8 & \text{if } (x,y) \in \Omega^1, \\ \sin(x) + 0.5 & \text{if } (x,y) \in \Omega^2, \\ -x + \ln(y+2) & \text{if } (x,y) \in \Omega^3, \end{cases}$$

and the diffusion coefficient

$$\beta(x,y) = \begin{cases} y^2 + 1 & \text{if } (x,y) \in \Omega^0, \\ e^x & \text{if } (x,y) \in \Omega^1, \\ y + 1 & \text{if } (x,y) \in \Omega^2, \\ x^2 + 1 & \text{if } (x,y) \in \Omega^3. \end{cases}$$

The solution and the diffusion coefficient are represented in figure 5.15.

The convergence of the solver is presented in table 5.7. We observe second order convergence for the solution and first order convergence for its gradient.

resolution	solution	order	gradient	order
2^4	$1.00 \cdot 10^{-3}$	-	$3.33 \cdot 10^{-3}$	-
2^5	$2.33 \cdot 10^{-4}$	2.11	$1.01 \cdot 10^{-3}$	1.72
2^6	$6.23 \cdot 10^{-5}$	1.90	$3.46 \cdot 10^{-4}$	1.54
2^7	$1.56 \cdot 10^{-5}$	2.00	$1.59 \cdot 10^{-4}$	1.12
2^8	$4.00 \cdot 10^{-6}$	1.96	$6.82 \cdot 10^{-5}$	1.22
2^9	$1.01 \cdot 10^{-6}$	1.99	$4.00 \cdot 10^{-5}$	0.77
2^{10}	$2.55 \cdot 10^{-7}$	1.99	$2.24 \cdot 10^{-5}$	0.84

Table 5.7: Convergence on the solution and its gradient in the L^∞ norm for example 5.4.5.

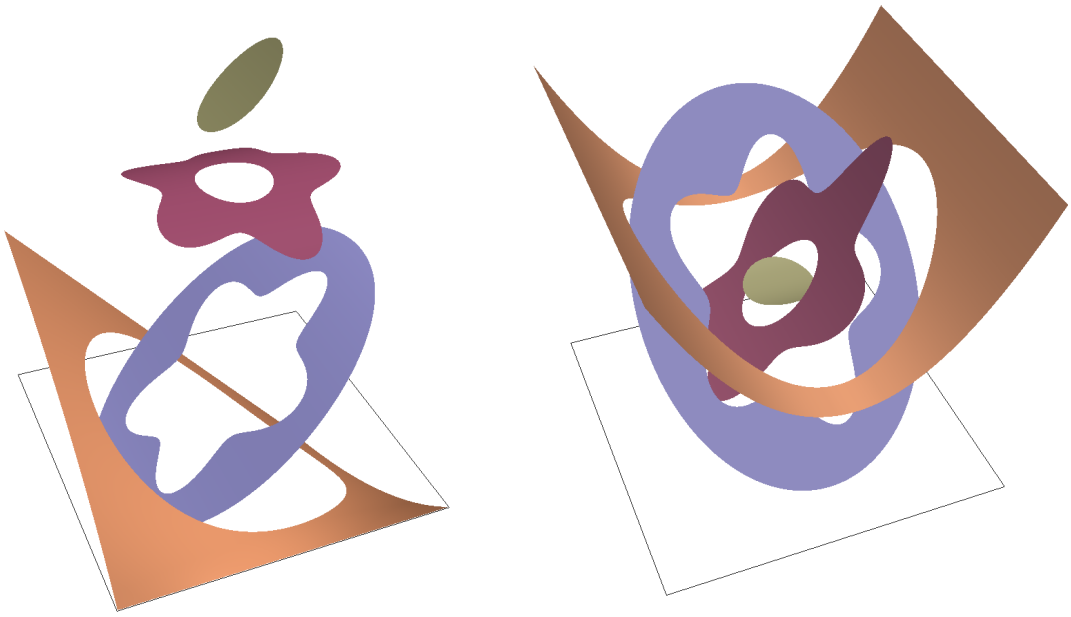


Figure 5.15: Visualization of the solution (left) and the diffusion coefficient (right) for example 5.4.5.

5.4.6 Application to three spatial dimensions

We now present an example in three spatial dimensions and with a spherical interface of radius 0.5 in a domain $\Omega = [-1, 1]^3$. We work with the exact solution

$$u(x, y, z) = \begin{cases} e^z & \text{if } \phi(x, y, z) < 0, \\ \cos(x)\sin(y) & \text{if } \phi(x, y, z) > 0, \end{cases}$$

and the diffusion coefficient

$$\beta(x, y, z) = \begin{cases} y^2 \ln(x + 2) + 4 & \text{if } \phi(x, y, z) < 0, \\ e^{-z} & \text{if } \phi(x, y, z) > 0. \end{cases}$$

The geometry, together with a slice of the solution and of the diffusion coefficient, is represented in figure 5.16. Table 5.8 presents the numerical results and indicates second-order convergence for the solution and first-order convergence for its gradient.

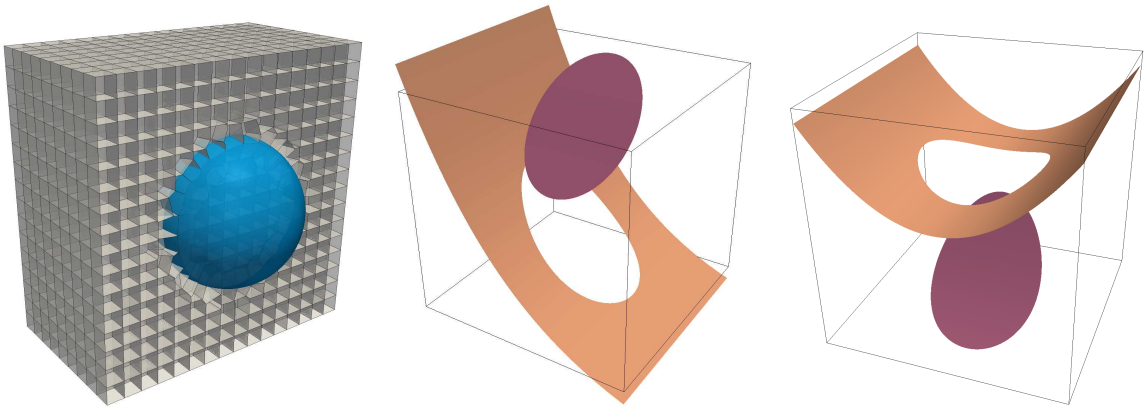


Figure 5.16: Left: representation of the irregular interface and the associated Voronoi mesh on a resolution $2^4 \times 2^4 \times 2^4$ for example 5.4.6. Center: visualization of the solution on the slice $x = 0$. Right: visualization of the diffusion coefficient on the slice $x = 0$. Note that the surfaces have been translated to facilitate the visualization.

resolution	solution	order	gradient	order
2^3	$3.61 \cdot 10^{-3}$	-	$1.13 \cdot 10^{-2}$	-
2^4	$1.21 \cdot 10^{-3}$	1.58	$7.69 \cdot 10^{-3}$	0.56
2^5	$3.04 \cdot 10^{-4}$	1.99	$3.83 \cdot 10^{-3}$	1.01
2^6	$7.74 \cdot 10^{-5}$	1.98	$2.43 \cdot 10^{-3}$	0.66
2^7	$1.97 \cdot 10^{-5}$	1.98	$1.24 \cdot 10^{-3}$	0.98

Table 5.8: Convergence on the solution and its gradient in the L^∞ norm on a sphere (example 5.4.6).

5.4.7 A complex geometry in three spatial dimensions

For a more complicated geometry, we select the intricate contour borrowed from [198] and parametrized by

$$\Gamma_{\text{trefoil}} = \left\{ \frac{R}{3} \begin{pmatrix} (2+\cos(3t))\cos(2t) \\ (2+\cos(3t))\sin(2t) \\ \sin(3t) \end{pmatrix}, t \in [0, 2\pi] \right\},$$

where $R = 0.7$ is the major radius of the trefoil. We then define

$$\Omega^+ = \left\{ \underline{\boldsymbol{x}} \in \mathbb{R}, \min_{\underline{\boldsymbol{y}} \in \Gamma_{\text{trefoil}}} \|\underline{\boldsymbol{x}} - \underline{\boldsymbol{y}}\|_2 < r \right\},$$

with $r = 0.15$ the minor radius of the trefoil. The contour is represented in figure 5.17.

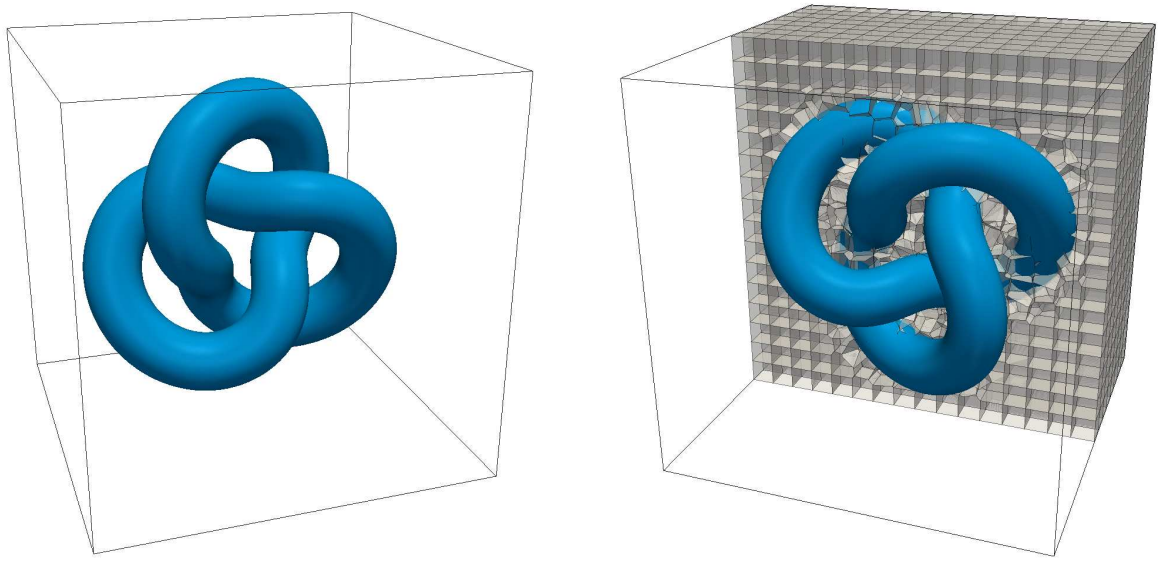


Figure 5.17: Representation of the irregular interface and the associated Voronoi mesh on a resolution $2^4 \times 2^4 \times 2^4$ for example 5.4.7.

For this example, we use the exact solution

$$u(x,y,z) = \begin{cases} yz\sin(x) & \text{if } \phi(x,y,z) < 0, \\ xy^2 + z^3 & \text{if } \phi(x,y,z) > 0, \end{cases}$$

and the diffusion coefficient

$$\beta(x,y,z) = \begin{cases} y^2 + 1 & \text{if } \phi(x,y,z) < 0, \\ e^{x+z} & \text{if } \phi(x,y,z) > 0. \end{cases}$$

Slices of the solution and of the diffusion coefficient are displayed in figure 5.18. The convergence of our method on this complex irregular interface is reported in table 5.9 and once more indicates second order convergence for the solution and first order convergence for its gradient.

resolution	solution	order	gradient	order
2^4	$3.89 \cdot 10^{-3}$	-	$1.44 \cdot 10^{-1}$	-
2^5	$1.34 \cdot 10^{-3}$	1.54	$2.56 \cdot 10^{-2}$	2.67
2^6	$3.45 \cdot 10^{-4}$	1.96	$9.75 \cdot 10^{-3}$	1.21
2^7	$8.25 \cdot 10^{-5}$	2.07	$5.04 \cdot 10^{-3}$	0.95

Table 5.9: Convergence on the solution and its gradient in the L^∞ norm on a complex three-dimensional contour (example 5.4.7).

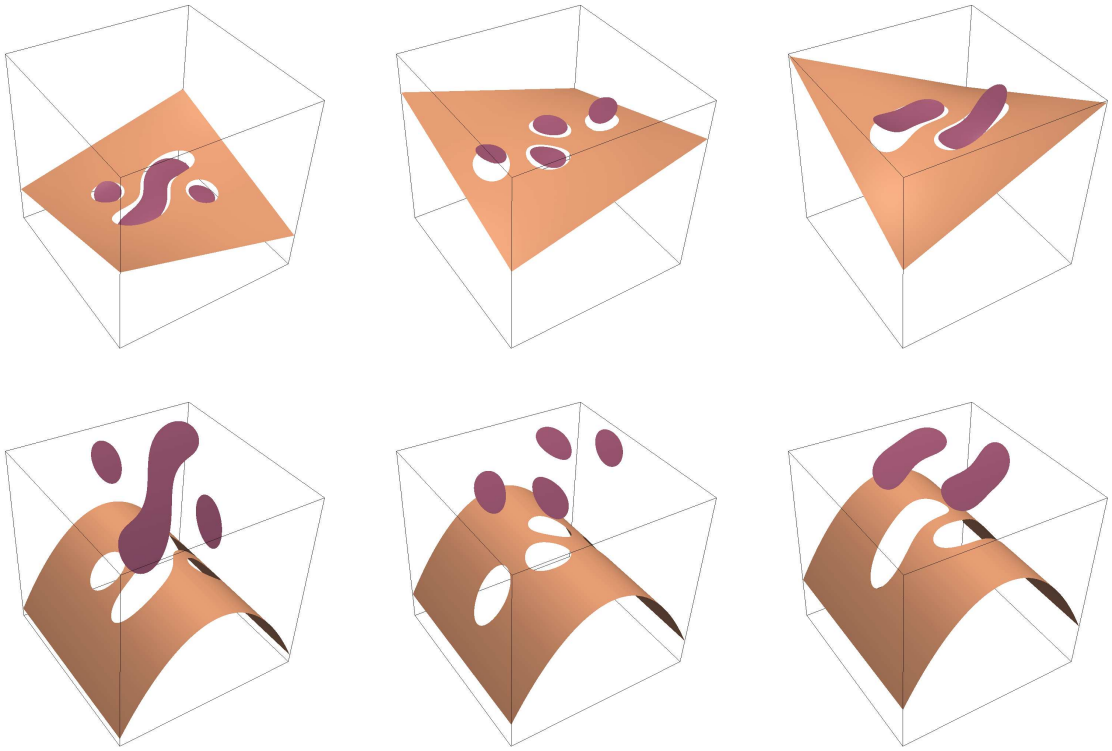


Figure 5.18: Visualization of the solution (top row) and the diffusion coefficient (bottom row) on three slices for example 5.4.7. The slices are taken, from left to right, at $x = 0.3$, $x = -0.3$ and $x = -0.5$. Note that the surfaces of the diffusion coefficient have been translated to facilitate the visualization.

5.4.8 The screened Poisson equation

This example and the following one add a non-zero k to the previous example 5.4.7. In this example, we choose $k < 0$ as

$$k(x,y,z) = \begin{cases} -e^x & \text{if } \phi(x,y,z) < 0, \\ -\cos(y)\sin(z) - 2 & \text{if } \phi(x,y,z) > 0. \end{cases}$$

The convergence results are presented in table 5.10 and show second order convergence with errors very similar to the ones obtained when $k = 0$.

resolution	solution	order	gradient	order
2^4	$3.87 \cdot 10^{-3}$	-	$1.44 \cdot 10^{-1}$	-
2^5	$1.34 \cdot 10^{-3}$	1.53	$2.25 \cdot 10^{-2}$	2.67
2^6	$3.44 \cdot 10^{-4}$	1.96	$9.74 \cdot 10^{-3}$	1.21
2^7	$8.22 \cdot 10^{-5}$	2.06	$5.04 \cdot 10^{-3}$	0.95

Table 5.10: Convergence on the solution and its gradient in the L^∞ norm on a complex three-dimensional contour for the screened Poisson equation (example 5.4.8).

5.4.9 The Helmholtz equation case

Our last example on a uniform base mesh is exactly the same than the one from the previous section but for the Helmholtz equation case, i.e. $k(x,y,z) > 0$. We set

$$k(x,y,z) = \begin{cases} e^y & \text{if } \phi(x,y,z) < 0, \\ \cos(x)\sin(z) + 2 & \text{if } \phi(x,y,z) > 0. \end{cases}$$

In this case, the linear system obtained is more complicated to solve because the matrix is no longer diagonally dominant, meaning that the problem is not convex and iterative solvers such as the Conjugate Gradient used so far are not guaranteed to converge.

Instead, we solve the linear directly with an LU decomposition. The numerical results are presented in table 5.11 and are almost identical to the results from the previous section, illustrating the second order convergence of our method for the Helmholtz equation.

resolution	solution	order	gradient	order
2^4	$3.91 \cdot 10^{-3}$	-	$1.44 \cdot 10^{-1}$	-
2^5	$1.35 \cdot 10^{-3}$	1.54	$2.26 \cdot 10^{-2}$	2.67
2^6	$3.46 \cdot 10^{-4}$	1.96	$9.75 \cdot 10^{-3}$	1.21
2^7	$8.27 \cdot 10^{-5}$	2.06	$5.05 \cdot 10^{-3}$	0.95

Table 5.11: Convergence on the solution and its gradient in the L^∞ norm on a complex three-dimensional contour for the Helmholtz equation (example 5.4.9).

5.5 Extension to adaptive meshes

In the previous section we demonstrated the efficiency of our method based on uniform meshes in both two and three spatial dimensions. However, it can be applied straightforwardly to any mesh and in this section we propose an implementation on Quad/Oc-trees.

5.5.1 Introduction to the Quad/Oc-tree data structure

A Quad/Oc-tree grid refers to a Cartesian grid that uses the Quad/Oc-tree data structure for its storage in two/three spatial dimensions. Starting from a root cell corresponding to the entire domain, four (eight in three spatial dimensions) children are created if the cell satisfies a given splitting criterion. The process is iterated recursively until the maximum allowed level is reached. The root cell has level 0 and the finest cells have the maximum level allowed. We denote a tree with coarsest level n and finest level m by level n/m. The process is illustrated in figure 5.19. This data structure provides a $O(\ln(n))$ access to the data stored at the leaves. We refer the reader to [14] for further details on the Quad/Oc-tree data structure and the associated discretization techniques.

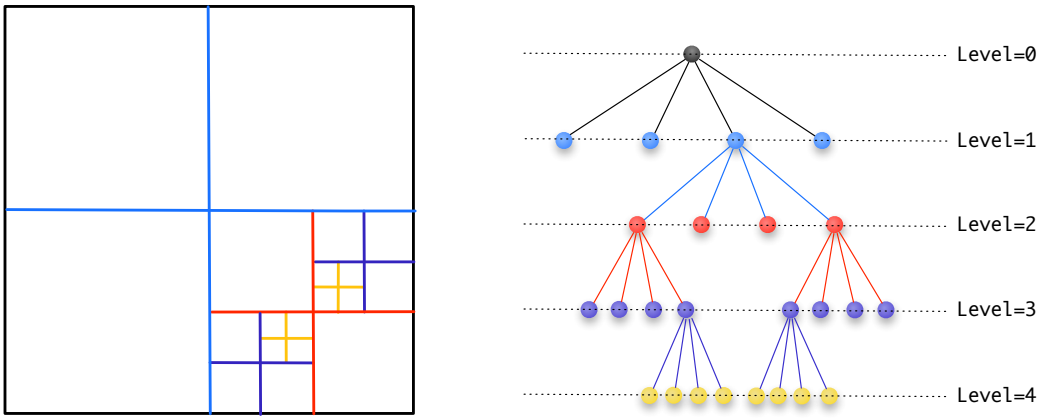


Figure 5.19: Example of a Quadtree grid.

Since this article is a proof of concept and we know the exact solution in all the numerical example we propose, we make use of this knowledge in the construction of the tree. We will use solutions of the form

$$u(\underline{\boldsymbol{x}}) = e^{-\alpha\|\underline{\boldsymbol{x}}-\underline{\boldsymbol{x}}_0\|_2^2},$$

and any given leaf $\mathcal{L}$ of the tree with center coordinates $\underline{\boldsymbol{x}}_c$ is split if $\|\underline{\boldsymbol{x}}_0 - \underline{\boldsymbol{x}}_c\|_2 < \lambda \cdot \texttt{diag}$, where `diag` is the length of the diagonal of $\mathcal{L}$ and λ controls the spread of the mesh around the peaks of the solution.

5.5.2 Solution on a Quadtree mesh

For this example, we consider the exact solution, represented in figure 5.20,

$$u(x,y) = \begin{cases} e^{-50((x+0.7)^2+(y-0.7)^2)} & \text{if } \phi(x,y) < 0, \\ e^{-50((x-0.1)^2+(y+0.1)^2)} & \text{if } \phi(x,y) > 0, \end{cases}$$

with $\beta^- = \beta^+ = 1$ and on the domain $\Omega = [-1, 1]^2$. We use the same geometry as in section 5.4.4, i.e. $\phi(x, y) = -\sqrt{x^2 + y^2} + 0.5 + 0.15\cos(5\theta)$.

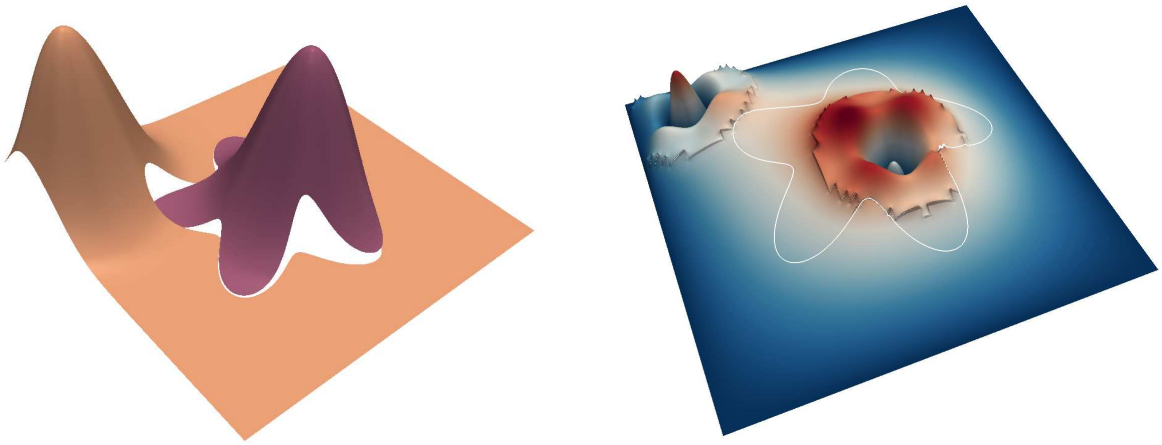


Figure 5.20: Left: visualization of the solution for example 5.5.2 with $\alpha = 10$. Higher values of α narrow the peaks. Right: representation of the error interpolated on the base Quadtree mesh of level 8/10 for $\alpha = 50$.

We start by constructing a mesh of level 5/7 with the criteria described in the previous section and with $\lambda = 8$. A visualization of this initial mesh is given in figure 5.21. Note that the mesh is not uniform close to the interface. We then split every cell of the mesh to monitor the convergence of the solver. The results are presented in table 5.12 and show second-order convergence for the solution and first-order convergence for its gradient.

resolution	solution	order	gradient	order
5/7	$2.56 \cdot 10^{-3}$	-	$8.89 \cdot 10^{-3}$	-
6/8	$6.48 \cdot 10^{-4}$	1.98	$5.14 \cdot 10^{-3}$	0.79
7/9	$1.69 \cdot 10^{-4}$	1.94	$2.52 \cdot 10^{-3}$	1.03
8/10	$4.30 \cdot 10^{-5}$	1.97	$1.33 \cdot 10^{-3}$	0.92
9/11	$1.09 \cdot 10^{-5}$	1.99	$6.84 \cdot 10^{-4}$	0.96

Table 5.12: Convergence on the solution and its gradient in the L^∞ norm on a Quadtree-based Voronoi mesh (example 5.5.2).

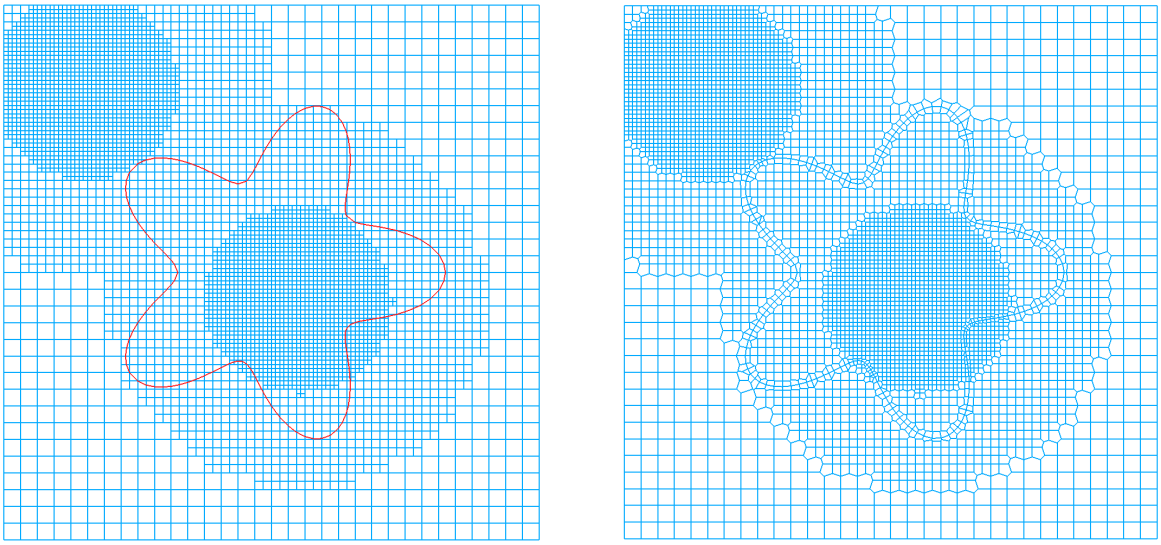


Figure 5.21: Left: the initial level 5/7 Quadtree mesh generated for example 5.5.2. The irregular interface Γ is represented in red. Right: the corresponding Voronoi mesh. Note that the mesh is not uniform along Γ.

5.5.3 Solution on an Octree mesh

For this last example, we work with the exact solution

$$u(x,y,z) = \begin{cases} e^{-50((x+0.7)^2+(y-0.7)^2+(z-.5)^2)} & \text{if } \phi(x,y,z) < 0, \\ e^{-50((x-0.1)^2+(y+0.1)^2+(z+.3)^2)} & \text{if } \phi(x,y,z) > 0, \end{cases}$$

and on the irregular interface described by

$$\phi(x,y,z) = -\sqrt{x^2+y^2+z^2} + r_0 + r_1 \cos(5\theta)\cos(\frac{2\pi}{r_0}z),$$

with $r_0 = 0.25$ and $r_1 = r_0/3$, and rotated around the z-axis, y-axis and x-axis by respectively 0.6 radians, 0.2 radians and 0.9 radians. We set $\beta^- = \beta^+ = 1$. The geometry and the Voronoi mesh generated for the initial level 3/5 are represented in figure 5.22. The convergence is presented in table 5.13 and shows second-order convergence for the

solution and first-order convergence for its gradient.

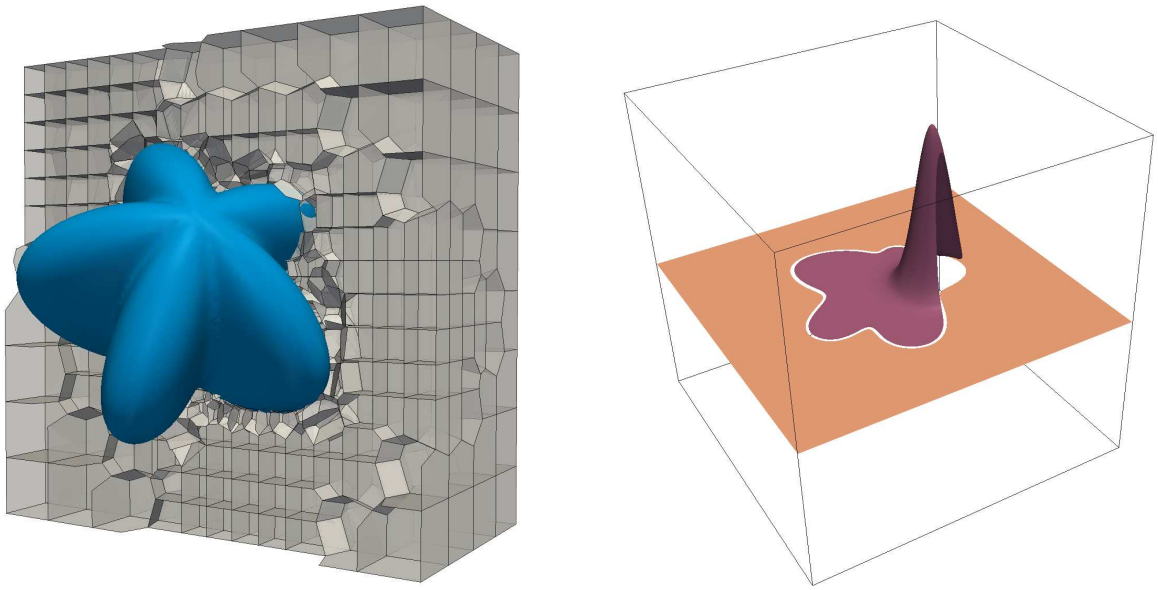


Figure 5.22: Left: representation of the geometry together with the Voronoi mesh on the initial mesh of resolution 3/5 for example 5.5.3. Note that the mesh is not uniform along Γ. Right: visualization of the solution on the $y = -0.1$ slice and after applying three successive refinement operations to the original mesh.

resolution	solution	order	gradient	order
3/5	$6.76 \cdot 10^{-2}$	-	$6.54 \cdot 10^{-2}$	-
4/6	$1.47 \cdot 10^{-2}$	2.20	$1.97 \cdot 10^{-2}$	1.73
5/7	$3.58 \cdot 10^{-3}$	2.04	$7.04 \cdot 10^{-3}$	1.49
6/8	$8.96 \cdot 10^{-4}$	2.00	$4.24 \cdot 10^{-3}$	0.73

Table 5.13: Convergence on the solution and its gradient in the L^∞ norm on star shaped irregular interface and on an Octree base mesh (example 5.5.3).

5.6 Summary

We have presented a novel fully implicit approach based on Voronoi diagrams for solving an Elliptic equation with discontinuities in the solution, its gradient, the diffusion

coefficient and the flux across an irregular interface. The interface was captured through a level-set framework, and the equation was discretized with a finite volumes approach on the local Voronoi mesh. The contributions from the discontinuities were included naturally in the right-hand-side of the linear system, preserving its positive symmetric definiteness. We demonstrated second-order convergence of the solution and first-order convergence of its gradient, in the L^∞ norm, in both two and three spatial dimensions and on both uniform and adaptive Quad/Oc-tree base meshes. Large ratios in the diffusion coefficients are readily considered. We believe this approach could be utilized for numerous physical applications including those mentioned in the introduction.

Chapter 6

Comparison of the Voronoi Interface Method with the Ghost Fluid Method

6.1 Introduction

The Poisson equation with discontinuities across irregular interfaces emerge in applications such as multiphase flows with and without phase change, in heat transfer, in electrokinetics or in the modeling of biomolecules' electrostatics. Several numerical methods have been proposed to solve this system, each with their own advantages and disadvantages. One approach is in the context of Discontinuous Galerkin methods, an extension of the finite element method (FEM), e.g. see [148, 149, 203, 151, 152, 153, 193, 204, 194, 195, 177, 205] and the references therein. Finite element methods lead to symmetric positive definite linear systems that can be efficiently solved with fast iterative solvers [206]. In addition, FEM-type approaches can derive and use *a priori* error estimates to refine the mesh where higher resolution is needed. However, FEM-type methods rely on the quality of the underlying mesh, which is often difficult to obtain in cases where the irregular domain undergoes large deformations. In this case, it is challenging to generate a mesh with elements that pass a quality measure needed to ensure

accurate solutions.

Another approach is within the context of finite difference methods (FDM), e.g. see [135, 207, 208, 209, 182, 104, 111, 210, 211, 212] and the references therein. For finite difference methods the grid is Cartesian (uniform or adaptive), which leads to a straightforward grid generation process. However, interfaces must be represented by other means and the treatment of boundary conditions requires additional considerations.

In order to capture the interface and enforce the correct boundary conditions, several approaches have been explored. The immersed boundary method (for example [99, 213, 214, 215, 58]) uses the δ-formulation that smears out the solution profile across the interface. This produces algorithms that are straightforward to implement since they are similar to solving the same equations on a regular domain. However, the smearing of the solution introduces $O(1)$ errors near the interface. The immerse interface method (IIM) (see for example [135, 163, 164, 165, 162, 160, 216]) is a sharp interface method that leads to second-order accurate solution, albeit it is not a robust second-order method since it reaches its order by minimizing the truncation error. The method leads to neither symmetric nor positive definite linear systems and the application of the immerse interface method may be difficult in three spatial dimensions.

In [182], Liu *et al.* introduced a Ghost-Fluid methodology that treats the jump condition in a dimension-by-dimension framework. This leads to a linear system that is symmetric positive definite and the jump conditions only affect the right-hand side of the linear system, leading to an easy-to-implement method. However, the GFM suffers a loss in accuracy due to smearing of the tangential derivative of the discontinuity at the interface caused by this dimension-by-dimension framework, as indicated in [182].

The Voronoi Interface Method (VIM) [3] avoids the loss of accuracy of the GFM of [182] by constructing local Voronoi partitioning near the interface. The cells adjacent to the interface therefore have their faces orthogonal to the fluxes of the solution, hence

providing a configuration that can leverage the Ghost-Fluid Methodology to its fullest. The advantage is therefore that the solution is second-order accurate. A drawback is that the solution is computed at the cells' center of the Voronoi partition and an additional interpolation step is required should the solution be needed on the original Cartesian mesh. As is the case of GFM of [182], the linear system is symmetric positive definite and only its right-hand side is modified by the jump conditions. Finally, we note that even though the construction of a global Voronoi partition may be difficult and costly, the construction of a local partition, i.e. only for cells that are adjacent to the interface, is straightforward. In [3], the authors used the `Voro++` library of [107].

In this paper we highlight the performance of both GFM and VIM.

6.2 Governing Equations and Numerical Methods

We consider the Poisson equation with variable coefficient and discontinuities (jumps) across an irregular interface, Γ, which splits the computational domain Ω into Ω^+ and Ω^-, both in $\mathbb{R}^n$, $n \in \mathbb{N}$. The governing equation is:

$$\nabla \cdot (\beta \nabla u) = f \quad \text{for } \underline{\boldsymbol{x}} \in \Omega^+ \cup \Omega^-, \tag{6.1}$$

where $f = f(\underline{\boldsymbol{x}})$ and $\beta = \beta(\underline{\boldsymbol{x}})$ are given. Here, β is bounded from below by a positive constant and f is in $L^2(\Omega)$. This equation is supplemented by jump conditions on the irregular interface,

$$[u] = g,$$
$$[\beta \nabla u \cdot \underline{\boldsymbol{n}}] = h,$$

where $[a] = a_{\Gamma^+} - a_{\Gamma^-}$ denotes the jump in a across Γ. The functions $g = g(\underline{\boldsymbol{x}})$ and $h = h(\underline{\boldsymbol{x}})$ are given. Either Dirichlet, Neumann or Robin boundary conditions can be imposed at the boundaries of the computational domain, $\partial\Omega$. In order to represent the irregular interface, we use the signed distance level-set function, ϕ, which is positive inside Ω^+, negative inside Ω^- and zero on Γ. The outward normal to the irregular interface can be computed from the level-set function by

$$\underline{\boldsymbol{n}} = (n_1, n_2, n_3) = \frac{\nabla\phi}{|\nabla\phi|}.$$

6.2.1 The Ghost-Fluid Method

A detailed description of the Ghost-Fluid Method is given in [182], we will explore here the main aspects in two spatial dimensions. The discontinuity in βu_n, where u_n refers to the derivative in the normal direction to the interface, is $[\beta u_n]_\Gamma = [\beta u_x]_\Gamma n_1 + [\beta u_y]_\Gamma n_2$. The discontinuity in βu_t, where u_t refers to the derivative in the tangential direction to the interface, is $[\beta u_t]_\Gamma = [\beta u_x]_\Gamma n_2 - [\beta u_y]_\Gamma n_1$. These equations lead to $[\beta u_x]_\Gamma = [\beta u_n]_\Gamma n_1 + [\beta u_t]_\Gamma n_2$ and $[\beta u_y]_\Gamma = [\beta u_n]_\Gamma n_2 - [\beta u_t]_\Gamma n_1$. However, in order to devise a method in a dimension-by-dimension framework, the Ghost-Fluid Method smears out the discontinuity in the tangential derivative leading to the simplification $[\beta u_x]_\Gamma = [\beta u_n]_\Gamma n_1$ and $[\beta u_y]_\Gamma = [\beta u_n]_\Gamma n_2$. The discretization at each grid point i, j is then given by

$$\begin{aligned} &\frac{\beta_{i+1/2,j}\left(\dfrac{u_{i+1,j}-u_{i,j}}{\Delta x}\right) - \beta_{i-1/2,j}\left(\dfrac{u_{i,j}-u_{i-1,j}}{\Delta x}\right)}{\Delta x} \; + \\ &\frac{\beta_{i,j+1/2}\left(\dfrac{u_{i,j+1}-u_{i,j}}{\Delta y}\right) - \beta_{i,j-1/2}\left(\dfrac{u_{i,j}-u_{i,j-1}}{\Delta y}\right)}{\Delta y} \;=\; f_{i,j} + F^x + F^y, \end{aligned}$$

where $\beta_{i+1/2,j} = \dfrac{\beta_i + \beta_{i+1}}{2}$ if $x_{i,j}$ and $x_{i+1,j}$ have ϕ are on the same side of the interface or by $\beta_{i+1/2,j} = \dfrac{\beta^+\beta^- \left(|\phi^-| + |\phi^+|\right)}{\beta^+|\phi^-| + \beta^-|\phi^+|}$ otherwise. Here $\beta^\pm$ refer to the value of β adjacent to the interface in the $\Omega^\pm$ domain and Δx and Δy are the cells' sizes in the x- and y- directions, respectively. The left-hand side thus gives the same symmetric positive definite matrix as the one generated by the standard five-point stencil discretization of the Poisson equation on regular domains and only the right-hand side is altered when a discontinuity occurs. Furthermore, $F^x = F^L + F^R$ and $F^y = F^B + F^T$ are only activated if there is a discontinuity present in the local five point stencil. Here, F^L and F^R and the contribution from the left and right grid points to the current grid point and F^B and F^T are the contributions from the bottom and top grid points.

We give the details for F^L and F^R in the x-direction, referring the reader to the original paper [182] for the description for F^B and F^T in the y-direction, since they follow the same procedure. If $\phi_{i-1,j}$ and $\phi_{i,j}$ have opposite signs, define

$$\begin{aligned}
\theta &= \frac{|\phi_{i-1,j}|}{|\phi_{i,j}| + |\phi_{i-1,j}|}, \\
a_\Gamma &= \frac{a_{i,j}|\phi_{i-1,j}| + a_{i-1,j}|\phi_{i,j}|}{|\phi_{i,j}| + |\phi_{i-1,j}|}, \\
\beta_\Gamma &= \frac{b_{i,j}n^1_{i,j}|\phi_{i-1,j}| + b_{i-1,j}n^1_{i-1,j}|\phi_{i,j}|}{|\phi_{i,j}| + |\phi_{i-1,j}|}.
\end{aligned}$$

F^L is then defined as

$$F^L = \begin{cases}
\dfrac{\beta_{i-1/2,j}a_\Gamma}{(\Delta x)^2} - \dfrac{\beta_{i-1/2,j}b_\Gamma\theta}{\beta^+\Delta x} & \text{if } \phi_{i,j} \le 0 \text{ and } \phi_{i-1,j} > 0, \\
\dfrac{\beta_{i-1/2,j}a_\Gamma}{(\Delta x)^2} - \dfrac{\beta_{i-1/2,j}b_\Gamma\theta}{\beta^+\Delta x} & \text{if } \phi_{i,j} > 0 \text{ and } \phi_{i-1,j} \le 0, \\
0 \quad \text{otherwise.}
\end{cases}$$

Similarly, if $\phi_{i+1,j}$ and $\phi_{i,j}$ have opposite signs, define

$$\begin{aligned}\theta &= \frac{|\phi_{i+1,j}|}{|\phi_{i,j}| + |\phi_{i+1,j}|},\\ a_\Gamma &= \frac{a_{i,j}|\phi_{i+1,j}| + a_{i+1,j}|\phi_{i,j}|}{|\phi_{i,j}| + |\phi_{i+1,j}|},\\ \beta_\Gamma &= \frac{b_{i,j}n^1_{i,j}|\phi_{i+1,j}| + b_{i+1,j}n^1_{i+1,j}|\phi_{i,j}|}{|\phi_{i,j}| + |\phi_{i+1,j}|}.\end{aligned}$$

F^R is then defined as

$$F^R = \begin{cases} \dfrac{\beta_{i+1/2,j}a_\Gamma}{(\Delta x)^2} + \dfrac{\beta_{i+1/2,j}b_\Gamma\theta}{\beta^+\Delta x} & \text{if } \phi_{i,j} \le 0 \text{ and } \phi_{i+1,j} > 0,\\ \dfrac{\beta_{i-1/2,j}a_\Gamma}{(\Delta x)^2} - \dfrac{\beta_{i-1/2,j}b_\Gamma\theta}{\beta^+\Delta x} & \text{if } \phi_{i,j} > 0 \text{ and } \phi_{i+1,j} \le 0,\\ 0 \quad \text{otherwise.} \end{cases}$$

The Ghost-Fluid Method leads to a symmetric positive definite linear systems that captures the discontinuity in the normal derivative while smearing out the discontinuity in the tangential direction to the interface.

6.2.2 The Voronoi Interface Method

We present a summary of the method and refer the reader to [3] for a detailed description. For a given set of seeds, the Voronoi cell of a seed is defined as the points of space that are closer to that seed than any other. The union of the Voronoi cells is a tessellation of space, and the first step for the Voronoi Interface Method is to generate the Voronoi mesh associated to the background mesh chosen, which in our case is a uniform Cartesian grid. The seeds are of two types,

- if a cell of the background mesh is not crossed by the irregular interface Γ, its center is a seed of the Voronoi mesh,

- if a cell of the background mesh is crossed by the irregular interface Γ, we locate the projection of its center onto the interface and generate two seeds located on either side of the projected point at a distance $\delta = \min(\Delta x, \Delta y)/5$.

The Voronoi partition associated with those seeds is constructed using a simple geometric algorithm. We refer the reader to the `Voro++` library [107] for an efficient tool to compute Voronoi partitions in both two and three spatial dimensions.

Equation (6.1) is then discretized on the Voronoi mesh using a finite volume approach. The complete derivation is presented in [3] and leads to the discretization of the interaction between point i and point j

$$\tilde{\beta}_{ij} s_{ij} \frac{u_j - u_i}{d_{ij}} = \tilde{\beta}_{ij} \frac{s_{ij}}{d_{ij}} \left(-\text{sign}(\phi_i)[u] + \frac{d_{ij}}{2\beta_j} [\beta \nabla u \cdot \underline{\boldsymbol{n}}_\Gamma] \right) + \text{Vol}(\mathcal{C}_i) f_i,$$

where any variable of the form γ_k would be the quantity γ at point k, d_{ij} is the distance between i and j, s_{ij} is the length of the face between i and j, $\tilde{\beta}_{ij} = \dfrac{2\beta_i\beta_j}{\beta_i + \beta_j}$ and $\text{Vol}(\mathcal{C}_i)$ is the volume of the Voronoi cell associated with point i. The discontinuities at the interface only affect the right-hand side of the linear system where the irregular interface is located and the system is symmetric positive definite.

We remark that the solution is provided at the center of the Voronoi cells. If the solution is needed on the original background mesh, then an interpolation step is required. This can be done for example with least square interpolations and it does not impact the order of accuracy of the solution as long as the order of the polynomial interpolants is high enough. In this article, we work with second-order polynomial interpolants.

6.3 Numerical Experiments

We present a pair of two dimensional numerical examples where the two methods are compared. Both examples have a star shaped irregular interface. The first example has a coefficient β that is constant in the whole domain. This example has a constant discontinuity in the solution across the irregular interface but neither a discontinuity in the normal nor tangential derivatives at the interface. The second example has a β coefficient that is not constant and has a discontinuity across the irregular interface. This example has a non-constant discontinuity in the solution and non-constant discontinuities in the normal and tangential components of the gradient of the solution at the irregular interface.

6.3.1 Constant β-coefficient

Let us consider $\nabla \cdot (\beta \nabla u) = f(x, y)$ in two spatial dimensions in $\Omega = [-1, 1]^2$ with the level set function $\phi = -\sqrt{x^2 + y^2} + 0.5 + 0.15 \cos\left(5 \arctan\left(\frac{y}{x}\right)\right)$. We take $\beta = 1$ when $\phi \leq 0$ and an exact solution of $u = \cos(x)\cos(y)$. For the region $\phi > 0$, we take $\beta = 1$ and the exact solution to be $u = \cos(x)\cos(y) + 1$. A representation of the solution is given in figure 6.1. The comparison of the Ghost-Fluid Method and the Voronoi Interface Method is shown in table 6.1 for the maximum error in the solution, in table 6.2 for the average error in the solution, in table 6.3 for the maximum error of the gradient and in table 6.4 for the average error of the gradient. For the Voronoi Interface Method, we present the errors on both the Voronoi mesh and the Cartesian mesh. For the Voronoi mesh, the gradients are computed on the faces of the Voronoi cells. For this example, the solution only experiences a constant discontinuity in its solution and no discontinuities in its normal or tangential derivatives, nor is there a discontinuity in the β coefficient, which is constant in the entire domain. Both methods give second-order accuracy in the

solution in the L^∞- and L^1-norms and the solution's gradient in the L^1-norm. However, the Voronoi method gives first order accuracy for the gradient of the solution in the L^∞-norm while the GFM gives second-order accuracy for such simplified problems. A likely explanation for this difference is that the fluxes in the volume of fluid derivation for the Voronoi Interface Method are not necessarily computed at the center of the faces between two points. We also observe that the interpolation from the Voronoi mesh to the Cartesian mesh does not impact the solution itself but affects its gradient. However, the order of accuracy is conserved. We conclude that for such simple problems the GFM is preferable.

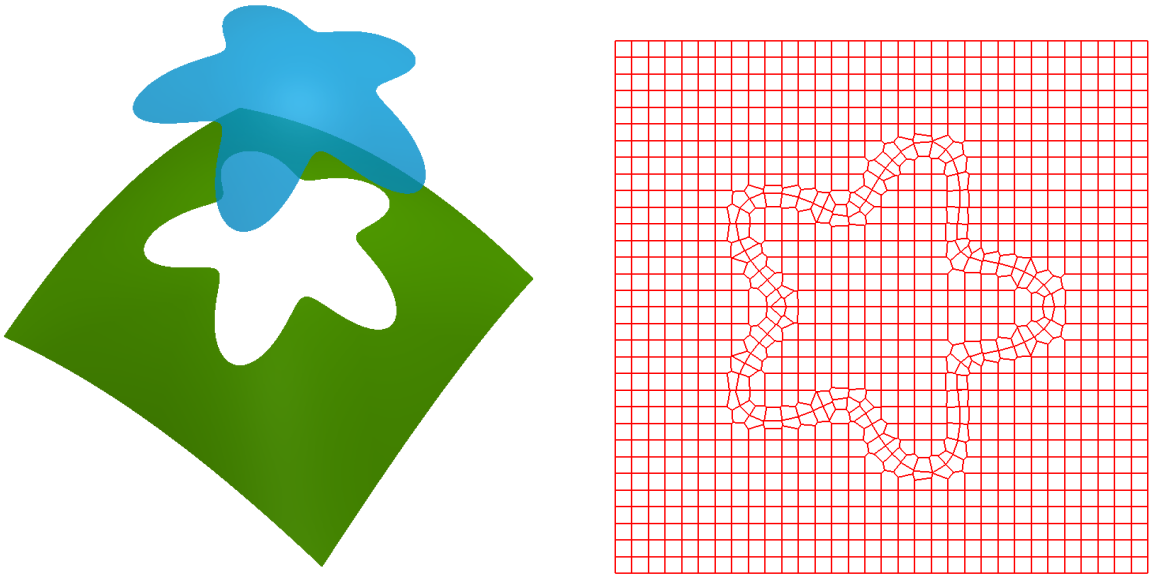


Figure 6.1: Left: visualization of the exact solution for example 6.3.1. Right: example of a Voronoi mesh generated.

6.3.2 Non-constant β-coefficient

Let us consider $\nabla \cdot (\beta \nabla u) = f(x, y)$ in two spatial dimensions in $\Omega = [-1, 1]^2$ with the level set function $\phi = -\sqrt{x^2 + y^2} + 0.5 + 0.15 \cos\left(5 \arctan\left(\frac{y}{x}\right)\right)$. We set $\beta =$

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|u-u_h\|\|_\infty$	Order	$\|\|u-u_h\|\|_\infty$	Order	$\|\|u-u_h\|\|_\infty$	Order
1/4	2.627×10^{-3}	—	3.709×10^{-3}	—	4.098×10^{-1}	—
1/8	6.587×10^{-4}	2.00	6.172×10^{-4}	2.59	5.901×10^{-2}	2.80
1/16	1.648×10^{-4}	2.00	1.565×10^{-4}	1.98	1.788×10^{-4}	8.37
1/32	4.121×10^{-5}	2.00	4.016×10^{-5}	1.96	4.030×10^{-5}	2.15
1/64	1.030×10^{-5}	2.00	1.020×10^{-5}	1.98	1.023×10^{-5}	1.98
1/128	2.576×10^{-6}	2.00	2.598×10^{-6}	1.97	2.656×10^{-6}	1.95
1/256	6.439×10^{-7}	2.00	6.591×10^{-7}	1.98	6.655×10^{-7}	2.00

Table 6.1: Comparison of the L^∞ error in the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.1.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|u-u_h\|\|_1$	Order	$\|\|u-u_h\|\|_1$	Order	$\|\|u-u_h\|\|_1$	Order
1/4	9.010×10^{-4}	—	1.790×10^{-2}	—	4.694×10^{-2}	—
1/8	2.597×10^{-4}	1.79	2.918×10^{-3}	5.04	1.417×10^{-3}	5.04
1/16	6.940×10^{-5}	1.90	7.365×10^{-5}	4.31	7.130×10^{-5}	4.31
1/32	1.792×10^{-5}	1.95	1.848×10^{-5}	2.00	1.780×10^{-5}	2.00
1/64	4.552×10^{-6}	1.98	4.625×10^{-6}	1.98	4.518×10^{-6}	1.98
1/128	1.147×10^{-6}	1.99	1.156×10^{-6}	1.98	1.142×10^{-6}	1.98
1/256	2.879×10^{-7}	1.99	2.890×10^{-7}	1.99	2.871×10^{-7}	1.99

Table 6.2: Comparison of the L^1 error in the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.1.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|\nabla u-\nabla u_h\|\|_\infty$	Order	$\|\|\nabla u-\nabla u_h\|\|_\infty$	Order	$\|\|\nabla u-\nabla u_h\|\|_\infty$	Order
1/4	4.829×10^{-3}	—	8.160×10^{-3}	—	1.872×10^{-0}	—
1/8	1.398×10^{-3}	1.79	1.550×10^{-3}	2.40	3.666×10^{-1}	2.35
1/16	3.776×10^{-4}	1.89	5.192×10^{-4}	1.58	4.180×10^{-2}	3.13
1/32	9.866×10^{-5}	1.94	4.367×10^{-4}	0.25	2.082×10^{-2}	1.01
1/64	2.530×10^{-5}	1.96	2.551×10^{-4}	0.78	1.037×10^{-2}	1.00
1/128	6.419×10^{-6}	1.98	1.376×10^{-4}	0.89	5.195×10^{-3}	0.99
1/256	1.618×10^{-6}	1.99	8.580×10^{-5}	0.68	2.597×10^{-3}	1.00

Table 6.3: Comparison of the L^∞ error in the gradient of the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.1.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|\nabla u - \nabla u_h\|\|_1$	Order	$\|\|\nabla u - \nabla u_h\|\|_1$	Order	$\|\|\nabla u - \nabla u_h\|\|_1$	Order
1/4	3.636×10^{-3}	—	2.575×10^{-3}	—	3.797×10^{-1}	—
1/8	8.704×10^{-4}	2.06	3.490×10^{-4}	2.88	2.908×10^{-2}	3.71
1/16	2.155×10^{-4}	2.01	7.553×10^{-5}	2.21	5.617×10^{-3}	2.37
1/32	5.343×10^{-5}	2.01	1.711×10^{-5}	2.14	1.474×10^{-3}	1.93
1/64	1.331×10^{-5}	2.01	4.126×10^{-6}	2.05	3.707×10^{-4}	1.99
1/128	3.322×10^{-6}	2.00	9.930×10^{-7}	2.05	9.345×10^{-5}	1.99
1/256	8.298×10^{-7}	2.00	2.465×10^{-7}	2.01	2.350×10^{-5}	1.99

Table 6.4: Comparison of the L^1 error in the gradient of the solution for the Ghost-Fluid method and the Voronoi Interface Method for example 6.3.1.

$y^2 \ln(x+2) + 4$ and the exact solution to $u = e^x$ in the region where $\phi \leq 0$. In the region where $\phi > 0$, we set $\beta = e^{-y}$ and the exact solution to $u = \cos(x)\sin(y)$. Figure 6.2 provides a representation of the solution and of the diffusion coefficient. The comparison of the Ghost-Fluid Method and the Voronoi Interface Method is shown in table 6.5 for the maximum error in the solution, in table 6.6 for the average error in the solution, in table 6.7 for the maximum error of the gradient and in table 6.8 for the average error of the gradient. Again, for the Voronoi Interface Method we present the results on both the Voronoi and on the Cartesian meshes. For this example, where there is a non-constant discontinuity in the solution, a non-constant discontinuity in the normal derivative of the solution to the interface, a non-constant discontinuity in the tangential derivative of the solution to the interface, a discontinuity in the β coefficient across the interface and a β coefficient that is not constant in each domain as in the previous example, the Ghost-Fluid Method is only first-order accurate in both the L^∞- and the L^1-norm, as well as in the L^1-norm for the gradient of the solution. However, it is not consistent in the L^∞-norm for the gradient of the solution. This example is a case where the lack of orthogonality between the cells' faces and the solution's fluxes lowers the accuracy of the GFM's dimension-by-dimension approach. The Voronoi Interface Method provides

a solution to that drawback and therefore produces a second-order accurate solution in the L^∞- and L^1-norms, and first-order accurate (resp. second-order accurate) gradients in the L^∞- (resp. L^1-) norm. Similarly to the previous example, we observe a decrease in accuracy for the gradient computed on the Cartesian mesh after the interpolation step, but the order of accuracy is conserved. The solution itself does not suffer from the interpolation step. For this type of complex problems, the Voronoi Interface Method is able to provide a second-order accurate solution and consistent gradient that the Ghost-Fluid Method cannot produce. VIM is therefore the recommended approach.

Figure 6.2: Visualization of the exact solution u (left) and of the diffusion coefficient β (right) for example 6.3.2.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|u-u_h\|\|_\infty$	Order	$\|\|u-u_h\|\|_\infty$	Order	$\|\|u-u_h\|\|_\infty$	Order
1/4	6.254×10^{-1}	—	9.802×10^{-3}	—	6.044×10^{-1}	—
1/8	4.063×10^{-1}	0.62	2.560×10^{-3}	1.94	9.178×10^{-2}	2.72
1/16	2.493×10^{-1}	0.70	7.709×10^{-4}	1.73	7.041×10^{-4}	7.03
1/32	1.474×10^{-1}	0.76	2.370×10^{-4}	1.70	2.160×10^{-4}	1.70
1/64	8.488×10^{-2}	0.80	5.560×10^{-5}	2.09	4.929×10^{-5}	2.13
1/128	4.802×10^{-2}	0.82	1.413×10^{-5}	1.98	1.293×10^{-5}	1.93
1/256	1.481×10^{-2}	0.84	3.618×10^{-6}	1.97	3.377×10^{-6}	1.94

Table 6.5: Comparions of the L^∞ error in the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.2.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|u-u_h\|\|_1$	Order	$\|\|u-u_h\|\|_1$	Order	$\|\|u-u_h\|\|_1$	Order
1/4	6.446×10^{-2}	—	2.291×10^{-3}	—	4.557×10^{-2}	—
1/8	3.393×10^{-2}	0.93	4.022×10^{-4}	2.51	2.041×10^{-3}	4.48
1/16	1.692×10^{-2}	1.00	1.038×10^{-4}	1.95	9.891×10^{-5}	4.37
1/32	7.908×10^{-3}	1.10	2.985×10^{-5}	1.80	2.718×10^{-5}	1.86
1/64	3.982×10^{-3}	0.99	7.243×10^{-6}	2.04	6.729×10^{-6}	2.01
1/128	2.155×10^{-3}	0.89	1.790×10^{-6}	2.02	1.717×10^{-6}	1.97
1/256	1.035×10^{-3}	1.06	4.509×10^{-7}	1.99	4.401×10^{-7}	1.96

Table 6.6: Comparison of the L^1 error in the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.2.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|\nabla u-\nabla u_h\|\|_\infty$	Order	$\|\|\nabla u-\nabla u_h\|\|_\infty$	Order	$\|\|\nabla u-\nabla u_h\|\|_\infty$	Order
1/4	1.259	—	1.309×10^{-2}	—	2.658×10^{-0}	—
1/8	1.227	0.04	9.931×10^{-3}	0.55	6.334×10^{-1}	2.07
1/16	1.139	0.11	2.954×10^{-3}	1.60	8.950×10^{-2}	2.82
1/32	1.156	-0.02	1.589×10^{-3}	0.89	4.373×10^{-2}	1.03
1/64	1.188	-0.04	1.038×10^{-3}	0.62	2.155×10^{-2}	1.02
1/128	1.205	-0.02	5.177×10^{-4}	1.00	1.070×10^{-2}	1.01
1/256	1.214	-0.01	3.194×10^{-4}	0.70	5.329×10^{-3}	1.01

Table 6.7: Comparison of the L^∞ error in the gradient of the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.2.

	Ghost-Fluid Method		Voronoi Interface Method		VIM Interpolated	
dx	$\|\|\nabla u - \nabla u_h\|\|_1$	Order	$\|\|\nabla u - \nabla u_h\|\|_1$	Order	$\|\|\nabla u - \nabla u_h\|\|_1$	Order
1/4	2.611×10^{-1}	—	2.126×10^{-3}	—	3.895×10^{-1}	—
1/8	1.438×10^{-1}	0.86	6.395×10^{-4}	1.73	3.400×10^{-2}	3.52
1/16	7.136×10^{-2}	1.01	1.957×10^{-4}	1.71	5.631×10^{-3}	2.59
1/32	3.922×10^{-2}	0.86	6.409×10^{-5}	1.61	1.464×10^{-3}	1.94
1/64	2.019×10^{-2}	0.96	1.593×10^{-5}	2.01	3.713×10^{-4}	1.98
1/128	1.033×10^{-2}	0.97	4.132×10^{-6}	1.95	9.359×10^{-5}	1.99
1/256	5.222×10^{-3}	0.98	1.099×10^{-6}	1.91	2.358×10^{-5}	1.99

Table 6.8: Comparison of the L^1 error in the gradient of the solution for the Ghost-Fluid Method and the Voronoi Interface Method for example 6.3.2.

6.4 Summary

This paper has considered the numerical solution of the Poisson equation with jump conditions across an irregular interface. In particular, we have compared the results obtained with the Ghost-Fluid Method and the Voronoi Interface Method. The Ghost-Fluid method imposes the jump conditions in a dimension-by-dimension framework, leading to a linear system that is symmetric positive definite in which the jump conditions only affect the right-hand side. However, the dimension-by-dimension approach forces a smearing of the tangential quantities in the jump, leading to a loss of accuracy unless both the discontinuity in the solution and the variable coefficient β are constant. The Voronoi Interface method solves that problem by constructing a Voronoi partition for cells adjacent to the interface. A finite volume discretization over those cells produces discretized fluxes that are orthogonal to the cells' faces themselves and aligned with the normal direction to the interface. The Ghost-Fluid philosophy can therefore be readily applied, resulting in a linear system that is also symmetric positive definite with only its right-hand side affected by the jump conditions. The resulting solution is second-order accurate (versus first-order accurate in the general case of the Ghost-Fluid Method) and the solution's gradient is first-order accurate (versus zeroth-order accurate in the case of the Ghost-Fluid

Method). The Voronoi Interface Method can therefore be considered superior in general. In the particular case where both the discontinuity across the interface is constant and the variable coefficient β is constant over each subdomains and across the interface, the Ghost-Fluid Method gives a second-order accurate solution and also second-order accurate gradient, giving this approach an advantage over the Voronoi Interface Method. The likely reason is that the discrete fluxes for the Voronoi Interface Method are not located at the center of the faces between two points. Finally, the Voronoi Interface Method provides the solution at the center of the Voronoi cells. An interpolation step is required if the solution on the original Cartesian mesh is needed which can for example be carried out with least square interpolations. The order of accuracy is then preserved though the quality of the gradient of the solution is impacted.

Chapter 7

Application of the Voronoi Interface Method to the electropermeabilization problem

7.1 Introduction

Electropermeabilization, also known as electroporation or electropulsation, is a significant increase in the permeability and in the electrical conductivity of the cell membrane that occurs when electric pulses of large amplitude (a few hundred volts per centimeter) are applied to cells membrane [217, 218, 219]. For high electric fields, the membrane is (reversibly or irreversibly) permeabilized, which enables the transfer of non-permeable molecules into the cell cytoplasm by diffusion through the electropermeabilized membrane areas. If the pulses are too long, too numerous, or if their amplitude is too high, the cell membrane is irreversibly destroyed and the cells are killed. Electroporation is important in the treatment of some cancers, as it provides an avenue to deliver therapeutic molecules directly into the cells of targeted areas.

Even though cell electropermeabilization is a well-known phenomenon, at least from the experimental point of view, there is a lack of predictive computational models that are validated by experiments [220]. This prevents a systematic use of electropermeabilization

in configurations far from the experimental settings. This is particularly pertinent to deep-seated tumor treatments for which irreversible electropermeabilization (IRE) or electrochemotherapy (ECT) need an accurate distribution of the electroporated region. The lack of a predictive computational framework is mainly due to the complexity of the electropermeabilization models at the cell scale [221, 222, 223, 224, 225], which are written in terms of partial differential equations in irregular geometries and that utilize non-standard transmission conditions through the cell's membrane. In addition, the experimental settings are usually far from the standard numerical configurations since most of the simulations deal with one single cell, while *in vitro* experiments, a large number of cells (in suspension or aggregated in spheroids) is considered. Therefore, important phenomena, such as cell screening, or electropermeabilization at the mesoscale (scale of cell aggregates) cannot be accounted for by existing computational approaches.

The aim of this paper is to present an accurate numerical method that provides a first step in that direction by enabling the computation of the voltage potential in cell aggregates when an electropermeabilizing electric field is applied. A cell membrane is very thin and acts as a capacitor, thus leading to a discontinuity in the electric potential where the jump is proportional to the electric flux. A variety of methods exist to solve elliptic systems with discontinuities. The finite element method is one of the most popular approach [148, 149, 151, 152]. It is well studied and guarantees high accuracy and a symmetric positive definite linear system. However, its efficiency relies on the quality of the mesh that must body-fit the irregular domain's boundary. In contrast, interface capturing methods are based on Cartesian grids that are easily generated and impose the discontinuous boundary condition implicitly. The Immersed Interface Method [135] and its development, the Immersed Finite Element Method and the Immersed Finite Volume Method [155, 156, 157], produce second-order accurate solutions in the L^∞ norm but produce asymmetric linear system in the case of discontinuous diffusion coef-

ficients. This is also the case for the interface treatment used in Mirzadeh *et al.* [226]. Cisterno and Weynans developed a second-order accurate method [199] and applied it to the electropermeabilization problem [227], though their method is not compact and leads to asymmetric linear systems. The Ghost Fluid Method [181], originally developed to address two-phase incompressible flows, has also been applied to discontinuous elliptic problems [182], resulting in a symmetric linear system, albeit limited to first-order accuracy [183]. The Ghost Fluid Method's low accuracy comes from its dimension by dimension structure. The Voronoi Interface Method, introduced in Guittet *et al.* [3], considers a structured conforming Voronoi mesh based on an underlying Cartesian grid before applying the ghost-fluid philosophy of [182]. As a result, the discretized fluxes are normal to the interface and second-order accuracy is achieved while preserving the symmetry positiveness of the linear system.

In this paper, we develop a VIM approach for the simulation of electropermeabilization for both single cells and spheroids. Multicellular tumor cells spheroids are particular cell aggregates that mimic the behavior of tumors [228], in particular from the electropermeabilization point of view [229]. Therefore, providing a computational framework that makes it possible to compute accurately the electric field in such cell aggregates is a crucial step forward for the clinical use of electropermeabilization-based therapies. In the first part of this article, we present the non-linear model for cell electropermeabilization before describing the numerical method. We then validate the computational method in two and three spatial dimensions and investigate the behavior of cell aggregates.

7.2 Electrical model for a single cell

Cell membranes are very thin and very resistive, therefore they are usually described as a two spatial dimensions surface, Γ, with given capacitance, C, and surface conduc-

tance, S, as described by Schwan, Stuchly *et al.* [230, 231] and depicted in figure 7.1.

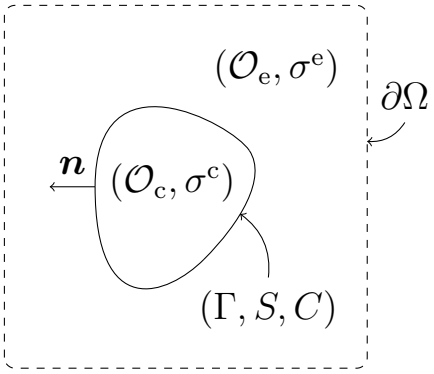


Figure 7.1: Geometry of the problem. The cell $\mathcal{O}_\mathrm{c}$ is imbedded in the extra-cellular matrix $\mathcal{O}_\mathrm{e}$. The entire domain Ω is defined by $\Omega = \mathcal{O}_\mathrm{e} \cup \overline{\mathcal{O}_\mathrm{c}}$.

Denoting by $\mathcal{O}_\mathrm{c}$ the cell cytoplasm and by $\mathcal{O}_\mathrm{e}$ the extracellular medium and defining by σ the conductivity of the medium as:

$$\sigma = \begin{cases} \sigma^\mathrm{e}, & \text{in } \mathcal{O}_\mathrm{e}, \\ \sigma^\mathrm{c}, & \text{in } \mathcal{O}_\mathrm{c}, \end{cases}$$

the electric potential satisfies the following boundary value problem:

$$u(0,\cdot) = 0 \quad \text{in } \mathcal{O}_\mathrm{e} \cup \mathcal{O}_\mathrm{c}, \tag{7.1a}$$

and for any $t > 0$,

$$\Delta u = 0, \quad \text{in } (0,T) \times (\mathcal{O}_\mathrm{e} \cup \mathcal{O}_\mathrm{c}), \tag{7.1b}$$

$$u(t,\cdot) = g(t,\cdot) \quad \text{on } (0,+\infty) \times \partial\Omega, \tag{7.1c}$$

with the jump conditions:

$$[\sigma \partial_{\mathbf{n}} u] = 0, \quad \text{on } (0,T) \times \Gamma, \tag{7.1d}$$

$$C\partial_t [u](t,\cdot) + S(t,[u])[u] = \sigma^{\mathrm{c}} \partial_{\mathbf{n}} u(t,\cdot)|_{\Gamma}, \quad \text{on } (0,T) \times \Gamma. \tag{7.1e}$$

The jump condition (7.1d) enforces the continuity of the flux and the transmission condition on the jump of the potential (7.1e) captures the influence of the thin resistive membrane.

We note that it is necessary to discretize the flux $\sigma^{\mathrm{c}}\ \partial_{\mathbf{n}} u|_{\Gamma}$ in (7.1e) implicitly to avoid the drastic CFL condition, as previously observed by Guyomarc'h *et al.* [170]. In [227], Poignard and colleagues proposed a second-order accurate finite volume method based on Cartesian grids to solve the equations in (7.1). However, the interface treatment they propose leads to wide stencils and large linear systems so that their approach cannot be readily applied to the simulation of a cluster with a large number of cells. The aim of this paper is to provide an efficient numerical method that makes it possible to solve the electric potential in a many-cell system. Such a numerical method is of great importance since it can help in understanding the macroscopic behavior of the potential in cell aggregates and therefore could lead the way to provide a numerical tool to compare micro- and meso-scale phenomena.

7.2.1 Electropermeabilization model

Electropermeabilization modeling consists in deriving a non-linear law for the surface membrane conductance S, or equivalently to add an electropermeabilization current in the Kirchhoff's law (7.1e). Generally speaking, these models describe the membrane

conductance as follows:

$$S(t, \lambda) := S_{\mathrm{L}} + S_{\mathrm{ep}}(t, \lambda), \tag{7.1f}$$

where S_{L} is the linear surface conductance of the membrane in the resting state and S_{ep} is the non-linear conductance due to the high transmembrane voltage. In this paper, we focus on the LMSP electropermeabilization model derived by Poignard and colleagues in [225], although our numerical method can also be used for the standard model of Krassowska and colleagues [221].

The LMSP model

Leguèbe, Poignard *et al.* have recently proposed in [225] a new phenomenological model, which discriminates the electroporated state of the membrane and the long-lasting permeabilized state. The electroporated state of the membrane is the highly conducting state of the membrane during the pulse delivery, while the long-lasting permeabilized state is persistent after the pulse. The model reads then as:

$$S(t, \lambda) = S_{\mathrm{L}} + S_0 X_0(t, \lambda) + S_1 X_1(t, X_0(t, \lambda)), \tag{7.2}$$

where S_{L}, S_0 and S_1 are the surface conductance of the membrane in the respective resting state, porated state and permeabilized state, and X_0 and X_1 are the degree of

poration and permeabilization respectively, which are solutions of

$$\begin{cases} \dfrac{\partial X_0(t,\lambda)}{\partial t} = \dfrac{\beta_0(\lambda(t)) - X_0}{\tau_{\text{ep}}}, \\ X_0(0,\lambda) = 0, \end{cases} \tag{7.3}$$

and

$$\begin{cases} \dfrac{\partial X_1(t,X_0)}{\partial t} = \max\left(\dfrac{\beta_1(X_0) - X_1}{\tau_{\text{perm}}}, \dfrac{\beta_1(X_0) - X_1}{\tau_{\text{res}}}\right), \\ X_1(0,\lambda) = 0, \end{cases} \tag{7.4}$$

where β_0 and β_1 are even-regularized step functions

$$\text{for all } \lambda \in \mathbb{R}, \qquad \beta_0(\lambda) := e^{-V_{\text{ep}}^2/\lambda^2}, \tag{7.5}$$

$$\text{for all } X \in \mathbb{R}, \qquad \beta_1(X) := e^{-X_{\text{ep}}^2/X^2}, \tag{7.6}$$

with V_{ep} and X_{ep} are given and correspond respectively to the threshold of the membrane voltage and of the degree of poration. The coefficients τ_{ep}, τ_{perm} and τ_{res} are the poration characteristic time, the permeabilization characteristic time and the resealing characteristic time, respectively.

7.3 Description of the computational method

7.3.1 Representation of irregular interfaces on Quad-/Oc-trees

The irregular interface (i.e. the cell's boundary), denoted by Γ, is implicitly captured as the zero level of the level-set function ϕ as first suggested by [4]. The extracellular domain $\mathcal{O}_{\text{e}}$ corresponds $\phi > 0$ and the cell cytoplasm $\mathcal{O}_{\text{c}}$ corresponds to $\phi < 0$.

In the case of the electropermeabilization model, the region where the solution under-

goes rapid variations is focused along the cell membrane, where the discontinuities are located. It is therefore desirable to focus the computational effort along the membrane, and we choose to work with adaptive Cartesian grid of type Quad-/Oc-trees [137, 138]. A Quadtree is constructed by starting from a computational cell representing the entire domain and splitting it in 4 (respectively 8 if working with Octrees in three spatial dimensions) recursively. We use the following refinement criteria, proposed by Strain [11] and extended by Min [232]: split a cell C if

$$\min_{v \in \text{vertices(C)}} |\phi(v)| \leq \texttt{Lip}(\phi) \cdot \texttt{diag}(C)$$

is satisfied. Here $\texttt{Lip}(\phi)$ is the Lipschitz constant associated to the level-set function ϕ, and $\texttt{diag}(C)$ is the length of the diagonal of C. The process is illustrated in figure 7.2. In practice, we choose ϕ to be the signed distance function to the irregular interface Γ, and set $\texttt{Lip}(\phi)$ conservatively to 1.2. A tree is characterized by its minimum and maximum level where a level is the depth in the tree. Thus a Quadtree of level 4/6 has a coarsest resolution equivalent to a uniform grid with 16^2 cells and a finest resolution equivalent to a uniform grid with 64^2 cells.

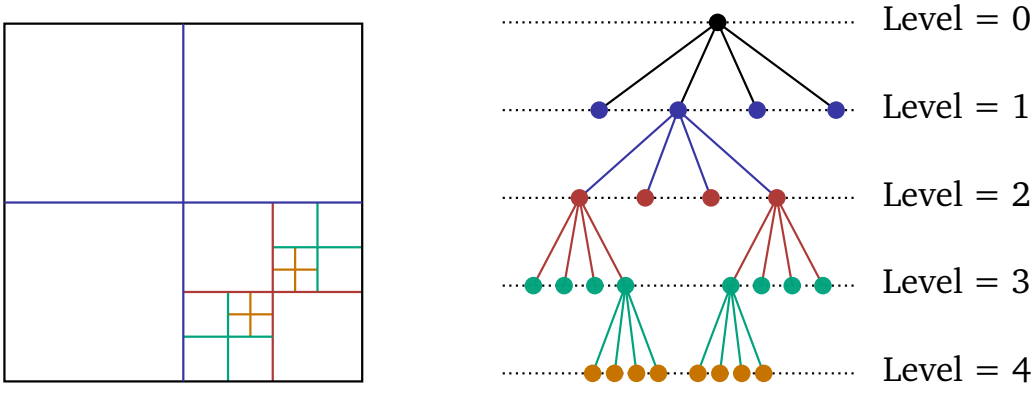


Figure 7.2: Illustration of a Quadtree mesh and its associated data structure.

7.3.2 The Voronoi Interface Method for electropermeabilization

The Voronoi Interface Method (VIM) introduced in [3] is designed to solve elliptic problems with discontinuities on irregular interfaces and with second-order accuracy. It consists in defining a Voronoi mesh based on an underlying Cartesian grid and solving the discontinuous elliptic problem on that new mesh. We now present an extension of the methodology for the electropermeabilization model where the discontinuity in the flux is non-trivial.

The first step consists in defining new degrees of freedom near the interface, placed along the normal to the interface, and building a corresponding Voronoi mesh. We define a Voronoi mesh as the collection of Voronoi cells such that each Voronoi cell is associated to a degree of freedom and defines the area of the computational domain that is closer to that degree of freedom than to any other. The degrees of freedom are the centers of the Quadtree cells except when the interface crosses a Quadtree cell, in which case the cell center is replaced by a pair of points on either side of the projection of the cell's center onto the interface, as illustrated in figure 7.3. The procedure enforces a minimum distance between the new degrees of freedom, as explained in [3].

We now proceed to present the numerical scheme and refer to the nomenclature presented in figure 7.4. In order to lighten the notations, for any function f defined in Ω, we denote by f^{e} (*resp.* f^{c}) the restriction of f to $\mathcal{O}_{\mathrm{e}}$ (*resp.* to $\mathcal{O}_{\mathrm{c}}$). We generically denote by f_i, f_j and f_p respectively the values of f at the point x_i, x_j and x_p on a Voronoi mesh.

We present the derivation for $\phi_i > 0$ and $\phi_j < 0$ and start by discretizing the time evolution of the membrane voltage, $[u]$, given by equation (7.1e) by a standard Backward Euler scheme:

$$C\frac{[u]^{n+1} - [u]^n}{\Delta t} + S^n[u]^{n+1} = (\sigma \partial_n u^{n+1})_{|\Gamma}$$

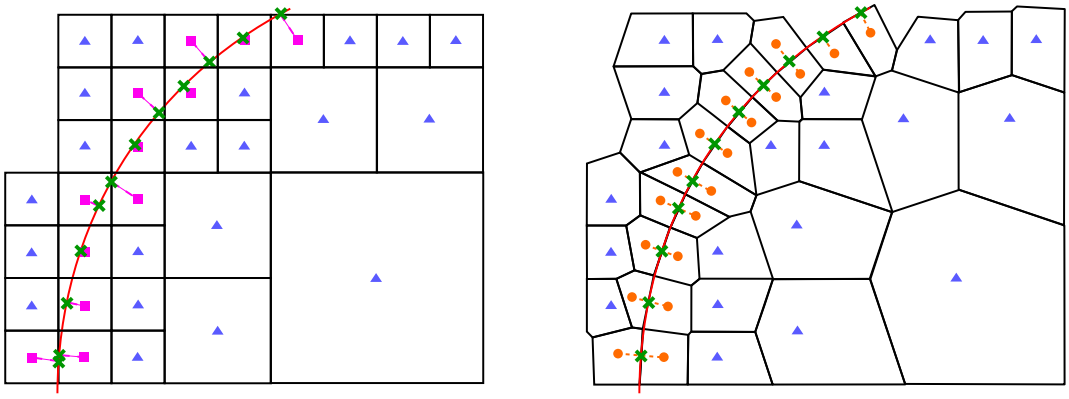


Figure 7.3: Illustration of the process to build a Voronoi mesh from a Quadtree grid. The cells crossed by the interface yield two new degrees of freedom on either side of the interface, avoiding the creation of points that are too close to each other. The left figure shows the original Quadtree Cartesian grid and the right figure depicts the corresponding Voronoi mesh.

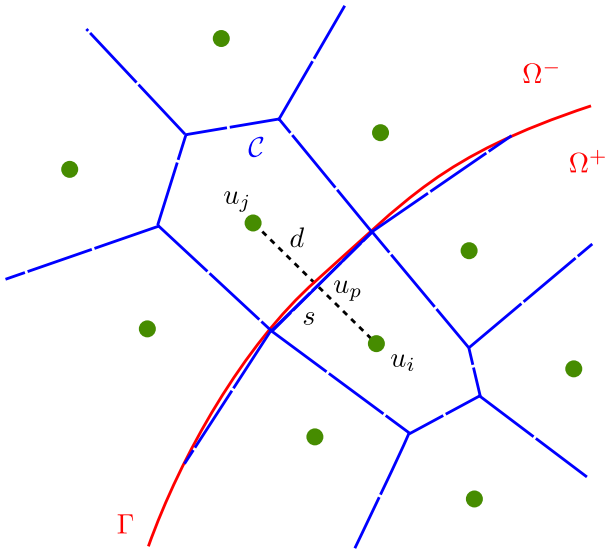


Figure 7.4: Nomenclature for the discretization on a Voronoi mesh. p is the projection of i on the interface Γ. It is also the projection of j on Γ and the halfway point between i and j. Furthermore, s is the length of the edge connecting the Voronoi cell of point i and the Voronoi cell of point j, and d is the distance between i and j.

from which we infer:

$$[u]^{n+1} = \frac{C[u]^n + \Delta t \sigma \partial_n u^{n+1}}{C + \Delta t S^n}. \tag{7.7}$$

We then use the continuity of the flux across the interface Γ at any time given by (7.1d) to write:

$$\sigma^{\mathrm{e}} \frac{u_p^{\mathrm{e}} - u_i^{\mathrm{e}}}{d/2} = \sigma^{\mathrm{c}} \frac{u_j^{\mathrm{c}} - u_p^{\mathrm{c}}}{d/2}.$$

The discontinuity condition at the interface $u_p^{\mathrm{c}} = u_p^{\mathrm{e}} - [u]^{n+1}$ and the expression of the flux across the interface $\sigma \partial_n u^{n+1} = \sigma^{\mathrm{e}} \frac{u_i^{\mathrm{e}} - u_p^{\mathrm{e}}}{d/2}$ leads to:

$$\frac{\sigma^{\mathrm{e}}}{d/2} \left(u_p^{\mathrm{e}} - u_i^{\mathrm{e}} \right) = \frac{\sigma^{\mathrm{c}}}{d/2} \left(u_j^{\mathrm{c}} - u_p^{\mathrm{e}} + [u]^{n+1} \right)$$

hence using (7.7) the equality

$$\sigma^{\mathrm{e}} \left(u_p^{\mathrm{e}} - u_i^{\mathrm{e}} \right) = \sigma^{\mathrm{c}} \left(u_j^{\mathrm{c}} - u_p^{\mathrm{e}} + \frac{C[u]^n + \Delta t \sigma \partial_n u^{n+1}}{C + \Delta t S^n} \right)$$

and thus

$$\sigma^{\mathrm{e}} \left(u_p^{\mathrm{e}} - u_i^{\mathrm{e}} \right) = \sigma^{\mathrm{c}} \left(u_j^{\mathrm{c}} - u_p^{\mathrm{e}} + \frac{C[u]^n}{C + \Delta t S^n} + \frac{\Delta t}{C + \Delta t S^n} \sigma^{\mathrm{e}} \frac{u_i^{\mathrm{e}} - u_p^{\mathrm{e}}}{d/2} \right),$$

which leads to the expression of u_p^{e}:

$$u_p^{\mathrm{e}} = \left(\sigma^{\mathrm{e}} u_i^{\mathrm{e}} + \sigma^{\mathrm{c}} u_j^{\mathrm{c}} + \frac{\sigma^{\mathrm{c}} C[u]^n}{C + \Delta t S^n} + \frac{\sigma^{\mathrm{e}} \sigma^{\mathrm{c}} \Delta t}{(C + \Delta t S^n) d/2} u_i^{\mathrm{e}} \right) / \left(\sigma^{\mathrm{e}} + \sigma^{\mathrm{c}} + \frac{\sigma^{\mathrm{e}} \sigma^{\mathrm{c}} \Delta t}{(C + \Delta t S^n) d/2} \right).$$

We can then use u_p^{e} into the discretization for the linear system. More precisely, from the equality

$$\sigma^{\mathrm{e}} s \frac{u_p^{\mathrm{e}} - u_i^{\mathrm{e}}}{d/2} = 0,$$

we get the following:

$$s\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}+\sigma^{\mathrm{c}}+\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}\Delta t}{(C+\Delta tS^n)d/2}}\frac{u_j^{\mathrm{c}}-u_i^{\mathrm{e}}}{d/2}=-s\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}+\sigma^{\mathrm{c}}+\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}\Delta t}{(C+\Delta tS^n)d/2}}\frac{C[u]^n}{(C+\Delta tS^n)d/2}.$$

Similarly, we obtain the final expression for the case when $\phi_i < 0$ and $\phi_j > 0$:

$$s\hat{\sigma}\frac{u_j-u_i}{d/2}=\mathrm{sgn}(\phi_i)s\hat{\sigma}\frac{C[u]^n}{(C+\Delta tS^n)d/2},$$

with

$$\hat{\sigma}=\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}+\sigma^{\mathrm{c}}+\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}\Delta t}{(C+\Delta tS^n)d/2}}.$$

If we use a second-order Backward Differentiation Formula for the time discretization, the above expression becomes

$$s\tilde{\sigma}\frac{u_j-u_i}{d/2}=\mathrm{sgn}(\phi_i)s\tilde{\sigma}\frac{C\left(2[u]^n-\frac{1}{2}[u]^{n-1}\right)}{(\frac{3}{2}C+\Delta tS^n)d/2},$$

with

$$\tilde{\sigma}=\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}+\sigma^{\mathrm{c}}+\frac{\sigma^{\mathrm{e}}\sigma^{\mathrm{c}}\Delta t}{(\frac{3}{2}C+\Delta tS^n)d/2}}.$$

The points far from the interface are treated with a classical finite volume discretization on the Voronoi partition. Since all the coefficients involved in $\hat{\sigma}$ are positive and the discontinuity $[u]^n$ appears only in the right hand side, this produces a symmetric positive definite linear system.

We next present numerical results that show the accuracy of our method in section 7.4. We then study numerically the permeabilization of cell aggregates in 3D configurations in section 7.5.

7.4 Numerical results

Throughout this section, the parameters described in table 7.1 are used.

Variable	Symbol	Value	Unit
Extracellular conductivity	σ^{e}	15	S/m
Intracellular conductivity	σ^{c}	1	S/m
Capacitance	C	$9.5 \cdot 10^{-3}$	F/m^2
Membrane surface conductivity	S_L	1.9	S/m^2
Cell radius	R_1	50	μm
Voltage threshold for poration	V_{ep}	$258 \cdot 10^{-3}$	V
Threshold for poration degree	X_{ep}	0.5	-
Poration characteristic time	τ_{ep}	10^{-6}	s
Permeabilization characteristic time	τ_{perm}	10^{-6}	s
Resealing characteristic time	τ_{res}	60	s
Porated membrane conductance	S_0	$1.1 \cdot 10^6$	S/m^2
Permeabilized membrane conductance	S_1	10^4	S/m^2

Table 7.1: Physical and computational parameters used for the simulations.

7.4.1 Solution to the static linear problem in two spatial dimensions

Consider the static linear problem in the case where the cell is a disk of radius R_1, embedded in a concentric bath of radius R_2. Assume that the electric potential at R_2 is $\frac{1}{2} g \cos\theta$, with $g = R_2 E$. In practice, we choose $E = 40\mathrm{kV/m}$ and $R_2 = 0.6\mathrm{mm}$. Then the exact solution of the static problem is explicitly given by:

$$u^{\mathrm{e}} = (\alpha^{\mathrm{e}} r + \frac{\beta^{\mathrm{e}}}{r}) \cos(\theta), \tag{7.8a}$$

$$u^{\mathrm{c}} = \alpha^{\mathrm{c}} r \cos(\theta), \tag{7.8b}$$

where the coefficients α^{c}, α^{e} and β^{e} are given by

$$\alpha^{\mathrm{c}} = \left(\left(\frac{\sigma^{\mathrm{c}}}{S_L R_1} + 1 + \frac{\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}} \right) R_2 + \left(\frac{\sigma^{\mathrm{c}}}{S_L R_1} + 1 - \frac{\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}} \right) \frac{R_1^2}{R_2} \right)^{-1} g, \tag{7.8c}$$

$$\alpha^{\mathrm{e}} = \frac{1}{2} \left(\frac{\sigma^{\mathrm{c}}}{S_L R_1} + 1 + \frac{\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}} \right) \alpha^{\mathrm{c}}, \tag{7.8d}$$

$$\beta^{\mathrm{e}} = \frac{1}{2} \left(\frac{\sigma^{\mathrm{c}}}{S_L R_1} + 1 - \frac{\sigma^{\mathrm{c}}}{\sigma^{\mathrm{e}}} \right) \alpha^{\mathrm{c}} R_1^2, \tag{7.8e}$$

from which we infer the static membrane voltage:

$$[u] = \frac{\sigma^{\mathrm{c}}}{S_L} \alpha^{\mathrm{c}} \cos(\theta). \tag{7.9}$$

Figure 7.5 presents a visualization of the computed electric potential and the computed membrane voltage. We monitor the convergence of the solver on the membrane voltage and report the results in table 7.2. The computed error corresponds to the largest error over the membrane between the exact membrane potential and the computed membrane potential. The convergence results indicate that second-order accuracy is achieved.

level (min/max)	L^∞ error on u	order	L^∞ error on $[u]$	order
3/5	$3.76 \cdot 10^{-6}$	-	$3.95 \cdot 10^{-6}$	-
4/6	$1.98 \cdot 10^{-6}$	0.93	$1.98 \cdot 10^{-6}$	1.00
5/7	$7.84 \cdot 10^{-7}$	1.34	$7.75 \cdot 10^{-7}$	1.35
6/8	$2.56 \cdot 10^{-7}$	1.62	$2.52 \cdot 10^{-7}$	1.62
7/9	$7.33 \cdot 10^{-8}$	1.80	$7.20 \cdot 10^{-8}$	1.81
8/10	$1.98 \cdot 10^{-8}$	1.89	$1.91 \cdot 10^{-8}$	1.91
9/11	$5.35 \cdot 10^{-9}$	1.89	$5.11 \cdot 10^{-9}$	1.90

Table 7.2: Convergence of the solver for the static case of subsection7.4.1.

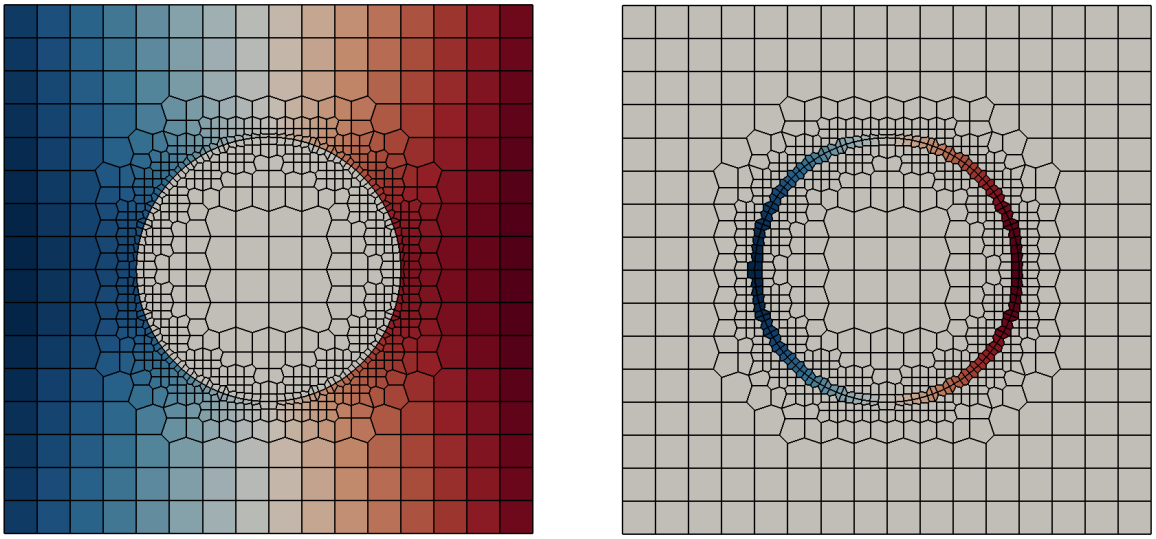


Figure 7.5: Representation of the numerical potential (left) and of the membrane voltage (right). These functions are approximations of the potential u and the membrane voltage $[u]$ given by equations (7.8) and (7.9), respectively.

7.4.2 Dynamic solution in two spatial dimensions

We now study the convergence of our method for the dynamic solution to the time-dependent linear problem in the two spatial dimensions case of a circular single cell. The geometrical framework is similar to that of section 7.4.1. We focus on the membrane voltage, which is the biophysically relevant quantity. Applying a time-dependent Dirichlet condition $(t, \theta) :\to g(t)\cos\theta$ on the outer boundary, we infer that the membrane voltage satisfies the ordinary differential equation

$$\partial_t[u] + \frac{S_L - B}{C}[u] = \frac{A}{C} g\cos(\theta),$$

with $A = -2\sigma^{\mathrm{c}} R_2 K$ and $B = \sigma^{\mathrm{c}} \frac{R_1^2+R_2^2}{R_1} K$, where

$$K = \frac{\sigma^{\mathrm{e}}}{R_1^2(\sigma^{\mathrm{c}} - \sigma^{\mathrm{e}}) - R_2^2(\sigma^{\mathrm{e}} - \sigma^{\mathrm{c}})}.$$

Given that the initial discontinuity is $[u]_{t=0} = 0$, we obtain:

$$[u](t,\theta) = \frac{A}{S_L - B} g \cos(\theta) \left(1 - e^{-\frac{S_L - B}{C} t}\right).$$

The solution and the membrane potential are represented in figure 7.6. We choose the time step as $\Delta t = \Delta x_{min}/40$ and solve the problem until the final time $t_f = 1\mu s$. The convergence of the solver on the membrane potential $[u]$ is presented in table 7.3 and figures 7.6 and 7.7. The error is computed as the maximum error over the entire membrane between the calculated membrane potential $[u]$ and the expected exact value. The results indicate an order of accuracy of about 1.5.

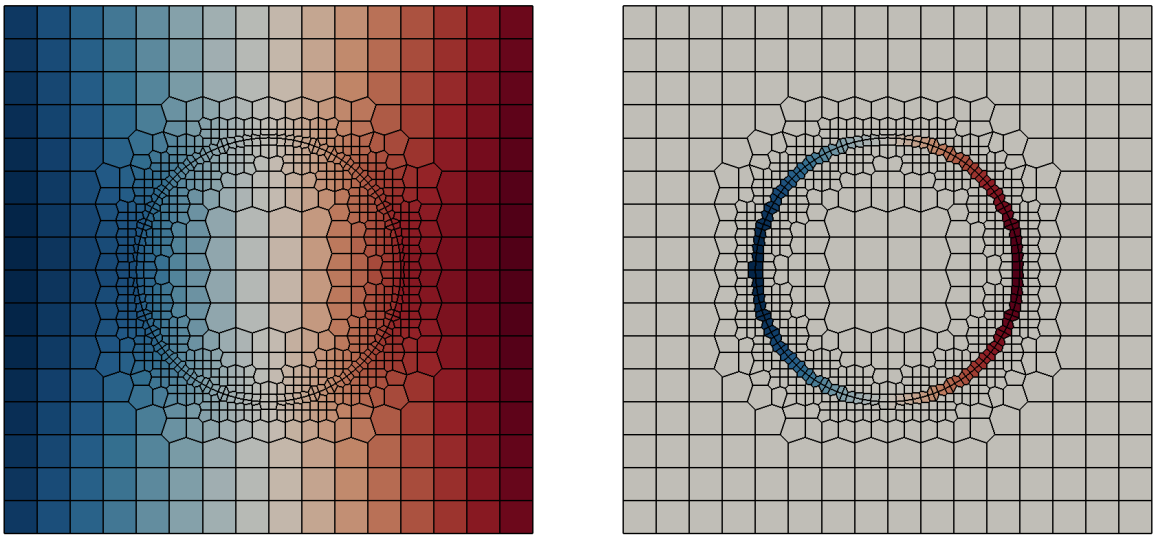


Figure 7.6: Representation of the electric potential and the transmembrane voltage given by (7.1), in the bidimensional circular framework of section 7.4.1.

	first order time		second-order time	
level (min/max)	error (L^∞)	order	error (L^∞)	order
3/5	$2.58 \cdot 10^{-5}$	-	$3.29 \cdot 10^{-6}$	-
4/6	$1.26 \cdot 10^{-5}$	1.03	$2.75 \cdot 10^{-6}$	0.26
5/7	$5.91 \cdot 10^{-6}$	1.09	$1.18 \cdot 10^{-6}$	1.22
6/8	$2.76 \cdot 10^{-6}$	1.10	$3.98 \cdot 10^{-7}$	1.57
7/9	$1.30 \cdot 10^{-6}$	1.08	$1.19 \cdot 10^{-7}$	1.74
8/10	$6.30 \cdot 10^{-7}$	1.05	$3.91 \cdot 10^{-8}$	1.61
9/11	$3.09 \cdot 10^{-7}$	1.03	$1.33 \cdot 10^{-8}$	1.56

Table 7.3: Error on the membrane electric potential discontinuity $[u]$ in the dynamic case 7.4.2 after $t_f = 1\mu s$, with $E = 40\text{kV/m}$ and $\Delta t = \Delta x_{min}/40$.

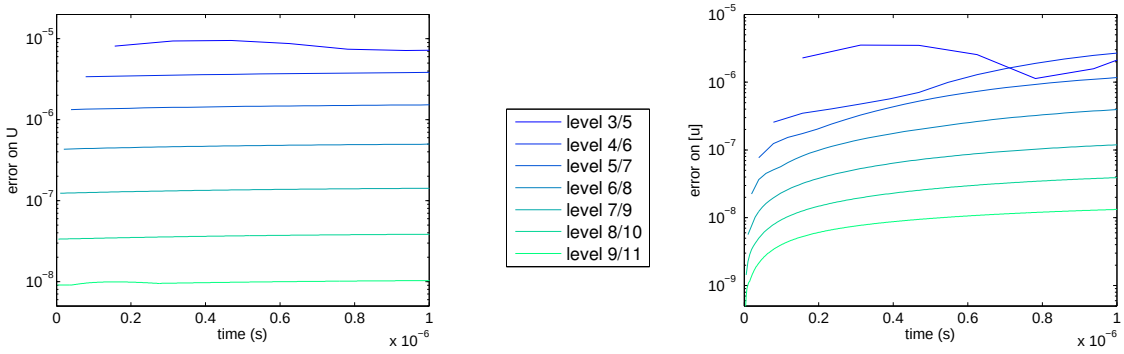


Figure 7.7: Visualization of the L^∞ error on the electric potential u (left) and on the membrane discontinuity $[u]$ (right) as a function of time for the dynamic analytical example 7.4.2, and for grid resolutions 3/5 (top curves) through 9/11 (bottom curves).

7.4.3 Validation with the static linear case in three spatial dimensions

In spherical coordinates, a possible form of the solution to the homogeneous Laplace equation that is independent of the azimuthal angle φ is

$$f(r,\theta,\varphi) = f(r,\theta) = \sum_{l=0}^{\infty} \left(A_l r^l + \frac{B_l}{r^{l+1}} \right) P_l(\cos(\theta)),$$

where P_l is the l^{th} Legendre polynomial and θ is the angle with the z-axis, in other words $\theta = \tan^{-1}(\sqrt{x^2+y^2}/z)$. For instance $P_1(x) = x$. We then choose:

$$U^{\text{e}} = (\alpha^{\text{e}} r + \frac{\beta^{\text{e}}}{r^2})\cos(\theta) \qquad \text{and} \qquad U^{\text{c}} = \alpha^{\text{c}} r \cos(\theta).$$

Matching the boundary conditions for an external potential $g = ER_2$ and a cell of radius R_1, from

$$\begin{cases} u^{\text{e}}(R_2) = g \\ [\sigma \partial_n u]_{|R_1} = 0 \\ [u]_{|R_1} = U^{\text{e}} - U^{\text{c}} \end{cases} \quad , \quad \text{we infer:} \quad \begin{cases} \alpha^{\text{e}} = R_2^2(\sigma^{\text{c}} + 2\sigma^{\text{e}})Kg - R_1^2\sigma^{\text{c}}K\frac{[u]}{\cos(\theta)} \\ \beta^{\text{e}} = R_1^3R_2^2(\sigma^{\text{e}} - \sigma^{\text{c}})Kg + R_1^2R_2^3\sigma^{\text{c}}K\frac{[u]}{\cos(\theta)} \\ \alpha^{\text{c}} = 3\sigma^{\text{e}}R_2^2Kg - \sigma^{\text{e}}(R_1^2 + 2\frac{R_2^3}{R_1})K\frac{[u]}{\cos(\theta)} \end{cases},$$

with $K^{-1} = R_1^3(\sigma^{\text{e}} - \sigma^{\text{c}}) + R_2^3(2\sigma^{\text{e}} + \sigma^{\text{c}})$. Since we are working with the static linear case (*i.e.* $S = S_L$), $[u]$ is given by:

$$[u] = \frac{A}{S_L - B}g, \quad \text{with} \quad A = 3\sigma^{\text{c}}\sigma^{\text{e}}R_2^2K \quad \text{and} \quad B = -\sigma^{\text{c}}\sigma^{\text{e}}(R_1^2 + 2\frac{R_2^3}{R_1})K. \qquad (7.10)$$

We monitor the convergence of the solver for this exact solution with $E = 10$ kV/m, $\Omega = [-10^{-4}, 10^{-4}]^3$, $R_1 = 5 \cdot 10^{-6}$ and $R_2 = 6 \cdot 10^{-4}$. The results are presented in table 7.4 and seem to indicate second order accuracy, though the asymptotic regime is not reached yet. Unfortunately, larger simulations are not practical with the current framework and would require a parallel environment, to which we plan to extend our method in the future.

	potential U		Membrane potential $[u]$	
level (min/max)	error (L^∞)	order	error (L^∞)	order
3/5	$4.14 \cdot 10^{-6}$	-	$5.71 \cdot 10^{-6}$	-
4/6	$3.72 \cdot 10^{-6}$	0.15	$3.80 \cdot 10^{-6}$	0.59
5/7	$2.01 \cdot 10^{-6}$	0.89	$1.87 \cdot 10^{-6}$	1.02
6/8	$7.49 \cdot 10^{-7}$	1.42	$7.14 \cdot 10^{-7}$	1.39
7/9	$2.43 \cdot 10^{-7}$	1.62	$2.32 \cdot 10^{-7}$	1.62

Table 7.4: Convergence of the solver for the linear static case in three spatial dimensions 7.4.3.

7.4.4 Validation with the dynamic linear case in three spatial dimensions

For this validation, we use the time-independent data g as in the previous section. However, $[u]$ now satisfies the dynamic equation given, for a constant $S = S_L$, as:

$$C\partial_t[u] + S_L[u] = \sigma^{\mathrm{c}}\partial_n U^{\mathrm{c}},$$

hence

$$\partial_t[u] + \frac{S_L - B}{C}[u] = \frac{A}{C} g \cos(\theta),$$

where A and B are given by equation (7.10). Given that the initial discontinuity is $[u]_{t=0} = 0$, we obtain:

$$[u](t,\theta) = \frac{A}{S_L - B} g\cos(\theta)\left(1 - e^{-\frac{S_L - B}{C}t}\right).$$

We monitor the convergence of the solver for this exact solution with $E = 40$ kV/m, $\Omega = [-10^{-4}, 10^{-4}]^3$, $R_1 = 5 \cdot 10^{-5}$, $R_2 = 6 \cdot 10^{-4}$, $t_f = 10^{-6}$s and $\Delta t = \Delta x_{min}/40$. The results are presented in table 7.5 and in figure 7.8.

	potential U		Membrane potential $[u]$	
level (min/max)	error (L^∞)	order	error (L^∞)	order
3/5	$4.16 \cdot 10^{-6}$	-	$9.99 \cdot 10^{-6}$	-
4/6	$3.71 \cdot 10^{-6}$	0.17	$5.63 \cdot 10^{-6}$	0.83
5/7	$1.99 \cdot 10^{-6}$	0.90	$2.46 \cdot 10^{-6}$	1.19
6/8	$7.41 \cdot 10^{-7}$	1.43	$8.24 \cdot 10^{-7}$	1.58
7/9	$3.39 \cdot 10^{-7}$	1.13	$6.11 \cdot 10^{-7}$	0.43

Table 7.5: Convergence of the solver for the linear dynamic case in three spatial dimensions 7.4.4 for $E = 40$kV/m and $\Delta t = \Delta x_{min}/40$.

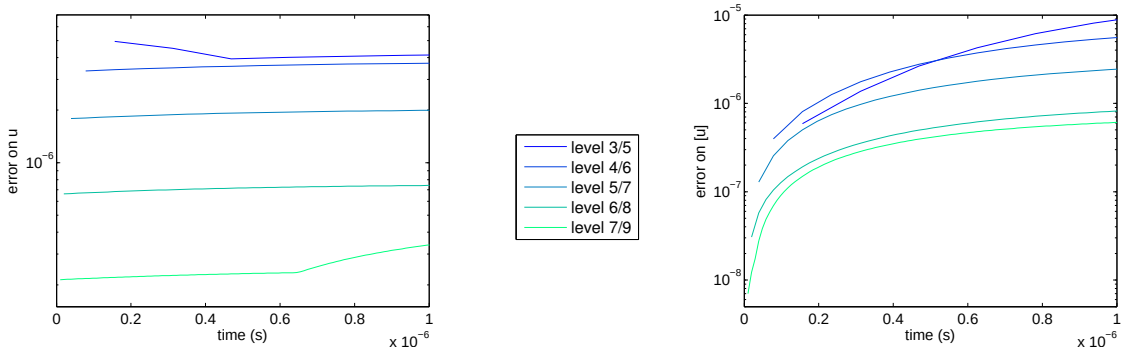


Figure 7.8: Visualization of the L^∞ error on the electric potential u (left) and on the membrane discontinuity $[u]$ (right) as a function of time for the dynamic analytical example 7.4.4, and for grid resolutions 3/5 through 7/9.

7.4.5 Convergence in time and space for the non-linear model in a single cell

We propose to monitor the convergence of the solver in time and space for the full non-linear model. We consider a spherical cell with radius $r_0 = 50\mu$m, centered in a box of length $4r_0$. We apply an electric field in the z-direction with intensity $E = 40$kV/m and compute the solution at a final time of $t_f = 1.5 \cdot 10^{-6}$s.

Starting with the convergence in space, we solve the problem for increasing spatial resolutions and with a fixed time step $\Delta t = 9.77 \cdot 10^{-9}$s. Starting with a mesh of resolution 3/6, i.e. with finest resolution equivalent to 64^3 cells, we increase the maximum resolution up to 9, i.e. equivalent to 512^3. The electric discontinuity $[u]$ at the pole of the cell is monitored in figure 7.9 and we observe convergence.

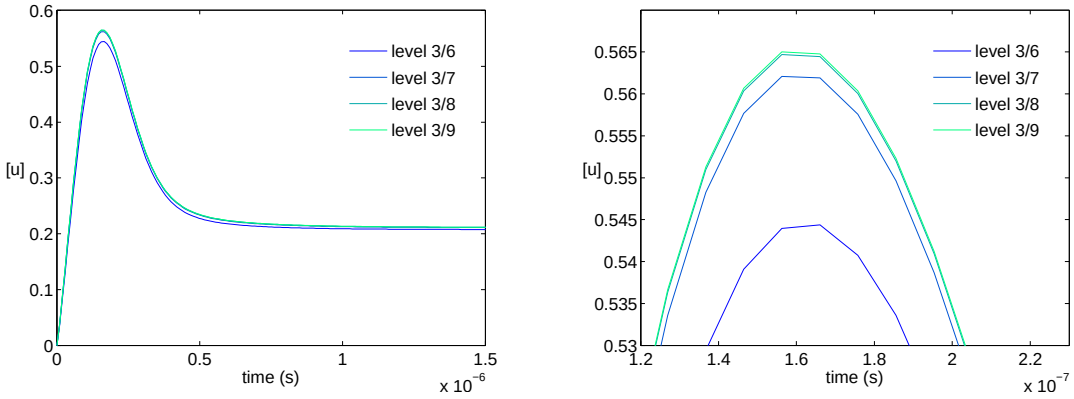


Figure 7.9: Convergence of the solver in space for the non-linear dynamic case with a single cell. The time step $\Delta t = 9.77 \cdot 10^{-9}$s is kept constant while the spatial resolution is increased. We observe convergence.

Next, we observe the convergence of the solver for a fixed space resolution of 3/7 as the time step is halved successively. Again, we monitor the electric discontinuity $[u]$ at the pole of the cell and the results are presented in figure 7.10 together with the finest resolution from the previous spatial convergence study. The solver converges, and

furthermore we observe similar accuracy with a resolution 3/7 than for the fine resolution 3/9 for the same time step.

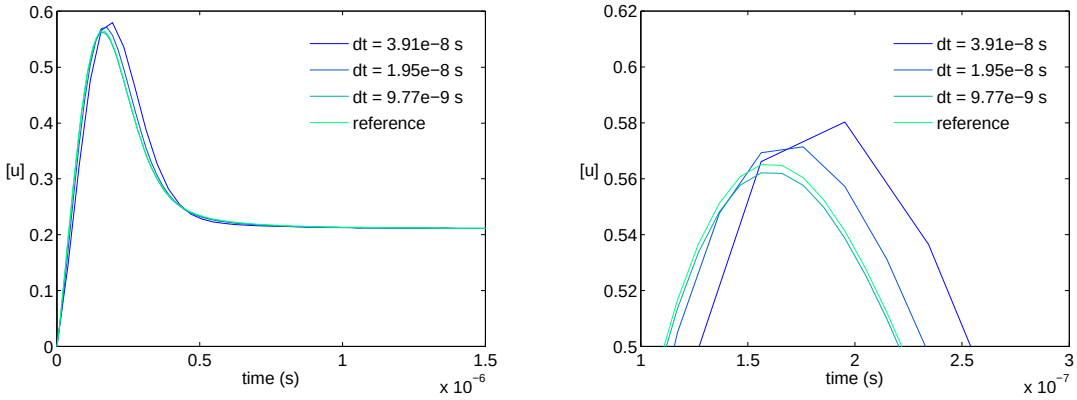


Figure 7.10: Convergence of the solver in time for the non-linear dynamic case with a single cell. The reference solution is computed on a mesh of level 3/9 and with $\Delta t = 9.77 \cdot 10^{-9} = \Delta x_{min}/40$. The other solutions are calculated on a mesh of level 3/7 for decreasing Δt ($\frac{\Delta x_{min}}{40}$, $\frac{\Delta x_{min}}{80}$ and $\frac{\Delta x_{min}}{160}$). The system's response converges.

These results are consistent with the model which is highly non-linear in time but well-behaved in space. Therefore, as long as the geometry is correctly resolved, the spatial accuracy is dominated by the temporal errors and a finer time step is more critical than a fine spatial resolution.

7.5 Computational study of the permeabilization of three dimensional cell arrays

The combination of the sharp treatment of the interfacial jump condition, the symmetry and definite positiveness of the linear system, with the adaptive Cartesian grid, enables the study of electroporation beyond a single cell. In what follows, we present electroporation simulations on arrays of cells and study the influence of cells' geometry

and the shadowing effects.

7.5.1 Shadowing effect

We study the behavior of the solver for a $3 \times 3 \times 3$ array of spherical cells with radii $r_0 = 5\mu$m, located periodically in an inner box of 1 cm^3. In order to ensure that the entire aggregate is embedded in an homogeneous electric field, we place it at the center of a 2 cm^3 computational domain as illustrated in the left part of figure 7.12. We take a time step $\Delta t = \frac{\Delta x_{min}}{400}$ and we impose an electric field $E = 40$kV/m in the z-direction. The Octree has a resolution 5/10, leading to a Voronoi mesh with 2,219,552 cells. The permeabilization of the cells averaged on each of the slices in the z-direction is presented in figure 7.11. We observe a shadowing effect on the middle slice, which presents a lower degree of permeabilization. The same setup with 5x5x5 cells exhibits a more pronounced shadowing, as depicted in figure in figure 7.12. The tree this time is level 5/9, leading to 2,614,488 Voronoi cells. Figure 7.13 summarizes this section by showing the average permeabilization of the entire cluster of cells for different densities of cells, from a single cell to a 5x5x5 array, demonstrating the effect of shadowing. Such results are consistent with the experimental observations of Pucihar *et al.* [233], in which dense cell suspensions increase shadowing effects. This example demonstrates the ability to study quantitatively shadowing effects with respect to the density of cell suspensions.

7.5.2 Influence of the shape

We propose to investigate the influence of the cells' geometry on the poration. We choose an array of N^3 cells spread evenly in a box of size 1cm^3 located in a computational domain of size 2cm^3 and apply a electric potential in the z-direction with magnitude 40kV. We select three shapes: spheres with radii 50μm, oblate ellipsoids with radii 46μm and

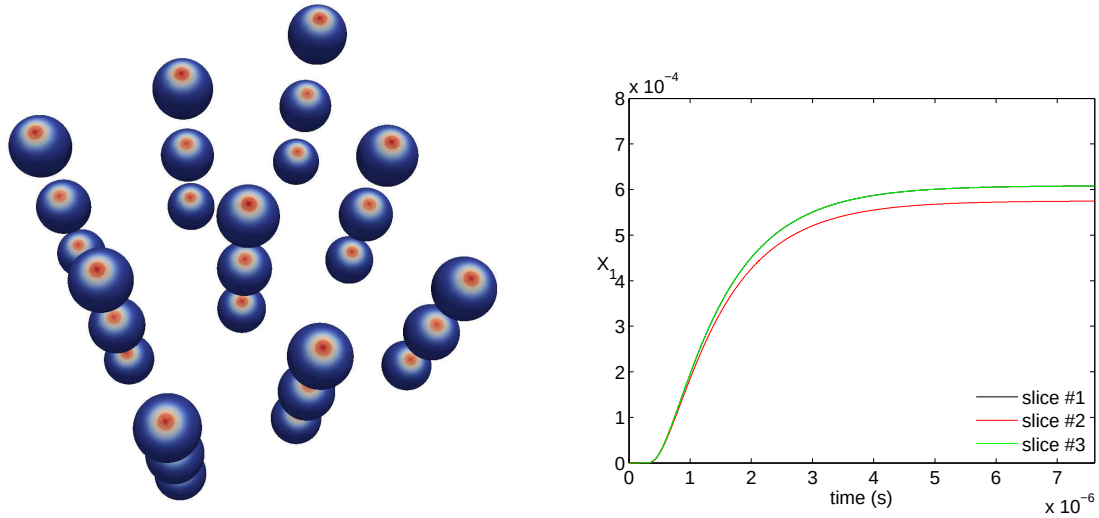


Figure 7.11: Left: Visualization of the permeabilization X_1 of a $3 \times 3 \times 3$ array of cells. Right: Evolution in time of the average permeabilization of the cell membranes for each z-slice of the $3 \times 3 \times 3$ array of cells. Note that due to symmetry the curves for the first and third slices are superposed.

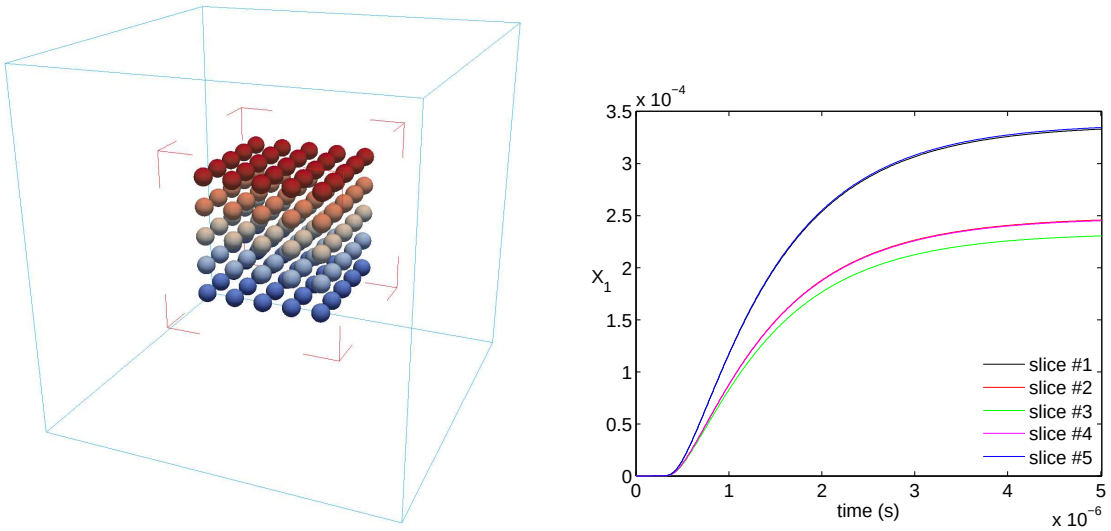


Figure 7.12: Left: setup for the simulation 7.5.1. The blue box represents the computational domain and the red corners mark the inner box in which the cells are located. The cells are colored with the electric potential u. Right: average permeabilization X_1 by z-slice as a function of time. The first and fifth slices are superposed due to symmetry, and so are the second and the third slices.

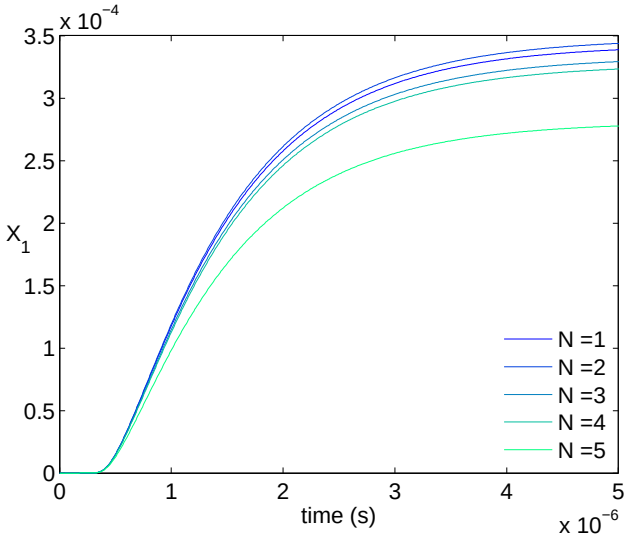


Figure 7.13: Influence of the cell packing on the average permeabilization X_1. A packing of N corresponds to a $N \times N \times N$ array of spherical cells. As N increases and the cells get closer, the shadowing effects become more pronounced and the permeabilization decreases.

prolate ellipsoids with radii 53μm. Here, an ellipsoid with radius r_0 is given by:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = r_0^2.$$

It is oblate for $a = b > c$ and prolate for $a = b < c$. The coefficients a, b and c are chosen such that the surface of the ellipsoidal cells is the same than that of the spherical cells. Figure 7.14 provides a visualization of the three shapes. The tree is level 5/9, leading to approximately 610,784 Voronoi cells for the spherical cells, 597,968 Voronoi cells for the oblate cells and 561,176 Voronoi cells for the prolate cells. The time step is set to $\Delta t = \frac{\Delta x_{min}}{400} \approx 9.77 \cdot 10^{-9}$s. The relevant physical parameters, i.e. the average conductance S, the average discontinuity in the electric potential $[u]$ at the tip of the cells, the average poration X_0 and the average permeabilization X_1, are represented in figures 7.15 and 7.16 for $N = 3$ and $N = 5$ respectively. We observe a strong influence of

the cell shapes, with orders of magnitude of difference in the case of the permeabilization X_1. These results are consistent with the biological experiments demonstrating that, in order to increase the efficacy of electroporation in muscles, the electric field has to be applied orthogonally to the fibers [234]. This computational example demonstrates that the approach described in this manuscript is capable of studying the influence of cells' geometries. In turn, this will enable the study of electroporation in more complex cell distributions, such as in brain tissue for example, and will help determine the optimal distribution of the electric field for maximizing cell electroporation.

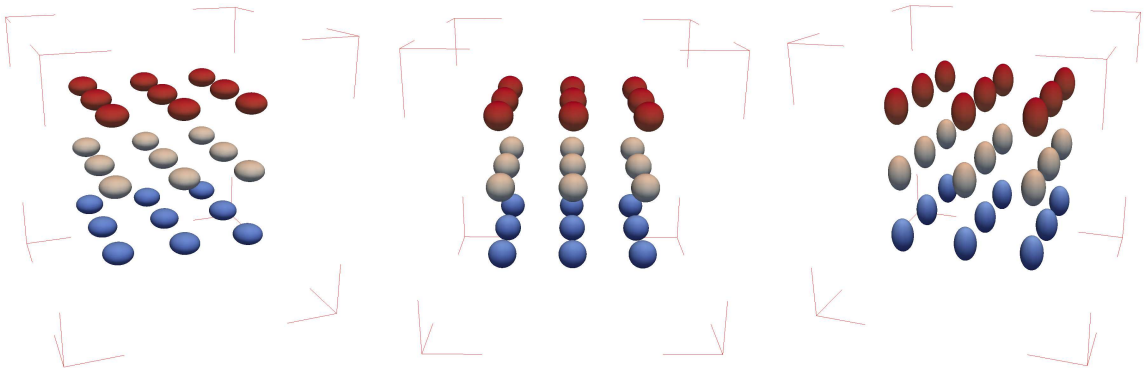


Figure 7.14: Visualization of an array of $3\times3\times3$ cells with oblate (left), spherical (center) and prolate (right) shapes colored with the electric potential u.

7.5.3 Random cluster with 100 cells

For this last example, we consider 100 cells located in a box of size 1cm^3 in a domain twice that size. We impose an electric field in the z-direction with intensity 40kV and choose $\Delta t = \frac{\Delta x_{min}}{400} \approx 9.77 \cdot 10^{-9}$s. The tree resolution is 5/9 and the cells are ellipsoids with random eccentricities, orientations and locations. The results are depicted in figure 7.17 and illustrate the capacity of our solver to handle complex layouts of numerous cells. This example is interesting for biological applications, for which cells do not have

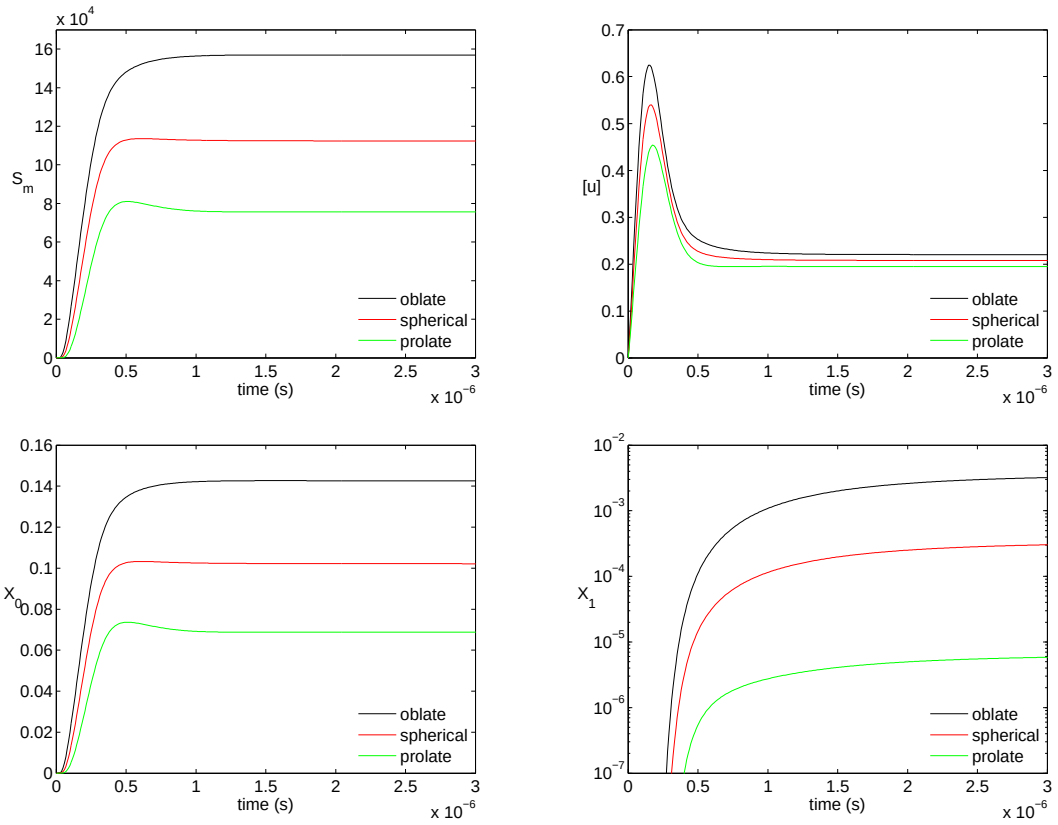


Figure 7.15: Representation of the relevant physical parameters as a function of time for the three different cell shapes decribed in section 7.5.2 and for an array of $3 \times 3 \times 3$ cells. Note that X_1 is represented with a logarithmic scale.

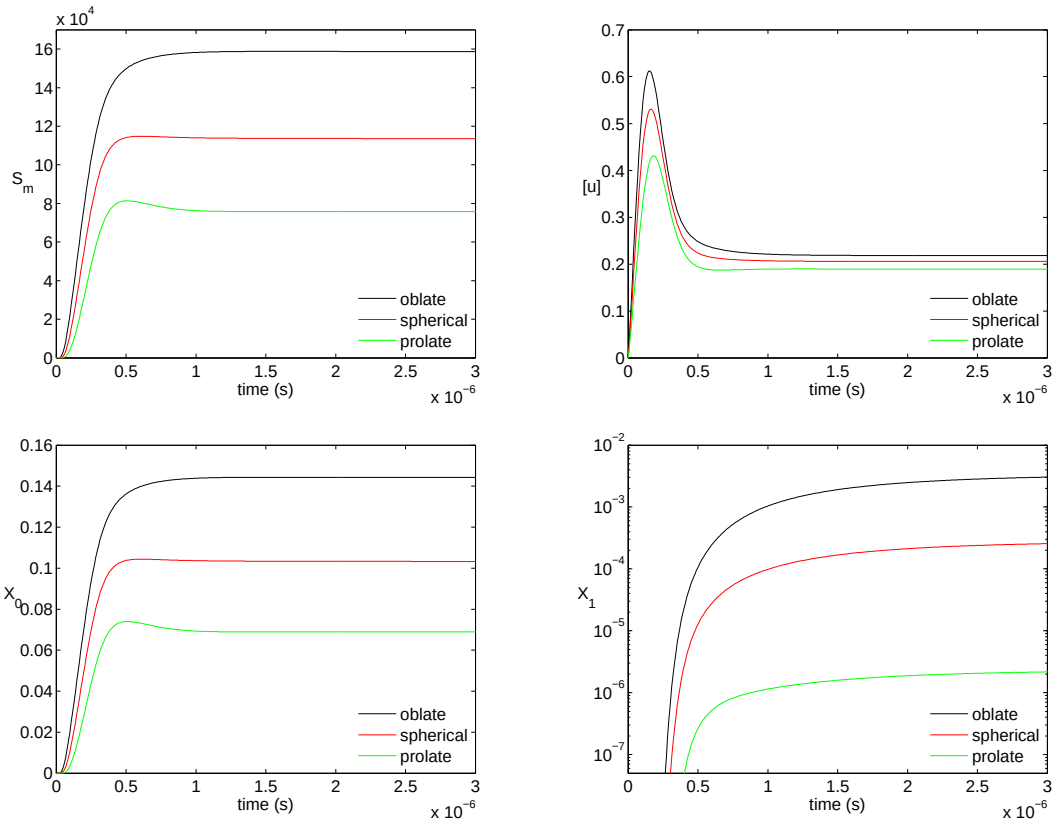


Figure 7.16: Representation of the relevant physical parameters as a function of time for three different cell shapes decribed in section 7.5.2 and for an array of $5 \times 5 \times 5$ cells. Note that X_1 is represented with a logarithmic scale.

exactly the same shape and volume. Yet, it is possible to determine the distribution of ellipsoidal shapes and diameters in a sample. Hence, our numerical method makes it possible to predict quantitatively the average degree of cell permeabilization as well as the distribution of permeabilized cells in such set up.

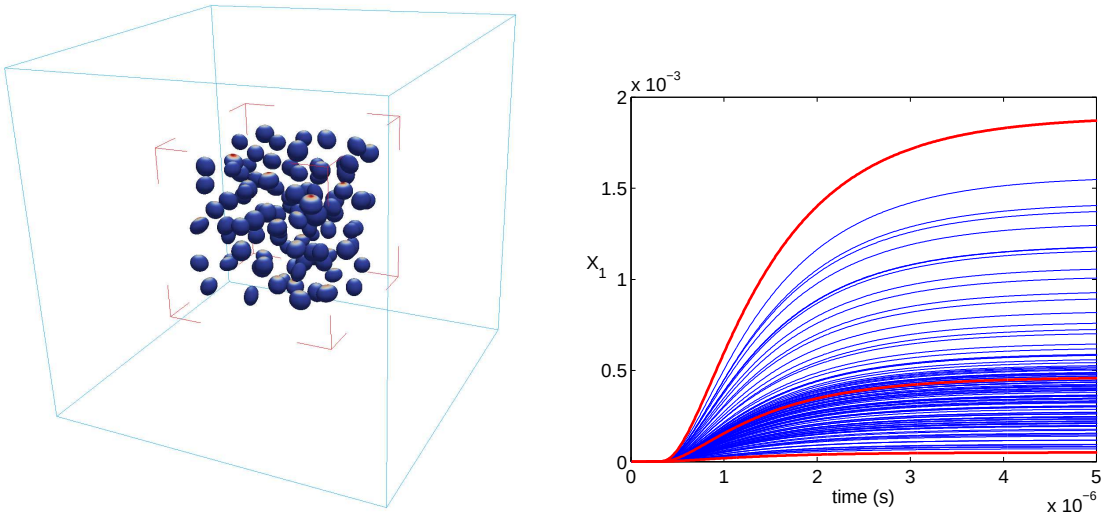


Figure 7.17: Representation of a cluster of 100 cells (left) and the corresponding permeabilization X_1 (right). The average, maximum and minimum levels of permeabilization are represented by the red lines.

7.6 Summary

We have presented a Voronoi Interface approach for the simulation of cell electropermeabilization. In particular, we have considered a nonlinear electropermeabilization model and imposed the jump condition in electrical potential in a sharp manner. The numerical treatment at the interface leads to a symmetric positive definite linear system that can be inverted efficiently. Together with the use of adaptive grids, this approach enables the study of cell aggregates. Computational experiments have illustrated the accuracy of the numerical approach and have been used to investigate the shadowing

effects as well as the influence of cell's geometries on the degree of permeabilization. We find that cells with elongated shapes are more prone to be electropermeabilized if the field is orthogonal to the long axis; this is consistent with the biological experiments of Corovic *et al.* in the context of muscles. Our work is a first-step towards studying electropermeabilization of mesoscale cell spheroids, which provide an interesting biological model of tumors. Future work will consider the extension to massively parallel architectures, which will provide a computational tool that makes it possible to compare with macroscale models obtained by either phenomenological considerations or by rigorous homogenization of a microscale single-cell model. Our approach can also serve as an advanced numerical tool that can enable the comparison between theoretical models and biological experiments of electropermeabilization of mesoscale spheroids.

Bibliography

[1] E. Brun, A. Guittet, and F. Gibou, *A local level-set method using a hash table data structure*, *J. Comp. Phys.* **231** (2012) 2528–2536.

[2] A. Guittet, M. Theillard, and F. Gibou, *A stable projection method for the incompressible navier-stokes equations on arbitrary geometries and adaptive quad/octrees*, *Journal of Computational Physics* (2015).

[3] A. Guittet, M. Lepilliez, S. Tanguy, and F. Gibou, *Solving elliptic problems with discontinuities on irregular domains - the Voronoi Interface Method*, *Journal of Computational Physics* **298** (2015) 747 – 765.

[4] S. Osher and J. Sethian, *Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations*, *Journal of Computational Physics* **79** (1988) 12–49.

[5] J. A. Sethian, *Level set methods and fast marching methods*. Cambridge University Press, 1999. Cambridge.

[6] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002. New York, NY.

[7] D. Adalsteinsson and J. Sethian, *A Fast Level Set Method for Propagating Interfaces*, *J. Comput. Phys.* **118** (1995) 269–277.

[8] M. B. Nielsen and K. Museth, *Dynamic Tubular Grid: An Efficient Data Structure and Algorithms for High Resolution Level Sets*, *Journal of Scientific Computing* **26** (Jan., 2006) 261–299.

[9] H. Samet, *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, New York, 1989.

[10] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing and GIS*. Addison-Wesley, New York, 1990.

[11] J. Strain, *Tree methods for moving interfaces*, *J. Comput. Phys.* **151** (1999) 616–648.

[12] S. Popinet, *Gerris: A tree-based adaptive solver for the incompressible euler equations in complex geometries*, *J. Comput. Phys.* **190** (2003) 572–600.

[13] F. Losasso, F. Gibou, and R. Fedkiw, *Simulating water and smoke with an octree data structure*, *ACM Trans. Graph. (SIGGRAPH Proc.)* (2004) 457–462.

[14] C. Min and F. Gibou, *A second order accurate level set method on non-graded adaptive Cartesian grids*, *J. Comput. Phys.* **225** (2007) 300–321.

[15] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, *et. al.*, *Collision detection for deformable objects*, in *Computer Graphics Forum*, vol. 24, pp. 61–81, Wiley Online Library, 2005.

[16] K. Steele, D. Cline, P. Egbert, and J. Dinerstein, *Modeling and rendering viscous liquids*, *Computer Animation and Virtual Worlds* **15** (2004), no. 3-4 183–192.

[17] T. Cormen, *Introduction to algorithms*. The MIT press, 2001.

[18] M. Sussman and E. Fatemi, *An efficient interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow*, *SIAM J. of Scientific Comput.* **20** (1999) 1165–1191.

[19] C.-W. Shu and S. Osher, *Efficient implementation of essentially non-oscillatory shock capturing schemes*, *J. Comput. Phys.* **77** (1988) 439–471.

[20] G. Russo and P. Smereka, *A remark on computing distance functions*, *J. Comput. Phys.* **163** (2000) 51–67.

[21] J. Sethian, *A fast marching level set method for monotonically advancing fronts*, *Proc. Natl. Acad. Sci.* **93** (1996) 1591–1595.

[22] J. Tsitsiklis, *Efficient Algorithms for Globally Optimal Trajectories*, *IEEE Trans. on Automatic Control* **40** (1995) 1528–1538.

[23] D. Xiu and G. Karniadakis, *A semi-Lagrangian high-order method for Navier-Stokes equations*, *J. Comput. Phys* **172** (2001) 658–684.

[24] C. Min and F. Gibou, *Geometric integration over irregular domains with application to level set methods*, *J. Comput. Phys.* **226** (2007) 1432–1443.

[25] J. B. Bell, P. Colella, and H. M. Glaz, *A second order projection method for the incompressible Navier-Stokes equations*, *J. Comput. Phys* **85** (1989) 257–283.

[26] E. Olsson and G. Kreiss, *A conservative level set method for two phase flow*, *J. Comput. Phys.* **210** (2005) 225–246.

[27] D. Juric and G. Tryggvason, *A front tracking method for dendritic solidification*, *J. Comput. Phys* **123** (1996) 127–148.

[28] G. Tryggvason, B. Bunner, A. Esmaeeli, D. Juric, N. Al-Rawahi, W. Tauber, J. Han, S. Nas, and Y.-J. Jan, *A front-tracking method for the computations of multiphase flow*, *J. Comput. Phys.* **169** (2001) 708–759.

[29] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell, *A hybrid particle level set method for improved interface capturing*, *J. Comput. Phys.* **183** (2002) 83–116.

[30] M. Theillard, F. Gibou, and T. Pollock, *A sharp computational method for the simulation of the solidification of binary alloys*, *J. Sci. Comput.* (2014).

[31] W. Bangerth, R. Hartmann, and G. Kanschat, *Deal.II - a general-purpose object-oriented finite element library*, *ACM Trans. Math. Software* **33** (2007).

[32] G. Karypis and V. Kumar, *METIS – Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, 1995.

[33] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, *Journal of Parallel Distributed Computing* **48** (Jan., 1998) 71–95.

[34] E. G. Boman, Ü. V. Çatalyürek, C. Chevalier, and K. D. Devine, *The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring*, *Scientific Programming* **20** (2012), no. 2 129–150.

[35] O. Sahni, M. Zhou, M. S. Shephard, and K. E. Jansen, *Scalable implicit finite element solver for massively parallel processing with demonstration to 160K cores*, in *SC09: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2009.

[36] J. R. Stewart and H. C. Edwards, *A framework approach for developing parallel adaptive multiphysics applications*, *Finite Elements in Analysis and Design* **40** (2004), no. 12 1599–1617.

[37] S. Aluru and F. E. Sevilgen, *Parallel domain decomposition and load balancing using space-filling curves*, in *Proceedings of the Fourth International Conference on High-Performance Computing*, HIPC '97, (Washington, DC, USA), pp. 230–, IEEE Computer Society, 1997.

[38] M. Griebel and G. W. Zumbusch, *Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves*, *Parallel Computing* **25** (1999) 827–843.

[39] P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco, *Dynamic octree load balancing using space-filling curves*, Tech. Rep. CS-03-01, Williams College Department of Computer Science, 2003.

[40] T. Tu, D. R. O'Hallaron, and O. Ghattas, *Scalable parallel octree meshing for terascale applications*, in *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, (Washington, DC, USA), pp. 4–, IEEE Computer Society, 2005.

[41] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L. C. Wilcox, and S. Zhong, *Scalable adaptive mantle convection simulation on petascale supercomputers*, in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, (Piscataway, NJ, USA), pp. 62:1–62:15, IEEE Press, 2008.

[42] R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, and G. Biros, *Dendro: Parallel algorithms for multigrid and amr methods on 2:1 balanced octrees*, in *International Conference for High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008.*, 2008.

[43] R. S. Sampath and G. Biros, *A parallel geometric multigrid method for finite elements on octree meshes*, *SIAM Journal on Scientific Computing* **32** (May, 2010) 1361–1392.

[44] C. Burstedde, L. C. Wilcox, and O. Ghattas, *`p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, *SIAM Journal on Scientific Computing* **33** (2011), no. 3 1103–1133.

[45] C. Burstedde, "`p4est`: Parallel Adaptive Mesh Refinement on Forests of Octrees." `http://www.p4est.org/`. Last accessed June 19, 2015.

[46] T. Isaac, C. Burstedde, L. C. Wilcox, and O. Ghattas, *Recursive algorithms for distributed forests of octrees*, *SIAM Journal on Scientific Computing* **37** (2015), no. 5 C497–C531.

[47] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas, *An extreme-scale implicit solver for complex PDEs: highly heterogeneous flow in earth's mantle*, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 5, ACM, 2015.

[48] A. Müller, M. A. Kopera, S. Marras, L. C. Wilcox, T. Isaac, and F. X. Giraldo, "Strong scaling for numerical weather prediction at petascale with the atmospheric model NUMA." http://arxiv.org/abs/1511.01561, 2015.

[49] F. Losasso, R. Fedkiw, and S. Osher, *Spatially adaptive techniques for level set methods and incompressible flow*, *Computers and Fluids* **35** (2006) 995–1010.

[50] S. Thomas and J. Côté, *Massively parallel semi-Lagrangian advection*, *Simulation Practice and Theory* **3** (1995), no. 4 223–238.

[51] J. Drake, I. Foster, J. Michalakes, B. Toonen, and P. Worley, *Design and performance of a scalable parallel community climate model*, *Parallel Computing* **21** (1995), no. 10 1571–1591.

[52] J. White III and J. J. Dongarra, *High-performance high-resolution semi-Lagrangian tracer transport on a sphere*, *Journal of Computational Physics* **230** (2011), no. 17 6778–6799.

[53] K. Wang, A. Chang, L. V. Kale, and J. A. Dantzig, *Parallelization of a level set method for simulating dendritic growth*, *Journal of Parallel and Distributed Computing* **66** (2006), no. 11 1379–1386.

[54] M. Sussman, *A parallelized, adaptive algorithm for multiphase flows in general geometries*, *Computers & structures* **83** (2005), no. 6 435–444.

[55] O. Fortmeier and H. M. Bücker, *A parallel strategy for a level set simulation of droplets moving in a liquid medium*, in *High Performance Computing for Computational Science–VECPAR 2010*, pp. 200–209. Springer, 2011.

[56] J. M. Rodriguez, O. Sahni, R. T. Lahey Jr, and K. E. Jansen, *A parallel adaptive mesh method for the numerical simulation of multiphase flows*, *Computers & Fluids* **87** (2013) 115–131.

[57] M. Herrmann, *A parallel Eulerian interface tracking/Lagrangian point particle multi-scale coupling procedure*, *Journal of Computational Physics* **229** (2010), no. 3 745–759.

[58] M. Sussman, P. Smereka, and S. Osher, *A level set approach for computing solutions to incompressible two-phase flow*, *J. Comput. Phys.* **114** (1994) 146–159.

[59] S. Osher and R. P. Fedkiw, *Level Set Methods: An Overview and Some Recent Results*, *Journal of Computational Physics* **169** (2001) 463–502.

[60] H. Zhao, *A fast sweeping method for eikonal equations*, *Mathematics of Computation* **74** (2004), no. 250 603–627.

[61] M. Herrmann, *A domain decomposition parallelization of the fast marching method*, tech. rep., DTIC Document, 2003.

[62] M. C. Tugurlan, *Fast marching methods-parallel implementation and analysis.* PhD thesis, Louisiana State University, 2008.

[63] M. Breuß, E. Cristiani, P. Gwosdek, and O. Vogel, *An Adaptive Domain-Decomposition Technique for Parallelization of the Fast Marching Method*, *Applied Mathematics and Computation* **218** (Sept., 2011) 32–44.

[64] H. Zhao, *Parallel implementations of the fast sweeping method*, *Journal of Computational Mathematics* **25** (2007) 421–429.

[65] M. Detrixhe, F. Gibou, and C. Min, *A Parallel Fast Sweeping Method for the Eikonal Equation*, *Journal of Computational Physics* **237** (Mar., 2013) 46–55.

[66] A. Chacon and A. Vladimirsky, *A parallel heap-cell method for eikonal equations*, *arXiv preprint arXiv:1306.4743* (2013).

[67] W. Jeong and R. Whitaker, *A Fast Iterative Method for Eikonal Equations*, *SIAM Journal on Scientific Computing* **30** (2008), no. 5 2512–2534.

[68] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler, *Algorithms and data structures for massively parallel generic adaptive finite element codes*, *ACM Transactions on Mathematical Software (TOMS)* **38** (2011), no. 2 14.

[69] T. Hoefler, C. Siebert, and A. Lumsdaine, *Scalable communication protocols for dynamic sparse data exchange*, *ACM SIGPLAN Notices* **45** (2010), no. 5 159–168.

[70] T. Isaac, C. Burstedde, and O. Ghattas, *Low-cost parallel algorithms for 2:1 octree balance*, in *Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pp. 426–437, 2012.

[71] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkens-Diehr, *Xsede: Accelerating scientific discovery*, *Computing in Science and Engineering* **16** (2014), no. 5 62–74.

[72] V. Alexiades, A. D. Solomon, and D. G. Wilson, *The formation of a solid nucleus in supercooled liquid. I, J, Non-Equilib. Thermodyn.* **13** (1988) 281–300.

[73] V. Alexiades and A. D. Solomon, *Mathematical Modeling of Melting and Freezing Processes*. Hemisphere, Washingotn, DC, 1993.

[74] H. Chen, C. Min, and F. Gibou, *A numerical scheme for the Stefan problem on adaptive Cartesian grids with supralinear convergence rate*, *J. Comput. Phys.* **228** (2009), no. 16 5803–5818.

[75] S. Balay, J. Brown, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *Petsc web page*, 2014.

[76] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg, 2002.

[77] J. W. Purvis and J. E. Burkhalter, *Prediction of critical mach number for store configurations*, *AIAA J.* **17** (1979) 1170–1177.

[78] R. Gaffney, H. Hassan, and M. Salas, *Euler calculations for wings using cartesian grids*, in *AIAA 25th Aerospace Sciences Meeting*, 1987.

[79] B. Grossman and D. Whitaker, *Supersonic flow computations using a rectangular-coordinate finite-volume method*, in *AIAA 24th Aerospace Sciences Meeting*, 1986.

[80] M. Vinokur, *On one-dimensional stretching functions for finite difference calculations*, *Journal of Computational Physics* **50** (1983), no. 2 215–234.

[81] E. J. Avital, N. D. Sandham, and K. H. Luo, *Stretched cartesian grids for solution of the incompressible navier-stokes equations*, *International Journal for Numerical Methods in Fluids* **33** (2000), no. 6 897–918.

[82] M. Berger and J. Oliger, *Adaptive mesh refinement for hyperbolic partial differential equations*, *J. Comput. Phys.* **53** (1984) 484–512.

[83] L. H. Howell and J. B. Bell, *An adaptive mesh projection method for viscous incompressible flow*, *SIAM Journal on Scientific Computing* **18** (1997), no. 4 996–1013, [http://dx.doi.org/10.1137/S1064827594270555].

[84] M. Sussman, A. Almgren, J. Bell, P. Colella, L. Howell, and M. Welcome, *An adaptive level set approach for incompressible two-phase flows*, *J. Comput. Phys.* **148** (1999) 81–124.

[85] M. Berger and P. Colella, *Local adaptive mesh refinement for shock hydrodynamics*, *J. Comput. Phys.* **82** (1989) 64–84.

[86] D. DeZeeuw and K. G. Powell, *An adaptively refined cartesian mesh solver for the euler equations*, *Journal of Computational Physics* **104** (1993), no. 1 56 – 68.

[87] S. Karman, *Splitflow: A 3d unstructured cartesian/prismatic grid cfd code for complex geometries*, in *33rd Aerospace Sciences Meeting and Exhibit*, 1995.

[88] J. Melton, *Automated three-dimensional cartesian grid generation and euler flow solutions for arbitrary geometries*. PhD thesis, Univerty of California, Davis, 1996.

[89] J. Melton, M. Berger, M. Aftosmis, and M. Wong, *3d applications of a cartesian grid euler method*, in *33rd Aerospace Sciences Meeting and Exhibit*, 1995.

[90] J. Melton, F. Enomoto, and M. Berger, *3d automatic cartesian grid generation for euler flows*, in *11th Computational Fluid Dynamics Conference, AIAA-93-3386-CP*, 1993.

[91] D. Young, R. Melvin, M. Bieterman, F. Johnson, S. Samant, and J. Bussoletti, *A locally refined rectangular grid finite element method: Application to computational fluid dynamics and computational physics*, *J. Comput. Phys.* **92** (1991) 1–66.

[92] C. R. Corporation, "http://www.calmarresearch.com/nf/stg/tranair/tranair.htm."

[93] M. Aftosmis, M. Berger, and M. Nemec, "http://people.nas.nasa.gov/aftosmis/cart3d/cart3dhome.html."

[94] A. Chorin, *A numerical method for solving incompressible viscous flow problems*, *J. Comput. Phys.* **2** (1967) 12–26.

[95] D. Brown, R. Cortez, and M. Minion, *Accurate projection methods for the incompressible navier-stokes equations*, *J. Comput. Phys.* **168** (2001) 464–499.

[96] F. Harlow and J. Welch, *Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface*, *Phys. Fluids* **8** (1965) 2182–2189.

[97] K. Lipnikov, J. Morel, and M. Shashkov, *Mimetic finite difference methods for diffusion equations on non-orthogonal non-conformal meshes*, *J. Comput. Phys.* **199** (2004) 589–597.

[98] C. Min and F. Gibou, *A second order accurate projection method for the incompressible Navier-Stokes equation on non-graded adaptive grids*, *J. Comput. Phys.* **219** (2006) 912–929.

[99] C. Peskin, *Flow patterns around heart valves: A numerical method*, *J. Comput. Phys.* **10** (1972) 252–271.

[100] E. Fadlun, R. Verzicco, P. Orlandi, and J. Mohd-Yusof, *Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations*, *Journal of Computational Physics* **161** (2000), no. 1 35 – 60.

[101] C. Batty, F. Bertails, and R. Bridson, *A fast variational framework for accurate solid-fluid coupling*, *ACM Trans. Graph. (SIGGRAPH Proc.)* **26** (2007), no. 3.

[102] Y. T. Ng, C. Min, and F. Gibou, *An efficient fluid–solid coupling algorithm for single-phase flows*, *Journal of Computational Physics* **228** (Dec., 2009) 8807–8829.

[103] F. Hermeline, *Two coupled particle-finite volume methods using delaunay-voronoi meshes for the approximation of vlasov-poisson and vlasov-maxwell equations*, *Journal of Computational Physics* **106** (1993) 1–18.

[104] F. Gibou, R. Fedkiw, L.-T. Cheng, and M. Kang, *A second–order–accurate symmetric discretization of the Poisson equation on irregular domains*, *J. Comput. Phys.* **176** (2002) 205–227.

[105] X. Long and C. Chen, *General formulation of second-order semi-lagrangian methods for convection-diffusion problems*, *Abstract and Applied Analysis* **2013** (2013).

[106] R. E. English, L. Qiu, Y. Yu, and R. Fedkiw, *An adaptive discretization of incompressible flow using a multitude of moving cartesian grids*, *Journal of Computational Physics* **254** (2013) 107 – 154.

[107] C. H. Rycroft, *Voro++: A three-dimensional voronoi cell library in c++*, *Chaos* **19** (2009) 041111.

[108] S. Balay, J. Brown, , K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Users Manual*. Argonne National Laboratory, 2012.

[109] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in *Modern Software Tools in Scientific Computing* (B. user Press, ed.), pp. 163—202, 2012.

[110] H. Chen, C. Min, and F. Gibou, *A supra-convergent finite difference scheme for the poisson and heat equations on irregular domains and non-graded adaptive cartesian grids*, *J. Sci. Comput.* **31** (2007) 19–60.

[111] F. Gibou and R. Fedkiw, *A fourth order accurate discretization for the Laplace and heat equations on arbitrary domains, with applications to the Stefan problem*, *J. Comput. Phys.* **202** (2005) 577–601.

[112] U. Ghia, K. N. Ghia, and C. T. Shin, *High-re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method*, *J. Comput. Phys.* **48** (1982) 387–411.

[113] E. Erturk, T. Corke, and C. Gokcol, *Numerical solutions of 2-d steady incompressible driven cavity flow at high reynolds numbers*, *International Journal for Numerical Methods in Fluids* (2005).

[114] S. Dennis and G. Chang, *Numerical solutions for steady flow past a circular cylinder at Reynolds number up to 100*, *J. Fluid. Mech.* **42** (1970) 471.

[115] M. Braza, P. Chassaing, and H. H. Minh, *Numerical study and phyiscal analysis of the pressure and velocity fields in the near wake of a circular cylinder*, *Journal of Fluid Mechanics* **165** (1986) 79–130.

[116] D. Calhoun, *A cartesian grid method for solving the two-dimensional streamfunction-vorticity equations in irregular regions*, *J. Comput. Phys.* **176** (2002) 231–275.

[117] M. Laroussi, M. Djebbi, and M. Moussa, *Triggering vortex shedding for flow past circular cylinder by acting on initial conditions: A numerical study*, *Computers & Fluids* **101** (2014) 194 – 207.

[118] C. Min and F. Gibou, *Robust second order accurate discretizations of the multi-dimensional heaviside and dirac delta functions*, *J. Comput. Phys.* **227** (2008) 9686–9695.

[119] M. S. Engelman and M.-A. Jamnia, *Transient flow past a circular cylinder: A benchmark solution*, *International Journal for Numerical Methods in Fluids* **11** (1990) 985–1000.

[120] M. Rosenfeld, D. Kwak, and M. Vinokur, *A fractional step solution method for the unsteady incompressible navier-stokes equations in generalized coordinate systems*, *Journal of Computational Physics* **94** (1991), no. 1 102 – 137.

[121] N. Mahír and Z. Altaç, *Numerical investigation of convective heat transfer in unsteady flow past two cylinders in tandem arrangements*, *International Journal of Heat and Fluid Flow* **29** (2008), no. 5 1309 – 1318.

[122] S. Muddada and B. Patnaik, *An active flow control strategy for the suppression of vortex structures behind a circular cylinder*, *European Journal of Mechanics - B/Fluids* **29** (2010), no. 2 93 – 104.

[123] H. Dutsch, F. Durst, S. Becker, and H. Lienhart, *Low-reynolds-number flow around an oscillating circular cylinder at low keulegan-carpenter numbers*, *Journal of Fluid Mechanics* **360** (1998) 249–271.

[124] J. H. Seo and R. Mittal, *A sharp-interface immersed boundary method with improved mass conservation and reduced spurious pressure oscillations*, *Journal of Computational Physics* **230** (2011), no. 19 7347 – 7363.

[125] C.-C. Liao, Y.-W. Chang, C.-A. Lin, and J. McDonough, *Simulating flows with moving rigid boundary using immersed-boundary method*, *Computers & Fluids* **39** (2010), no. 1 152 – 167.

[126] R. Mittal, H. Dong, M. Bozkurttas, F. Najjar, A. Vargas, and A. von Loebbecke, *A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries*, *Journal of Computational Physics* **227** (2008), no. 10 4825 – 4852.

[127] S. Marella, S. Krishnan, H. Liu, and H. Udaykumar, *Sharp interface cartesian grid method i: An easily implemented technique for 3d moving boundary computations*, *Journal of Computational Physics* **210** (Nov., 2005) 1–31.

[128] B. L. Clair and A. Hamielec, *A numerical study of the drag on a sphere at low and intermediate reynolds numbers*, *Journal of the atmospheric sciences* **27** (1969).

[129] T. A. Johnson and V. C. Patel, *Flow past a sphere up to a reynolds number of 300*, *Journal of Fluid Mechanics* **378** (1, 1999) 19–70.

[130] P. Bagchi, M. Y. Ha, and S. Balachandar, *Direct numerical simulation of flow and heat transfer from a sphere in a uniform cross-flow*, *Journal of Fluids Engineering* **123** (2001), no. 2 347–358.

[131] M. Mirzadeh, A. Guittet, C. Burstedde, and F. Gibou, *Parallel level-set methods on adaptive tree-based grids*, *Journal of Computational Physics, in review.*

[132] K. Shahbazi, P. F. Fischer, and C. R. Ethier, *A high-order discontinuous galerkin method for the unsteady incompressible navier-stokes equations*, *Journal of Computational Physics* **222** (2007), no. 1 391 – 407.

[133] V. Girault, B. Riviere, and M. F. Wheeler, *A discontinuous galerkin method with nonoverlapping domain decomposition for the stokes and navier-stokes problems*, *Mathematics of Computation* **74** (2005) 53–84.

[134] M.-C. Lai and C. S. Peskin, *An immersed boundary method with formal second-order accuracy and reduced numerical viscosity*, *Journal of Computational Physics* **160** (2000), no. 2 705 – 719.

[135] R. LeVeque and Z. Li, *The immersed interface method for elliptic equations with discontinuous coefficients and singular sources 31:1019–1044, 1994*, *SIAM J. Numer. Anal.* **31** (1994) 1019–1044.

[136] J. A. Benek, J. Steger, and F. C. Dougherty, *A flexible grid embedding technique with applications to the Euler equations*, *6th Computational Fluid Dynamics Conference, AIAA, 373–382.* (1983).

[137] R. Finkel and J. Bentley, *Quad trees a data structure for retrieval on composite keys*, *Acta Informatica* **4** (1974), no. 1 1–9.

[138] D. Meagher, *Geometric modeling using octree encoding*, *Computer Graphics and Image Processing* **19** (1982), no. 2 129 – 147.

[139] W. J. Coirier, *An adaptively-refined, cartesian, cell-based scheme for the euler and navier-stokes equations*, tech. rep., 1994.

[140] A. Khokhlov, *Fully threaded tree algorithms for adaptive refinement fluid dynamics simulations*, *Journal of Computational Physics* **143** (1998), no. 2 519 – 543.

[141] J. Strain, *Semi-lagrangian methods for level set equations*, *Journal of Computational Physics* **151** (1999) 498–533.

[142] H.-J. Bungartz, M. Mehl, and T. Weinzierl, *A parallel adaptive Cartesian PDE solver using space–filling curves*, *Euro-Par 2006 Parallel Processing* (2006) 1064–1074.

[143] T. Weinzierl and M. Mehl, *Peano—a traversal and storage scheme for octree-like adaptive Cartesian multiscale grids*, *SIAM Journal on Scientific Computing* **33** (Oct., 2011) 2732–2760.

[144] I. D. Mishev, *Finite volume methods on voronoi meshes*, *Numerical Methods for Partial Differential Equations* **14** (1998), no. 2 193–212.

[145] J. Kim and H. Choi, *An immersed-boundary finite-volume method for simulation of heat transfer in complex geometries*, *KSME International Journal* **18** (2004), no. 6 1026–1035.

[146] G. Constantinescu and K. Squires, *Les and des investigations of turbulent flow over a sphere at re = 10,000*, *Flow, Turbulence and Combustion* **70** (2003), no. 1-4 267–298.

[147] J.-I. Choi, R. C. Oberoi, J. R. Edwards, and J. A. Rosati, *An immersed boundary method for complex incompressible flows*, *Journal of Computational Physics* **224** (2007), no. 2 757 – 784.

[148] I. Babuška, *The finite element method for elliptic equations with discontinuous coefficients*, *Computing* **5** (1970) 207–213.

[149] J. Bramble and J. King, *A finite element method for interface problems in domains with smooth boundaries and interfaces*, *Adv. Comput. Math.* **6** (1996) 109–138.

[150] Z. Chen and J. Zou, *Finite element methods and their convergence for elliptic and parabolic interface problems*, *Num. Math.* **79** (1996) 175–202.

[151] M. Dryja, *A neumann-neumann algorithm for mortar discretization of elliptic problems with discontinuous coefficients*, *Num. Math.* **99** (2005) 645–656.

[152] J. Huang and J. Zou, *A mortar element method for elliptic problems with discontinuous coefficients*, *IMA J. Numer. Anal.* **22** (2001) 549–576.

[153] B. Lamichhane and B. Wohlmuth, *Mortar finite elements for interface problems*, *Computing* **72** (2004) 333–348.

[154] Sandia National Laboratory, *International meshing roundtable*, (Thistle Marble Arch, London, United Kingdom), 2014.

[155] Z. Li, T. Lin, and X. Wu, *New cartesian grid methods for interface problems using the finite element formulation*, *Numerische Mathematik* **96** (2003), no. 1 61–98.

[156] Y. Gong, B. Li, and Z. Li, *Immersed-interface finite-element methods for elliptic interface problems with nonhomogeneous jump conditions*, *SIAM Journal on Numerical Analysis* **46** (2008), no. 1 472–495.

[157] R. E. Ewing, Z. Li, T. Lin, and Y. Lin, *The immersed finite volume element methods for the elliptic interface problems*, *Mathematics and Computers in Simulation* **50** (1999), no. 1–4 63 – 76.

[158] T. Chen and J. Strain, *Piecewise-polynomial discretization and krylov-accelerated multigrid for elliptic interface problems*, *Journal of Computational Physics* **227** (2008), no. 16 7503 – 7542.

[159] Z. Li, *A fast iterative algorithm for elliptic interface problems*, *SIAM J. Numer. Anal.* **35** (1998) 230–254.

[160] Z. Li and K. Ito, *The Immersed Interface Method – Numerical Solutions of PDEs Involving Interfaces and Irregular Domains*, vol. 33. SIAM Frontiers in Applied mathematics, 2006.

[161] A. Wiegmann and K. Bube, *The explicit-jump immersed interface method: finite difference methods for PDEs with piecewise smooth solutions*, *SIAM J. Numer. Anal.J* **37** (2000) 827–862.

[162] P. A. Berthelsen, *A decomposed immersed interface method for variable coefficient elliptic equations with non-smooth and discontinuous solutions*, *Journal of Computational Physics* **197** (2004) 364–386.

[163] L. Adams and T. Chartier, *New geometric immersed interface multigrid solvers*, *SIAM J. of Scientific Comput.* **25** (2004) 1516–1533.

[164] L. Adams and T. Chartier, *A comparison of algebraic multigrid and geometric immersed interface multigrid methods for interface problems*, *SIAM J. of Scientific Comput.* **26** (2005) 762–784.

[165] L. Adams and Z. Li, *The immersed interface/multigrid methods for interface problems*, *SIAM J. of Scientific Comput.* **24** (2002) 463–479.

[166] A. Mayo, *The fast solution of Poisson's and the biharmonic equations on irregular regions*, *SIAM J. Numer. Anal.* **21** (1984) 285–299.

[167] S. Zhao and G. Wei, *High-order {FDTD} methods via derivative matching for maxwell's equations with material interfaces*, *Journal of Computational Physics* **200** (2004), no. 1 60 – 103.

[168] S. Yu, Y. Zhou, and G. Wei, *Matched interface and boundary (mib) method for elliptic problems with sharp-edged interfaces*, *J. Comput. Phys.* **224** (June, 2007) 729–756.

[169] A. J. Lew and G. C. Buscaglia, *A discontinuous-Galerkin-based immersed boundary method*, *Int. J. for Num. Meth. in Eng.* **76** (2008) 427–454.

[170] G. Guyomarch, C.-O. Lee, and K. Jeon, *A discontinuous Galerkin method for elliptic interface problems with application to electroporation*, *Commun. Numer. Methods Eng.* **25** (2009) 991–1008.

[171] N. Moës, J. Dolbow, and T. Belytschko, *A finite element method for crack growth without remeshing*, *Int. J. for Num. Meth. Eng.* **46** (1999) 131–150.

[172] C. Daux, N. Moës, J. Dolbow, N. Sukumar, and T. Belytschko, *Arbitrary branched and intersecting cracks with the extended finite element method*, *Int. J. for Num. Meth. Eng.* **48** (2000) 1741–1760.

[173] T. Belytschko, N. Moës, S. Usui, and C. Parimi, *Arbitrary discontinuities in finite elements*, *Int. J. for Num. Meth. Eng.* **50** (2001) 993–1013.

[174] N. Moës, M. Cloirec, P. Cartraud, and J. Remacle, *A computational approach to handle complex microstructure geometries*, *Comput. Methods Appl. Mech. Eng.* **192** (2003) 3162–3177.

[175] H. Ji and J. Dolbow, *On strategies for enforcing interfacial constraints and evaluating jump conditions with extended finite element method*, *Int. J. for Num. Meth. in Eng.* **61** (2004), no. 2508-2535.

[176] T. Fries and T. Belytschko, *The intrinsic XFEM: a method for arbitrary discontinuities without additional unknowns*, *Int. J. for Num. Meth. in Eng.* **68** (2006) 1358–1385.

[177] S. Groß and A. Reusken, *An extended pressure finite element space for two-phase incompressible flows with surface tension*, *J. Comp. Phys.* **224** (2007) 40–58.

[178] F. van der Bos and V. Gravemeier, *Numerical simulation of premixed combustion using an enriched finite element method*, *J. Comp. Phys.* **228** (2009) 3605–3624.

[179] R. Crockett, P. Colella, and D. Graves, *A cartesian grid embedded boundary method for solving the poisson and heat equations with discontinuous coefficients in three dimensions*, *Journal of Computational Physics* **230** (2011), no. 7 2451 – 2469.

[180] M. Oevermann, C. Scharfenberg, and R. Klein, *A sharp interface finite volume method for elliptic equations on Cartesian grids*, *Journal of Computational Physics* **228** (2009) 5184–5206.

[181] R. Fedkiw, T. Aslam, B. Merriman, and S. Osher, *A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)*, *J. Comput. Phys.* **152** (1999) 457–492.

[182] X. Liu, R. Fedkiw, and M. Kang, *A boundary condition capturing method for Poisson's equation on irregular domains*, *J. Comput. Phys.* **154** (2000) 151.

[183] X.-D. Liu and T. Sideris, *Convergence of the ghost-fluid method for elliptic equations with interfaces*, *Math. Comp.* **72** (2003) 1731–1746.

[184] J. Papac, F. Gibou, and C. Ratsch, *Efficient symmetric discretization for the Poisson, heat and Stefan-type problems with Robin boundary conditions*, *Journal of Computational Physics* **229** (Feb., 2010) 875–889.

[185] J. Papac, A. Helgadottir, C. Ratsch, and F. Gibou, *A level set approach for diffusion and stefan-type problems with robin boundary conditions on quadtree/octree adaptive cartesian grids*, *Journal of Computational Physics* (2012), no. 0 –.

[186] F. Gibou, C. Min, and R. Fedkiw, *High resolution sharp computational methods for elliptic and parabolic problems in complex geometries*, *J. Sci. Comput.* **54** (2013) 369–413.

[187] A. Coco and G. Russo, *Second order multigrid methods for elliptic problems with discontinuous coefficients on an arbitrary interface, i: One dimensional problems*, *Numerical Mathematics: Theory, Methods & Applications* **5** (2012) 19.

[188] A. Coco and G. Russo, *Finite-difference ghost-point multigrid methods on cartesian grids for elliptic problems in arbitrary domains*, *Journal of Computational Physics* **241** (2013) 464 – 501.

[189] M.Latige, T.Colin, and G.Gallice, *A second order cartesian finite volume method for elliptic interface and embedded dirichlet problems*, *Computers and Fluids* **83** (2013) 70–76.

[190] S. Hou, W. Wang, and L. Wang, *Numerical method for solving matrix coefficient elliptic equation with sharp-edged interfaces*, *Journal of Computational Physics* **229** (2010), no. 19 7162 – 7179.

[191] N. Molino, J. Bao, and R. Fedkiw, *A virtual node algorithm for changing mesh topology during simulation*, *ACM Trans. Graph. (SIGGRAPH Proc.)* **23** (2004) 385–392.

[192] Z. Bao, J.-M. Hong, J. Teran, and R. Fedkiw, *Fracturing rigid materials*, *IEEE Trans. on Vis. and Comput. Graph.* **13** (2007) 370–378.

[193] A. Hansbo and P. Hansbo, *A finite element method for the simulation of strong and weak discontinuities in solid mechanics*, *Comput. Meth. in Appl. Mech. and Eng.* **1993** (2004) 3523–3540.

[194] J.-H. Song, P. Areias, and T. Belytschko, *A method for dynamic crack and shear band propagation with phatom nodes*, *Int. J. for Num. Meth. in Eng.* **67** (2006) 868–893.

[195] J. Dolbow and I. Harari, *An efficient finite element method for embedded interface problems*, *Int. J. for Num. Meth. in Eng.* **78** (2009) 229–252.

[196] E. Sifakis, K. Der, and R. Fedkiw, *Arbitrary cutting of deformable tetrahedralized objects*, in *Proceedings of SIGGRAPH 2007*, pp. 73–80, 2007.

[197] C. Richardson, J. Hegemann, E. Sifakis, J. Hellrung, and J. Teran, *An XFEM method for modeling geometrically elaborate crack propagation in brittle materials*, *Int. J. for Num. Meth. in Eng.* **88** (2011) 1042–1065.

[198] J. L. J. Hellrung, L. Wang, E. Sifakis, and J. M. Teran, *A second order virtual node method for elliptic problems with interfaces and irregular domains in three dimensions*, *Journal of Computational Physics* **231** (2012), no. 4 2015 – 2048.

[199] M. Cisternino and L. Weynans, *A parallel second order Cartesian method for elliptic interface problems*, *Commun. Comput. Phys.* **12** (2012) 1562–1587.

[200] R. Vanselow, *Relations between fem and fvm applied to the poisson equation*, *Computing* **57** (Sept., 1996) 93–104.

[201] N. Sukumar, *Voronoi cell finite difference method for the diffusion operator on arbitrary unstructured grids*, *International Journal for Numerical Methods in Engineering* **57** (2003), no. 1 1–34.

[202] R. D. Falgout and U. M. Yang, *hypre: A library of high performance preconditioners*, in *Computational Science - ICCS 2002* (P. M. Sloot, A. G. Hoekstra, C. K. Tan, and J. J. Dongarra, eds.), vol. 2331 of *Lecture Notes in Computer Science*, pp. 632–641. Springer Berlin Heidelberg, 2002.

[203] Z. Chen and J. Zou, *Finite element methods, based on nitsche's, method for elliptic interface problems*, *Numer. Math.* **79** (1998) 175 – 202.

[204] A. Hansbo and P. Hansbo, *An unfitted finite element method, based on nitsche's, method for elliptic interface problems*, *Comput. Methods Appl. Mech. Eng.* **191** (2002) 5537 – 5552.

[205] L. Parussini and V. Pediroda, *Fictious Domain approach with hp-finite element approximation for incompressible fluid flow*, *J. Comput. Phys.* **228** (2009) 3891 – 3910.

[206] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[207] Z. Tan, D. Le, Z. Li, K. Lim, and B. Khoo, *An immersed interface method for solving incompressible viscous flows with piecewise constant viscosity across a moving elastic membrane*, *Journal of Computational Physics* **227** (Dec., 2008) 9955–9983.

[208] J. Beale and A. Layton, *On the accuracy of finite difference methods for elliptic problems with interfaces*, *Commun. Appl. Math. Comput.* **1** (2006) 207 – 208.

[209] A. Weigmann and K. Bube, *The explicit-jump immersed interface method: finite difference method for pdes with piecewise smooth solutions*, *SIAM J. Sci. Comput.* **37** (2000) 827 – 862.

[210] Z. Jomaa and C. Macaskill, *The embedded finite difference method for the poisson equation in a domain with and irregular boundary and dirichlet boundary conditions*, *J. Comput. Phys.* **202** (2005) 488–506.

[211] I.-L. Chern and Y.-C. Shu, *A coupling interface method for elliptic interface problems*, *J. Comput. Phys.* **225** (2007) 2138–2174.

[212] Y. Zhou, S. Zhao, M. Feig, and G. Wei, *High order mathced interterface and boundary method for elliptic equations with discontinuous coefficients and singular sources*, *J. Comput. Phys.* **213** (2006) 1 – 30.

[213] C. Peskin, *Numerical analysis of blood flow in the heart*, *J. Comput. Phys.* **25** (1977) 220–252.

[214] C. Peskin, *The immersed boundary method*, *Acta Numerica* **11** (2002) 479–517.

[215] C. Peskin and B. Printz, *Improved volume conservation in the computation of flows with immersed elastic boundaries*, *J. Comput. Phys.* **105** (1993) 146–154.

[216] T. Ho, Z. Li, S. Osher, and H. Zhao, *A hybrid method for moving interface problems with application to the Hele–Shaw flow*, *J. Comput. Phys.* **134** (1997) 236–252.

[217] L. Mir, *Therapeutic perspectives of in vivo cell electropermeabilization*, *Bioelectrochemistry* **53** (2001) 1–10.

[218] B. Gabriel and J. Teissié, *Time courses of mammalian cell electropermeabilization observed by millisecond imaging of membrane property changes during the pulse.*, *Biophys. J.* **76** (1999), no. 4 2158–2165 (electronic).

[219] M. Vernhes, P. Cabanes, and J. Teissié, *Chinese hamster ovary cells sensitivity to localized electrical stresses.*, *Bioelectrochem. Bioenerg.* **48** (1999) 17–25.

[220] J. Teissié, M. Golzio, and M. Rols, *Mechanisms of cell membrane electropermeabilization: A minireview of our present (lack of ?) knownledge*, *Biochimica et Biophysica Acta* **1724** (2005) 270–280.

[221] K. DeBruin and W. Krassowska, *Modelling electroporation in a single cell. I. Effects of field strength and rest potential.*, *Biophysical Journal* **77** (Sept, 1999) 1213–1224.

[222] J. Weaver, *Electroporation of cells and tissues*, *IEEE Trans. on Plasma Sci.* **28** (2000).

[223] Z. Vasilkoski, A. T. Esser, T. R. Gowrishankar, and J. C. Weaver, *Membrane electroporation: The absolute rate equation and nanosecond time scale pore creation.*, *Phys Rev E Stat Nonlin Soft Matter Phys* **74** (Aug, 2006) 021904.

[224] O. Kavian, M. Leguèbe, C. Poignard, and L. Weynans, *"Classical" electropermeabilization modeling at the cell scale*, *J. Math. Biol.* **68** (2014), no. 1-2 235–265.

[225] M. Leguèbe, A. Silve, L. Mir, and C. Poignard, *Conducting and permeable states of cell membrane submitted to high voltage pulses: Mathematical and numerical studies validated by the experiments*, *Journal of Theoretical Biology* **360** (2014) 83–94. cited By 0.

[226] M. Mirzadeh, M. Theillard, and F. Gibou, *A Second-Order Discretization of the Nonlinear Poisson-Boltzmann Equation over Irregular Geometries using Non-Graded Adaptive Cartesian Grids*, *Journal of Computational Physics* **230** (Dec., 2010) 2125–2140.

[227] M. Leguèbe, C. Poignard, and L. Weynans, *A second-order Cartesian method for the simulation of electropermeabilization cell models*, *J. Comput. Phys.* **292** (2015) 114–140.

[228] F. Hirschhaeuser, H. Menne, C. Dittfeld, J. West, W. Mueller-Klieser, and L. A. Kunz-Schughart, *Multicellular tumor spheroids: an underestimated tool is catching up again*, *Journal of Biotechnology* **148** (July, 2010) 3–15.

[229] L. Gibot, L. Wasungu, J. Teissié, and M.-P. Rols, *Antitumor drug delivery in multicellular spheroids by electropermeabilization*, *Journal of Controlled Release: Official Journal of the Controlled Release Society* **167** (Apr., 2013) 138–147.

[230] K. Foster and H. Schwan, *Dielectric properties of tissues and biological materials: a critical review*, *CRC in Biomedical Engineering* **17** (1989), no. 1 25–104.

[231] E. Fear and M. Stuchly, *Modelling assemblies of biological cells exposed to electric fields.*, *IEEE Trans Biomed Eng* **45** (Oct, 1998) 1259–1271.

[232] C. Min, *Local level set method in high dimension and codimension*, *J. Comput. Phys.* **200** (2004) 368–382.

[233] G. Pucihar, T. Kotnik, J. Teissié, and D. Miklavčič, *Electropermeabilization of dense cell suspensions*, *European Biophysics Journal* **36** (2007), no. 3 173–185.

[234] S. Čorović, A. Županič, S. Kranjc, B. Al Sakere, A. Leroy-Willig, L. M. Mir, and D. Miklavčič, *The influence of skeletal muscle anisotropy on electroporation: in vivo study and numerical modeling*, *Medical & Biological Engineering & Computing* **48** (july, 2010) 637–648.