Scalable Mesh Partitioning and Robust Sub-Cell Geometry Treatment for the Ghost-Cell Immersed Boundary Method on Cartesian Meshes

F. Salmon[a,*], A. Lemoine[a], S. Glockner[a], D. Lacanette[a]

[a]*Université de Bordeaux, CNRS, ENSAM, Bordeaux-INP, UMR 5295 I2M, 33405 Talence, France*

Abstract

In computational fluid dynamics, complex geometries can be efficiently handled by embedding their surface representation into a Cartesian grid and enforcing boundary or interface conditions using immersed boundary methods. While this approach greatly simplifies mesh generation, it introduces two major challenges in large-scale parallel simulations. First, classical mesh partitioning strategies may allocate a significant fraction of the computational domain outside the physical domain, leading to unnecessary and sometimes prohibitive computational costs. Second, the interaction between complex geometries and Cartesian meshes may produce degenerate cells, which severely limits the robustness of the ghost-cell method when mesh refinement or geometry adaptation is not possible. In this work, we address both issues within a unified parallel framework. A scalable mesh partitioning strategy is first proposed, in which partition boundaries are dynamically adjusted to exclude subdomains lying entirely outside the physical domain. This sliding-partition approach significantly reduces the number of required processors while preserving load balance and parallel efficiency. The method is applied to complex two- and three-dimensional geometries and demonstrates low computational times up to 50 billion cells distributed over 500,000 partitions. Second, we introduce a geometry modification algorithm that locally adapts arbitrary 2D and 3D geometries to a given Cartesian mesh, ensuring the complete elimination of degenerate cells. The resulting geometries intersect each mesh cell through a single line segment in 2D or a continuous surface in 3D, thereby guaranteeing the robustness of ghost-cell formulations. The algorithm is fast, parallelizable, and applicable to any geometry–mesh combination. The combined approach enables robust and scalable ghost-cell immersed boundary simulations on Cartesian meshes. Its effectiveness is demonstrated through large-scale simulations of flow in porous media and haemodynamics in complex arterial networks.

Keywords: *Immersed boundary method, ghost-cell, partitioning, degenerate cell, geometry reconstruction algorithm*

1. Introduction

Immersed Boundary Methods (IBMs), originally developed by Peskin [1] and used particularly in Computational Fluid Dynamics (CFD), are numerical approaches often based on a Cartesian grid that avoid the sensitive meshing process of body-fitted grids. Instead, the geometry is immersed in the discretised domain and appropriate numerical treatments of the boundary condition are considered. The numerical domain is divided into an inner and an outer domain. The inner domain corresponds to the physical domain in which

*Corresponding author

Email address: fabien.salmon@bordeaux-inp.fr (F. Salmon)

the partial differential equations are solved, while the outer domain consists of the cells in the remaining part of the mesh. In the context of this article, the outer domain is not associated with any model, but still fills the matrices of the linear systems to balance the load between processors.

IBMs can be divided into two categories [2, 3], continuous or discrete forcing. The former consists in adding a source term to the equations before discretisation. The latter consists in modifying the discretised equations at the boundaries. Among them, the cut-cell method [4, 5, 6, 7] is based on the finite volume approach applied to the cut non-Cartesian boundary cells. The direct forcing ghost-cell methods [8, 9] are based on cells outside the inner domain where values are enforced depending on the boundary conditions. Fig. 1 shows how the mesh cells are divided into three groups: inner, outer and ghost cells. This IBM is widely used [10, 11, 12, 13] and has been extended by various adjustments to reduce the stencil size [14, 15]. It yields band matrices, which ensures the use of highly efficient massively parallel linear solvers such as the hypre library [16].

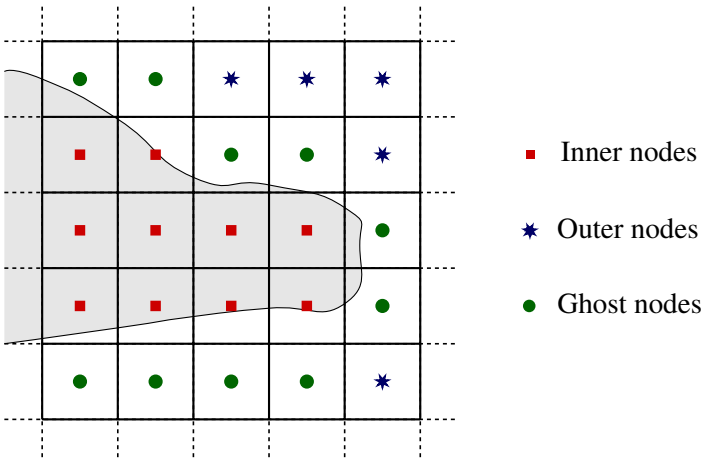

Figure 1: Immersed Boundary Method. The numerical domain is separated into inner, outer and ghost cells.

Immersing geometry in a Cartesian mesh requires that its bounding box encompasses the entire geometry. For complex geometries, this can lead to an excessive number of cells outside the physical domain, making IBMs CPU-time inefficient, especially for very large meshes. Few authors have attempted to reduce the number of outer cells in order to minimise the use of computational resources. A first approach was proposed by Anupindi et al. [17], who first performed the Cartesian partitioning and then removed the partitions outside the physical domain. This approach preserves the main advantage of IBMs, since each partition remains rectangular in 2D or cuboidal in 3D. However, this simple approach still leads to partitions with few inner cells. A second approach was proposed by Zhu et al. [18], who first used an unstructured partitioning of the inner cells, managed by the METIS partitioning library [19], and then added outer cells to build rectangular (or cuboidal in 3D) partitions. Unfortunately, it should be noted that METIS becomes too CPU time consuming for very large meshes. Moreover, the parallel version of METIS (parMETIS) cannot guarantee contiguous and compact partitions [20], making this method unusable on very large meshes.

The ghost-cell method allows the application of second-order boundary conditions, which leads to a better approximation of the flow near boundaries compared to the first-order volume penalization method

[1, 21, 22]. The generation of Cartesian cut-cell meshes is useful and can be seen as a prerequisite before applying a ghost-cell method [23]. Cart3D [24] and EBChombo [25] have been developed to generate such meshes, based on ray-casting and level-set methods respectively. According to Tao et al. [26], these codes are not generic enough. They therefore proposed a new tool, called Mandoline [26], which aims to be more powerful, although it still fails on certain configurations such as isolated triangular mesh geometries. However, a numerical constraint of the ghost-cell method is to avoid cells containing more than one boundary line segment in 2D and one continuous boundary surface in 3D. Cells that do not satisfy this condition are called degenerate cells. Without special treatment, they can lead to numerical instabilities. Such cells are particularly common with complex boundaries, and no existing Cartesian cut-cell meshing tool can remove degenerate cells from meshes. It should be noted that complex geometries can be simplified thanks to Laplacian smoothing [27, 28, 29], which reduces irregularities. However, this process affects the immersed geometry without considering the Cartesian mesh. It is therefore not possible to guarantee that there are no degenerate cells regardless of the mesh, and the process must therefore be performed manually and iterated several times. This method is therefore as efficient as a simple manual modification of the geometry. Instead of working on the immersed geometry connection with the Cartesian mesh, some authors have worked on numerical methods to be able to consider degenerate cells in the calculations. A common approach for the cut-cell method is to simply approximate the boundary with the Volume-Of-Fluid method [30, 31, 32]. This approach is a rough approximation of the geometry and therefore suffers from inaccuracies. Another way to overcome this problem is to merge small cut cells to neighbours [6, 7, 33, 34]. This technique allows an accurate description of the geometry, but increases the computational time for complex geometry processing. Ji et al. [35] proposed a hybrid cut-cell/ghost-cell method to avoid the problem of degenerate cells. For the ghost cell methods, no method has been proposed to solve this problem. Currently, the only way to avoid degenerate cells is to refine meshes or to use adaptive mesh refinement close to the geometry, although this cannot solve every configuration as shown in this article.

Both problems (partitioning and degenerate cells) are reasons for the scarce use of IBMs with complex geometries and large meshes [36, 37], which become very CPU time consuming. There is therefore a need for an automatic tool that makes any Cartesian grid work with any geometry. In this article, we propose two novel algorithms that improve the capability of IBMs for fully 2D and 3D irregular geometries. The first algorithm tackles the problem of the large number of outer cells. We improve the Anupindi algorithm [17] by sliding lines of partitions along each other. While keeping rectangular or cuboidal partitions, the number of unnecessary outer cells is reduced and more computation time is saved. The second algorithm generates a Cartesian mesh without degenerate boundary cells. It consists of two parts: the reconstruction of the geometry and its modification to remove degenerate cells. The first part thus corresponds to the construction of a cut-cell Cartesian mesh with no restriction on the input geometry, unlike the meshing tools mentioned in the introduction. The second part goes further by locally modifying the geometries so that the cells contain only a line segment (2D) or a continuous surface (3D).

The proposed decomposition domain algorithm is presented in Section 2. The Anupindi method is compared with the present algorithm on several very large meshes, and a parallelism analysis is performed on two irregular geometries up to 500,000 partitions and 50 billion cells. Section 3 focuses on the algorithm for reconstructing geometries without degenerate cells. The novel method is applied to different 2D and 3D examples and a parallelism analysis is also performed for meshes composed of up to one billion cells. The last section deals with two validation cases and the simulation of arterial haemodynamics and fluid flow through a porous medium using the open-source code Notus [38], where the algorithms have been

implemented.

2. Improved Cartesian domain decomposition

2.1. Partition line sliding algorithm

IBMs requires that the numerical domain contains the whole physical domain. For irregular geometries, many cells can be outside the physical domain but still included in the simulation, leading to excessive use of computational resources. As mentioned in the introduction, Anupindi et al. [17] have proposed to remove the partitions that do not contain any cells of the physical domain. This still allows each processor to be associated with a rectangle (or cuboidal in 3D) domain, preserving the strength of IBMs. Fig. 2.a shows a simple 2D configuration where the global domain is divided into nine partitions and the physical domain is surrounded by the red immersed boundary. Fig 2.b shows the remaining domain after applying the Anupindi method, which has removed two outer partitions. Some partitions contain only a small part of the physical domain, such as the two partitions at the top in Fig. 2.b. In this case, the Anupindi method results in a partitioning that includes partitions that are mainly filled with outer cells.

In this article, we propose to add a step to the Anupindi method by sliding lines of partitions to fill some partitions with only outer cells in order to remove them from the partitioning. The lines of partitions can move in all directions. This method preserves rectangular or hexahedral partitions as well as the load balance between them. Fig. 2.b shows three rows, highlighted by three arrows in front of them, where a partition can fall outside the red contour after sliding. These rows will be referred to as slidable rows in the following. A line can only slide if the partitions are aligned, so the three slidable lines cannot slide together. For instance, after the horizontal movement of row 1, the central vertical movement (row 3) is no longer possible and only the horizontal movement of row 2 remains (Fig. 2.c). From this simple example, we easily deduce that the order of the movements affects the final result. The movements should therefore be ordered to optimise the process. The complexity of this optimisation problem is likely to be exponential, so we suggest a heuristic approach. For each possible sliding motion, we define the number of locked degrees of freedom as the number of slidable lines that could not slide after the motion due to misalignment. Among all the possible sliding motions, we choose the one with the minimum number of locked degrees of freedom. If several equivalent choices are possible, the one that minimizes the communication load between processors is selected. In Fig. 2.b, the displacement of rows 1 and 2 would only affect row 3, while the vertical displacement of row 3 would affect rows 1 and 2. Let us select the displacement associated with row 1. Fig. 2.c displays the result of this movement. As previously noticed, the central vertical line 3 can no longer slide because it is locked by the non-aligned partition of row 1. The only possible movement is therefore the one associated with row 2. Fig. 2.d displays the final result: two additional partitions have been removed compared to the Anupindi method. Note that there are several possible displacement amplitudes for row 2. We suggest always moving the line of partitions by the minimum length so that one partition is outside the physical domain as in Fig. 2.d.

Algorithm 1, based on these ideas, works as follows. Given an initial number of processors n_{procs}, every possible domain decomposition of the computational domain is computed. Applying the algorithm to each domain decomposition may indeed yield different results. This is done by integer factorising of the number of processors with two (2D) or three (3D) integers. Each integer decomposition $n_{proc} = n_x \times n_y \times n_z$ corresponds to a domain decomposition. For each domain decomposition, the outer partitions are removed and the sliding procedure is applied. The final choice depends on the number of remaining partitions and the amount of exchange load between processors. In general, the fewer partitions there are, the less data

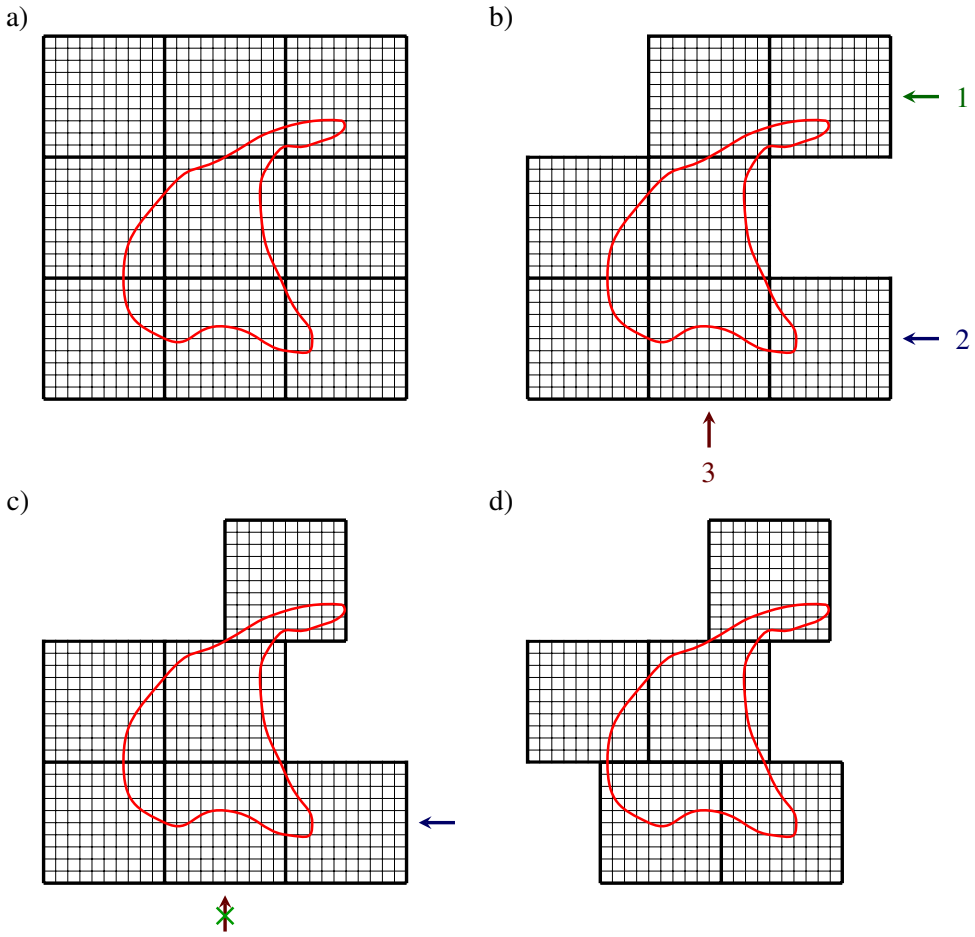

Figure 2: Cartesian domain decomposition algorithm. a) The red contour surrounds the physical domain. Nine partitions are considered in the decomposition domain. b) The outer partitions are first removed from the calculation. We see three potential slip motions represented by the coloured arrows. c) The top row has been horizontally slided and one partition has been removed. There is only one potential move left (blue arrow). d) The bottom row has been moved and another partition has been removed. There is no other potential sliding movement.

is exchanged. Therefore, a good choice is to select the domain decomposition with the smallest number of remaining partitions.

2.2. Results

In this section, we present the results of this algorithm and the comparison with the Anupindi method on two irregular geometries: an arterial system (Fig. 3) and the Lascaux Cave (Fig. 4).

Tab. 1 summarises the results for both geometries with meshes decomposed into 512 and 8,192 partitions. With 8,192 initial partitions, the Anupindi method leads to a domain decomposition with only 856 partitions for the arterial system, i.e. 10.4% of the initial number of partitions. With the proposed algorithm, the number of partitions is reduced by 20% compared to the Anupindi method. With the Anupindi method and the proposed algorithm, the percentage of inner cells increases from 3.2% to 30.9% and 38.6% respectively. For the Lascaux Cave, initially decomposed with 512 partitions, 112 partitions remain with

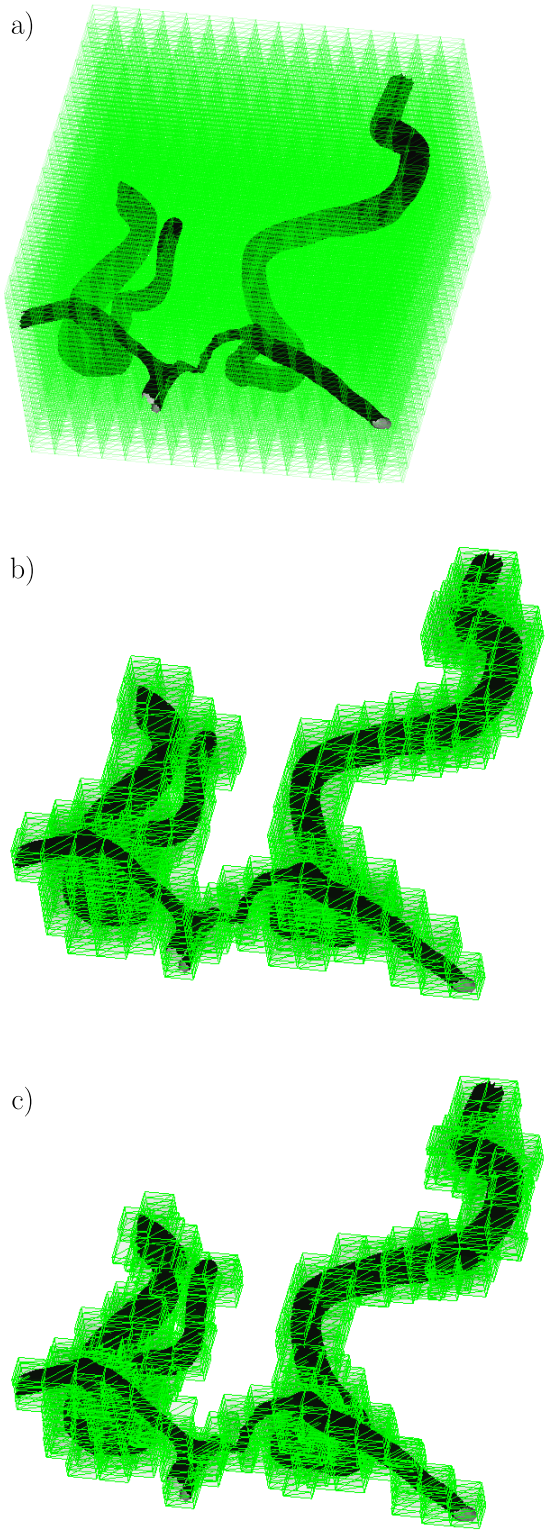

Figure 3: Domain decomposition (in green) of the arterial system (in black). a) Initial partitioning (8192 partitions). b) Partitioning after removing the blocks outside the physical domain (856 partitions left). c) Final partitioning after the sliding process (686 partitions left).

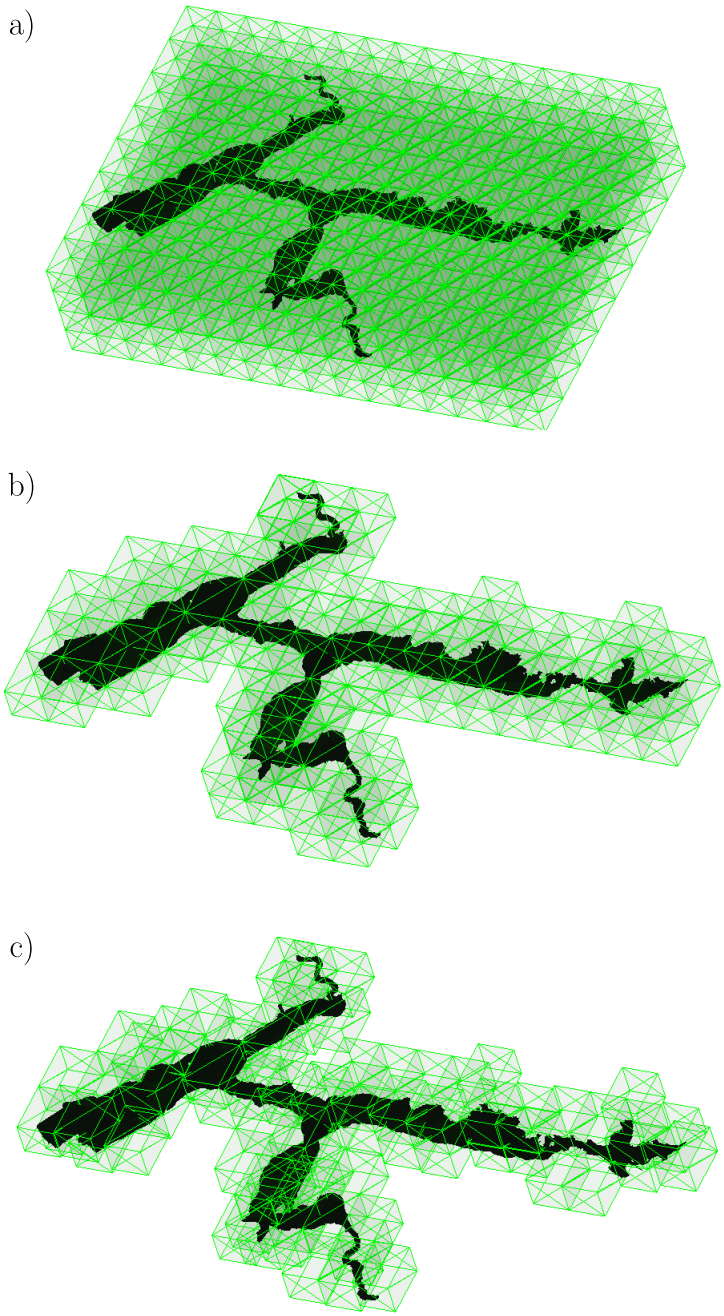

Figure 4: Domain decomposition (in green) of the Lascaux Cave (in black). a) Initial partitioning (512 partitions). b) Partitioning after removing the blocks outside the physical domain (112 partitions left). c) Final partitioning after the sliding process (76 partitions left).

Algorithm 1 Cartesian domain decomposition

Input Number of partitions composing the bounding box

for every possible decomposition domain **do**
  remove the outside partitions (Fig. 2.b)
  while a slide motion can make a partition disappear **do**
    store every possible slide motion
    choose the motion that minimizes the number of affected degrees of freedom
    slide the selected row and remove the outside partition (Fig. 2.c & d)
  end while
  compute the number of remaining partitions and the size of exchanged data
end for
select one domain decomposition based on the number of partitions and the size of exchanged data

the Anupindi method and only 76 partitions with the proposed method, 30% less than with the Anupindi method. With 512 partitions, 9.8% of the cells are inside the physical domain but with 8,192 partitions, the percentage increases to 26.5%. As expected, the percentage of inner cells increases with the number of partitions.

	Arterial system			Lascaux Cave		
	Initial	Anupindi	Proposed method	Initial	Anupindi	Proposed method
Number of partitions	512	119	87	512	112	76
Mesh size (billion)	53	12.3	9	5.8	1.26	0.85
Percentage of inner cells	3.2	13.8	18.8	1.4	6.6	9.8
	Initial	Anupindi	Proposed method	Initial	Anupindi	Proposed method
Number of partitions	8192	856	686	8192	524	389
Mesh size (billion)	53	5.5	4.4	5.8	0.371	0.275
Percentage of inner cells	3.2	30.9	38.6	1.4	19.6	26.5

Table 1: Comparison of the Anupindi method with the proposed algorithm.

In figure 5, we compare the proposed algorithm with the Anupindi method for both geometries and different initial partition numbers. The graph shows the relative difference of the remaining partitions χ, defined as $\chi = \frac{N^{Anupindi} - N^{Present\ study}}{N^{Anupindi}}$ where N refers to the number of remaining partitions after applying the Anupindi method or the proposed algorithm. When the number of initial partitions is small, the gap between both methods is significant (greater than 25% for both geometries). As the number of initial partitions increases, the efficiency of the sliding process decreases. This behaviour is expected because as the number of initial partitions increases, the remaining partitions inside the physical domain become smaller and closely match the boundary, reducing the number of possible sliding motions. The impact of the sliding process therefore diminishes. However, even with 500,000 partitions, the proposed method outperforms the Anupindi method by approximately 10%.

2.3. Parallelization of the algorithm

The sliding algorithm is inherently sequential and cannot be parallelized since each step requires the previous movements. However, the algorithm can still be parallelized by looping over each possible domain

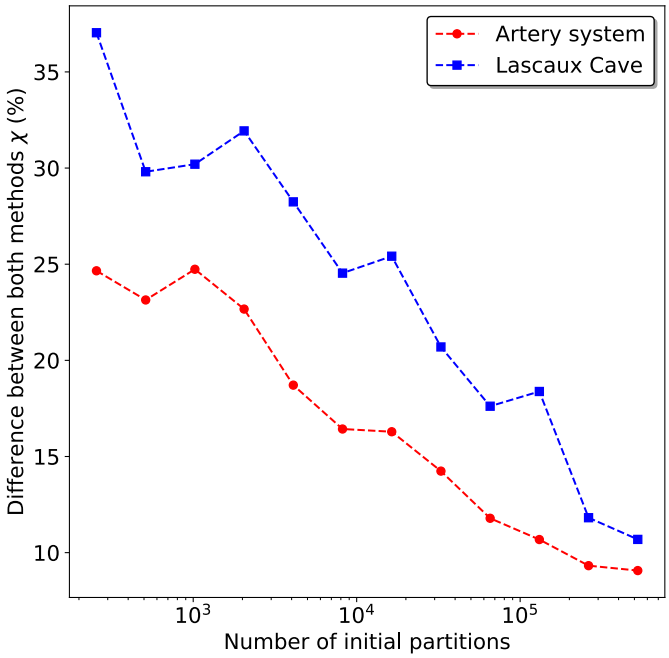

Figure 5: Comparison between the Anupindi method and the proposed algorithm on the arterial system and the Lascaux Cave geometries with a fixed mesh ($\chi = \frac{N^{Anupindi} - N^{Present\ study}}{N^{Anupindi}}$).

decomposition. Let n_{dec} be the number of domain decompositions and n_{proc} the number of processors. Each processor can handle the integer part of $\frac{n_{dec}}{n_{proc}}$ (+1 for some processors) domain decompositions.

We compare in Tab. 2 the computation time for different configurations and numbers of processors. The computational cost of the sliding algorithm scales with the number of iterations of the while-loop. As each iteration removes one partition, the number of iterations is equal to the reduction in the number of partitions between Anupindi's method and the proposed approach. Two large meshes (Lascaux Cave and arterial system) composed of 1 and 24 billion cells are considered. The simulations are performed on Intel Xeon Gold SKL-6130 2.1 GHz processors. The scalability of the algorithm is efficient up to 16 or 32 processors depending on the mesh size. However, the computational time of the domain decomposition remains relatively small considering the mesh size. With 128 processors, it is about 1 minute for both meshes composed of 1 billion cells and between 15 and 19 minutes with 24 billion cells for the Lascaux Cave and the arterial system respectively. As the decomposition domain process is achieved only once at the beginning of the simulations, the computational time of the algorithm remains negligible compared to the total simulation time. This is a clear advantage of the proposed method in comparison with other approaches based on METIS [18]. With METIS, based on the same processors, the domain decomposition of a mesh composed of 50 million cells takes 15 minutes. Meshes as large as those presented in Tab. 2 cannot be managed by Metis and would necessitate the parallel version of Metis. But we remember that parMETIS cannot guarantee contiguous and partitions.

	Lascaux Cave				Arterial system			
Mesh size	1 billion cells		24 billion cells		1 billion cells		24 billion cells	
Number of processors	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)	Efficiency	Time (s)	Efficiency
1	2,080	1	49,920	1	2,440	1	57,310	1
2	1,110	0.93	25,200	0.99	1,220	1	29,000	0.99
4	590	0.89	12,800	0.97	650	0.94	15,000	0.95
8	320	0.82	6,650	0.94	370	0.83	8,450	0.85
16	200	0.65	3,950	0.79	210	0.73	4,800	0.75
32	130	0.52	2,340	0.66	120	0.63	2,700	0.67
64	90	0.37	1,410	0.55	70	0.52	1,600	0.56
128	67	0.24	890	0.44	50	0.37	1,120	0.4

Table 2: Comparison of computational times according to the number of processors, the mesh, and the geometry.

3. Immersion of geometries in Cartesian meshes without degenerate boundary cells

In this section, we are only concerned with geometric objects for which it is possible to define an 'inside' and an 'outside', such as polygons, spheres or polyhedrons. On the contrary, the Klein bottle, the Möbius strip or simply a ribbon cannot separate an inside from an outside and such shapes are therefore excluded from the study. The geometries can be given by surface meshes (CAD files), analytical shapes, Boolean operations between these types of object or geometric transformations.

In the following, we first recall the IBMs and the basis of ghost-cell methods before introducing the immersion algorithms in 2D and 3D.

3.1. Immersed Boundary Methods

As mentioned in the introduction, there are several IBMs to apply boundary conditions thanks to ghost cells. Mittal *et al.* [9] first considered an image point I of the ghost point G (Fig. 6.a). The field value at point I u_I is derived from a polynomial interpolation $\mathcal{P}$ based on the surrounding points P_i:

$$u_I = \mathcal{P}(u_{P_1}, u_{P_2}, u_{P_3}, u_{P_4}) \tag{1}$$

Then, for Dirichlet boundary conditions, an interpolation involving G, B (the boundary point) and I is performed (Fig. 6.a) to compute the field value at the ghost point G. In the case of a linear interpolation,

$$\frac{u_G + u_I}{2} = u_B \tag{2}$$

The Neumann boundary condition is discretised with I, G, and the boundary gradient value $\partial_n u_B$. For a linear interpolation,

$$\frac{u_I - u_G}{||IG||} = \partial_n u_B \tag{3}$$

Instead of introducing an image point, Coco and Russo [8] proposed to compute the ghost value u_G based on a polynomial interpolation at a boundary point B involving surrounding points P_i and G (Fig. 6.b).

For a Dirichlet boundary condition, the boundary field value u_B is given by

$$u_B = \mathcal{Q}(u_{P_1}, u_{P_2}, u_{P_3}, u_G) \tag{4}$$

where $\mathcal{Q}$ is the interpolation polynomial. Equation (4) allows the calculation of the field value u_G with the value at points P_i and at point B which is given by the boundary condition. For Neumann boundary conditions, the differentiation of equation (4) is considered

$$\nabla u_B = \mathcal{R}(u_{P_1}, u_{P_2}, u_{P_3}, u_G) \tag{5}$$

where $\mathcal{R}$ is the algebraic differentiation of the interpolation polynomial $\mathcal{Q}$. The ghost value u_G is then deduced from the boundary condition $\nabla u_B \cdot \mathbf{n} = \partial_n u_B$.

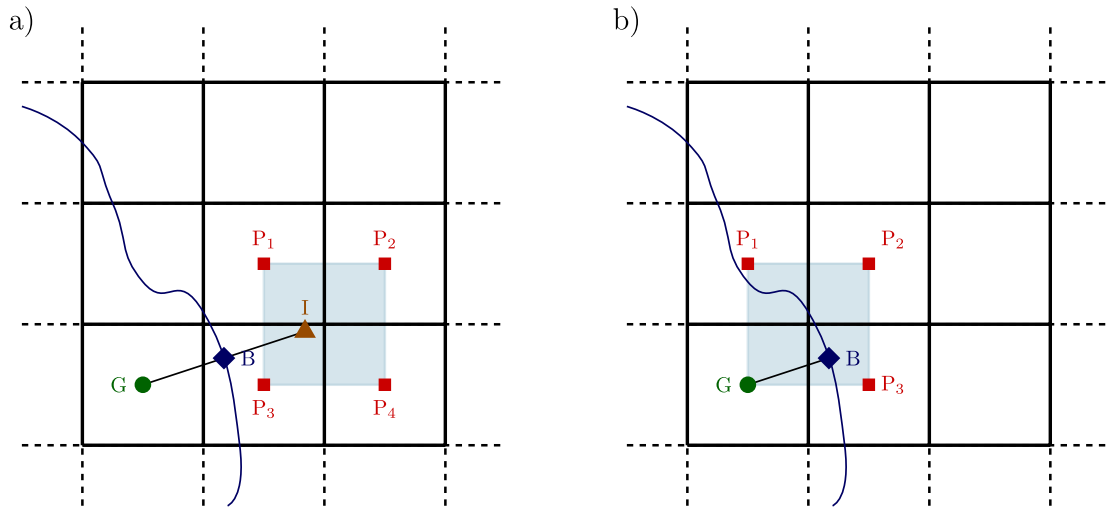

Figure 6: a) Description of the Mittal *et al.* method [9]. b) Description of the Coco and Russo method [8]. In both figures, the points included in the shaded square area are considered in the interpolation.

The stencil size (2 for the Mittal method and 1 for the Coco and Russo method), which increases with the aspect ratio of rectangular cells, is a disadvantage of these methods. Shifting the ghost node or the image point position has been found to yield smaller stencils whilst maintaining convergence [14, 15], the stencil size no longer depending on the aspect ratio of the cells. The simulations achieved in section 4 with Notus [38] are based on the linear ghost-node shifting methods [14, 15]. It should be noted, however, that the choice of these methods does not affect the algorithms presented in the rest of the section.

For the ghost-cell method, it is numerically preferable that boundary cells contain at most one line segment in 2D or one continuous surface in 3D. Cells that do not meet this condition are called degenerate cells. For example, the two shaded cells in Fig. 7 contain more than one isolated line. The central cell shows a tunneling configuration, while the lower left cell corresponds to a split cell. With the ghost-cell method, if the central shaded cell is a ghost cell, its value cannot satisfy the boundary condition on both sides, which can cause numerical instability. One option is to refine the mesh, but one can imagine very small irregularities in the geometry right next to the degenerate cells in Fig. 7, which could involve new degenerate cells with a finer mesh, invisible with the coarsest mesh in Fig. 7. Therefore, deeper refinement

cannot be considered a safe method either.

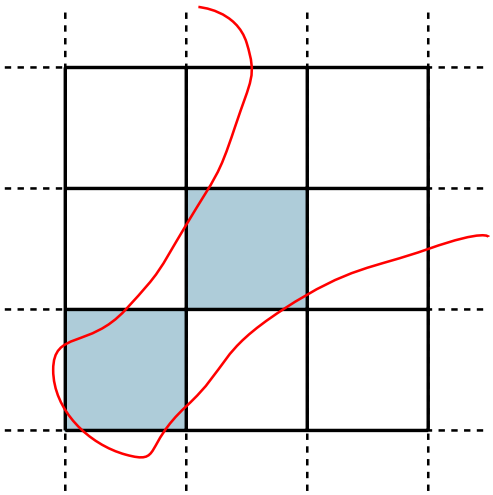

Figure 7: Example of a degenerate case. The shaded cells are degenerate since they contain more than one line.

In the following, we present two algorithms capable of modifying locally the geometry without modifying the grid mesh while avoiding degenerate cells, first in 2D and then in 3D.

3.2. Immersion of 2D geometries

3.2.1. Proposed algorithm

The algorithm consists of two steps. The first step is to reconstruct the contour of the input geometry using line segments. The second step is to modify the reconstructed contour to remove the degenerate cells.

Contour reconstruction. This part of the algorithm is based on ray tracing, a technique widely used in image rendering to simulate rays of light. The rays are traced along the edges of the grid and the intersection points between the rays and the geometry are calculated (Fig. 8). This technique can be used either on geometries described by surface meshes or analytically.

In cells containing two intersection points, a line segment is constructed connecting both points (Fig. 9.a). An exception occurs when both points are on corners. Fig. 9.b displays two different configurations where two intersection points on cell corners must and must not be linked. To circumvent this issue, an arbitrary horizontal or vertical ray is traced through the centre of the cell. If this ray meets the geometry inside the cell, then both corner points must be connected. Otherwise, no line segment will be created.

Cells that contain at least three points necessitate further treatment to connect them to each other. This treatment consists in performing ray tracing recursively within cell subdivisions. These cells are traversed by two orthogonal rays passing through the cell centre (Fig. 10). This creates four sub-cells where the number of intersection points is calculated. If there are two points in a sub-cell, the subdivision process stops since the configuration is the same as previously explained. If there are strictly more than two points, the sub-cell is again divided into four sub-cells at a deeper level. The process continues until all the sub-cells of the initial cells contain no more than two points. Each pair of points can then be connected to sub-line

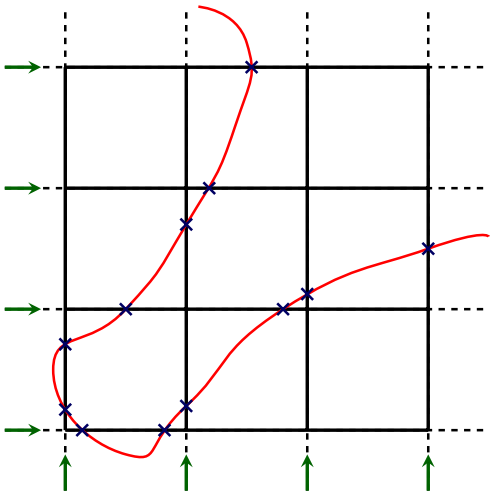

Figure 8: Ray tracing applied to an irregular contour (in red). The rays (green arrows) propagate along the edges of the mesh. The intersection points (blue crosses) are stored.

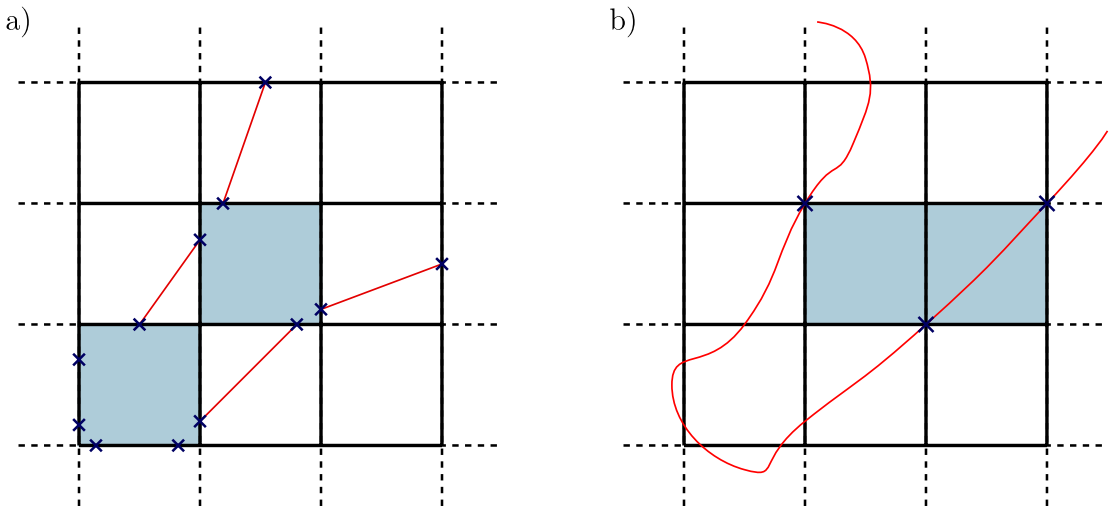

Figure 9: a) The cells containing exactly two points are filled with a line segment linking both points. The cells containing more than two points (shaded cells) cannot be treated so far. b) Two configurations where a cell contains two intersection points on corners: in the central shaded cell, the corner points must not be linked, in the other one, the corner points must be connected with a line segment.

segments in sub-cells (Fig. 10.c), which form paths between the initial intersection points on the edges of the cells. These paths allow the creation of line segments linking the corresponding pair of points (Fig. 10.d). Note that the subdivision process only affects the geometry inside the cell. The remaining parts of the geometry that have not been reconstructed with line segments are modified during the subdivision process in the neighbouring cells that contain these parts.

The first part of the algorithm leads to a final reconstruction of the initial contour from Fig. 11.a, which

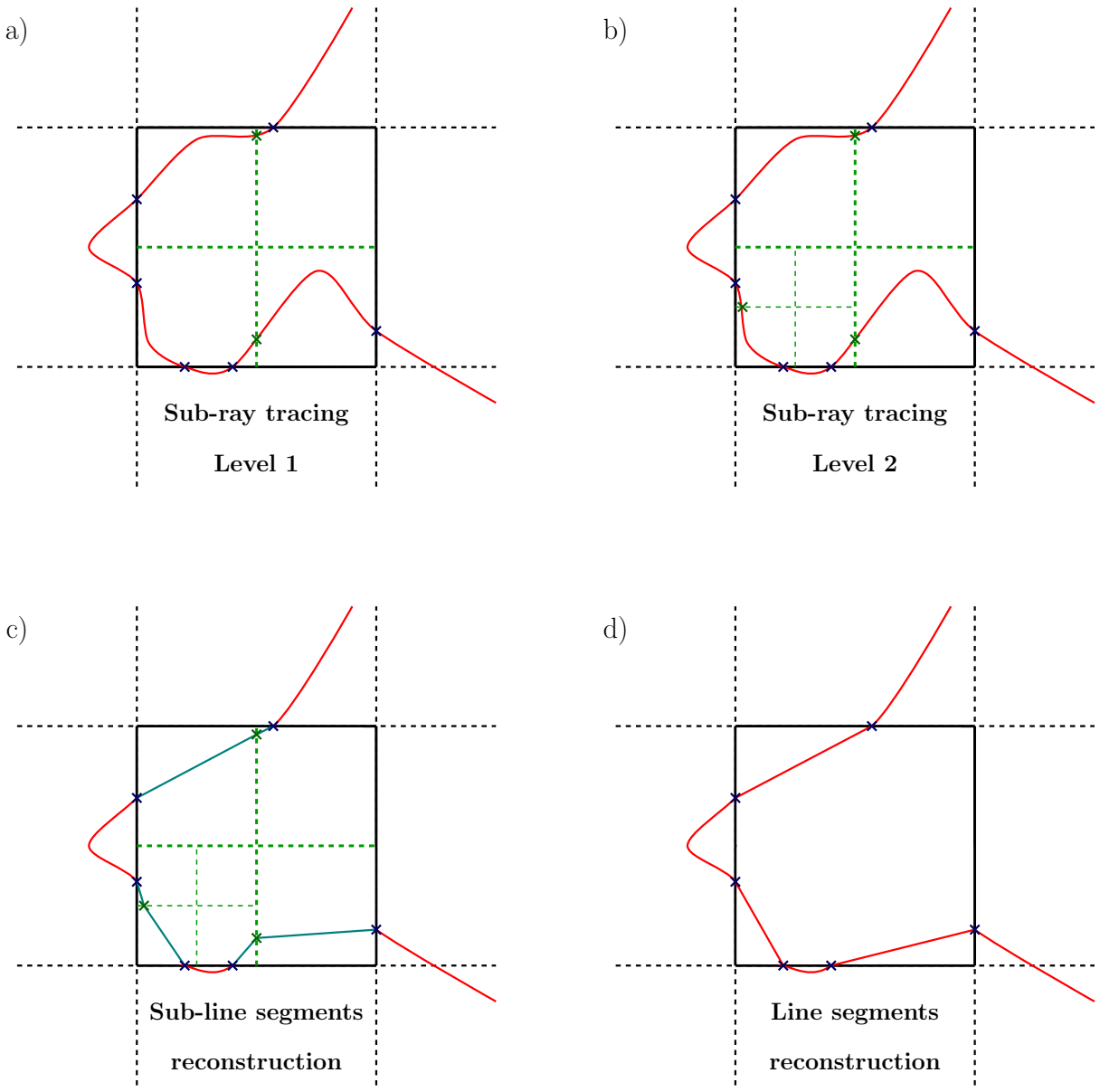

Figure 10: Sub-ray tracing process. a) The cells containing more than three points are subdivided into four sub-cells with two orthogonal rays. b) The sub-cells containing more than three points are subdivided again into four sub-cells until all sub-cells contain less than three points. c) The sub-line segments (indigo lines) are constructed in the sub-cells. d) All the intersection points on the grid edges (blue crosses) that are connected by sub-line segments are linked with a line segment.

looks like the line composed of line segments in Fig. 11.b.

Degenerate cells elimination. The next step enforces cells to contain at most one line segment. Two operations are required depending on the configuration of these cells.

First, cells containing several connected line segments are considered, such as the bottom left cell (1)

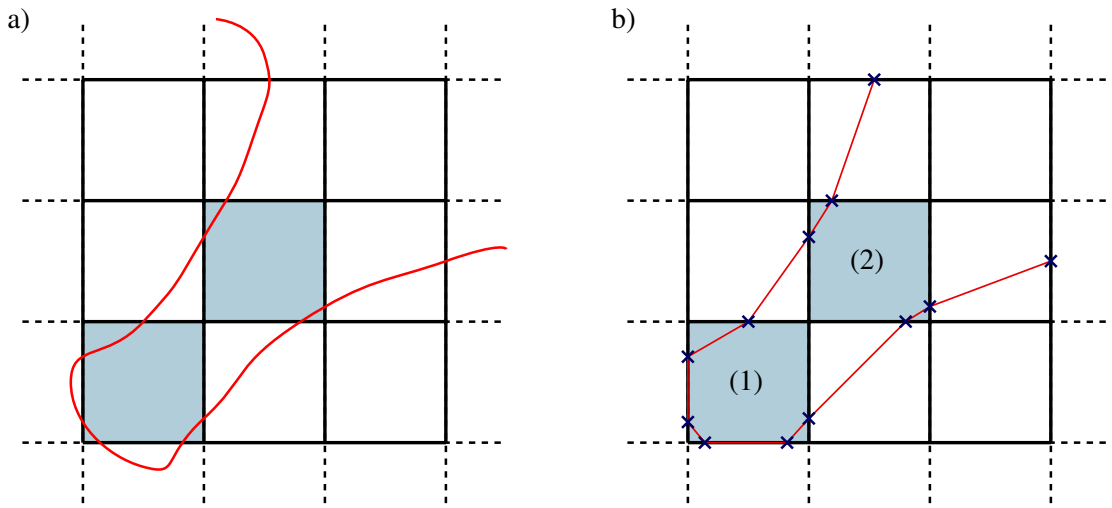

Figure 11: a) Contour line of the initial shape presented in Fig. 7. b) Final contour line at the end of the contour reconstruction algorithm.

in Fig. 11. Such cells are identified with the following condition: there are exactly two points associated to only one line segment within the cell. All line segments within these cells are replaced by a line segment connecting the points initially involved in only one line segment within the cell. Fig. 12 displays the procedure.

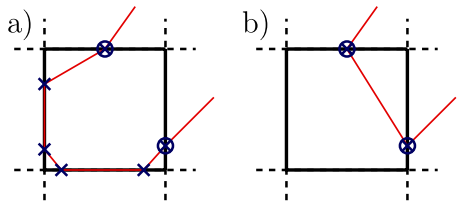

Figure 12: a) Initial configuration with several connected line segments. b) The line segments are removed and replaced by one line segment.

Second, we deal with the remaining cells containing at least two unconnected line segments such as the central cell (2) in Fig. 11. In this case, as detailed below, the geometry is recursively coarsened to remove subgrid scale parts. Super-cells are defined as fictitious cells composed of nine cells, centred on the troublesome cells. Fig. 13.a displays such a super-cell in green (thick contour), which is centered on the shaded cell containing two line segments. We then consider only the intersection points on the super-cell (green crosses in Fig. 13). The intersection points that are connected to line segments inside the super-cell are connected by a super-line segment (Fig. 13.b). The initial line segments are then removed from the nine cells and the super-line segments are divided into as many line segments as the number of cut cells. If there are no longer cell containing more than one line segment among those composing the super-cell, the process ends. Otherwise, the super-cell is recursively expanded to 25 cells (5×5), 49 cells (7×7), etc., and the same procedure is applied until there are no more degenerate cells. This process therefore avoids tunneling and

split cells by coarsening the geometry contour. If an extended super-cell reaches the domain boundaries, this means that the entire geometry is a sub-grid contour. This can happen for very narrow tunnels with a width smaller than the cell size. Such a configuration cannot be handled by the proposed algorithm, but still needs a finer mesh for accurate simulations.

Algorithm 2 itemizes the procedure of the proposed method.

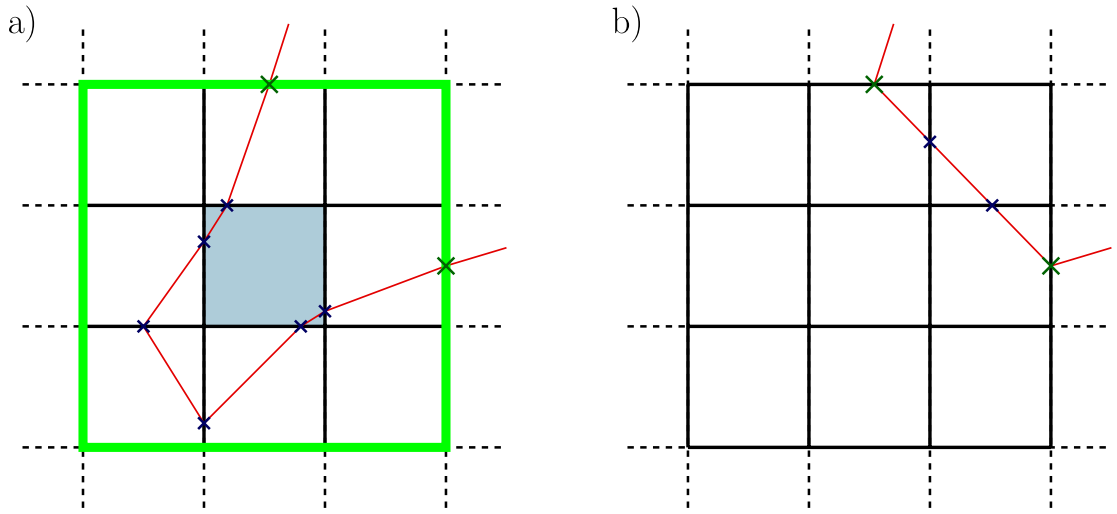

Figure 13: a) Initial state: construction of a super-cell composed of 9 cells. The intersection points between the super-cell and the grid edges (green crosses) are stored. b) Final state: the initial line segments are removed and replaced by line segments connecting the corresponding intersection points located on the super-cell.

3.2.2. Examples

We present a first example of an irregular smooth contour (Fig. 14). The geometry consists of two smooth wavy lines without narrow part, so no degenerate cell is expected. Figure 14 shows that the precision of the reconstructed contour increases with the mesh refinement as the number of line segments composing the contour depends on the grid size.

Fig. 15 displays the contour reconstruction of an irregular polygon with sharp angles. Without special treatment, this geometry leads to degenerate cells regardless of the mesh resolution. The presented algorithm basically smooths the initial geometry and cuts out the sharp areas. The super-cell process can coarsely modify the initial geometries, such as the circled zone in Fig. 15. One could expect the sharp angle to be less cut. However, we have chosen to build 3×3 super-cells centered on degenerate cells, which involves rougher cuts. Off-center 2×2 super-cells would impact the geometries more shallowly. However, the choice between four possible constructions of 2×2 super-cells in 2D (eight in 3D) is beyond the scope of this article.

It is worth noting that mesh refinement does not reduce the number of critical zones since the super-cell process was only required for three cells for the coarsest mesh, one cell for the medium mesh and two cells for the finer mesh. With a fine mesh, there are fewer subgrid parts in the geometry, but there are more cells. The number of cells requiring the super-cell process is therefore the result of these two opposing trends.

Algorithm 2 Single-line segment cut cells

Input Exact 2D geometry contour

Contour reconstruction
for each grid edges direction **do**
trace rays and store the intersection points with the geometry (Fig. 8)
end for
for each cell containing two points **do**
if both points are on grid corners **then**
trace a ray through the center of the cell (Fig. 9.b)
if there is an intersection with the geometry **then**
create a line segment between the two corner points
end if
else
create a line segment with the two points (Fig. 9.a)
end if
end for
for each other cell **do**
trace two orthogonal rays through the center of the cell (Fig. 10.a)
while a sub-cell contains more than two intersection points **do**
perform sub-ray tracing in the troublesome sub-cells (Fig. 10.b)
end while
reconstruct the sub-line segments inside the sub-cells containing two points (Fig. 10.c)
reconstruct the line segments in the cell based on the sub-line segments (Fig. 10.d)
end for

Degenerate cells elimination
for each cell containing connected line segments **do**
remove the line segments
replace them by one line segment connecting the points initially involved in only one line segment (Fig. 12)
end for
for each cell still containing more than one line segment **do**
$n = 3$
construct a super-cell composed of 3×3 cells around the cell (Fig. 13.a)
while a cell in the super-cell contains more than one line segment **do**
if $n > 3$ **then**
construct a super-cell composed of $n \times n$ cells
end if
store the intersection points between the super-cell and the grid edges
remove the internal line segments
create the line segments connecting the associated intersection points located on the super-cell (Fig. 13.b)
$n = n + 2$
end while
end for

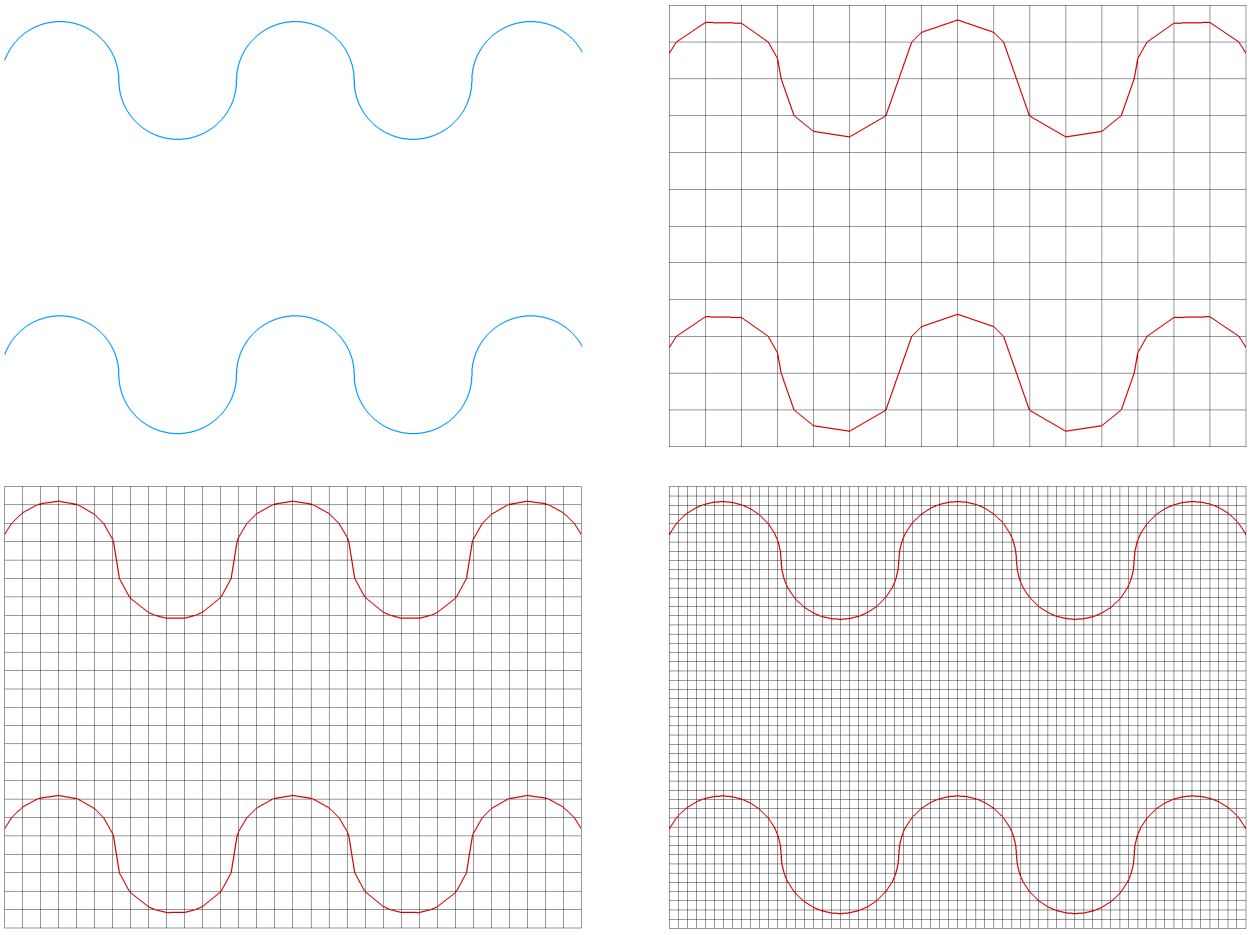

Figure 14: Contour reconstruction of two wavy lines according to the grid meshes.

However, as expected, the effect of the cutting process on the final geometry lessens with increasing mesh size, since the finest geometry is close to the expected initial contour.

The convergence of the proposed method is assessed for these two test cases by evaluating the symmetric difference under successive mesh refinements. Let A and B denote two sets. The symmetric difference is defined as the volume (or area in 2D) of the symmetric difference

$$A \triangle B = (A \setminus B) \cup (B \setminus A) \tag{6}$$

where $\setminus$ denotes the set difference. It therefore measures the portion of the domain occupied by one reconstruction but not the other. As the mesh is refined, the symmetric difference area between reconstructions obtained on successive mesh levels is expected to decrease, providing evidence of the convergence of the reconstruction algorithm. Table 3 reports the symmetric difference area between each mesh and a refined reference mesh consisting of 16,384 cells for the wavy-line case and 102,400 cells for the irregular polygon case. The relative symmetric difference area is defined as the symmetric difference area normalized by the

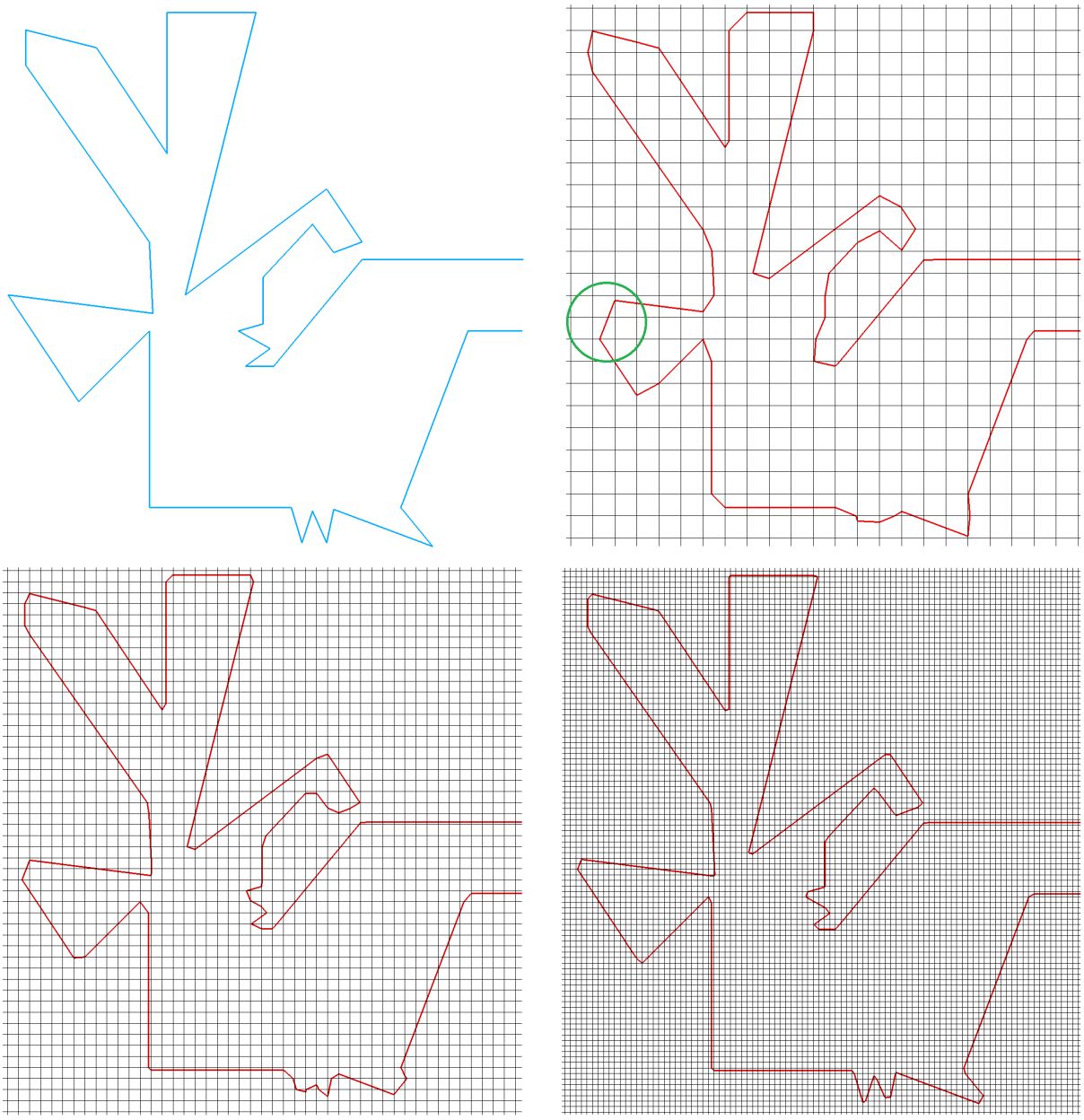

Figure 15: Contour reconstruction of an irregular polygon depending on the grid meshes. The part of the polygon circled in green is roughly cut during the super-cell process.

total area enclosed by the contour, equal to 0.5 for the wavy-line case and 0.92 for the irregular polygon case. For both geometries, the symmetric difference area decreases consistently as the mesh resolution increases, demonstrating that the changes between two consecutive reconstructions become progressively smaller. Although the irregular polygon exhibits larger differences on coarse meshes due to its geometric complexity, the relative symmetric difference area decreases to only 0.5% on the finest mesh, while it reaches 0.15% for the wavy lines. The convergence analysis reveals that the method achieves an approximate order of 1.8

for the wavy lines and 1.9 for the irregular polygon, indicating a near-second-order accuracy in both cases. These results confirm the robustness and convergence of the proposed reconstruction method.

Wavy lines			Irregular polygon		
Number of cells	Symmetric difference area	Relative area	Number of cells	Symmetric difference area	Relative area
256	0.0091	1.8%	1,600	0.067	7.3%
1,024	0.0025	0.5%	6,400	0.016	1.7%
4,096	0.0007	0.15%	25,600	0.0044	0.5%

Table 3: Symmetric difference area between contour reconstructions from each mesh and a refined reference mesh for the wavy lines and irregular polygon test cases. The relative area is defined as the ratio of the symmetric difference area to the total area enclosed by the contour.

3.2.3. Parallelization of the algorithm

The contour reconstruction is fully parallel since each MPI process deals with the part of the geometry in its own subdomain. The second part of the algorithm on the elimination of the degenerate cells can partially be executed in parallel since the super-cells can spread over boundaries between partitions and be composed of cells belonging to different processes. The degenerate cells far away from the processor boundaries could be treated by the corresponding processes, but this procedure would not always give the same final geometry. Indeed, the order in which the degenerate cells are treated does affect the results if there are close degenerate cells. In such cases, the neighboring cells involved in the super-cells could be different depending on whether the other close degenerate cells have already been treated. Each cell within super-cells must therefore be gathered by the main process, which applies the algorithm and then sends the results to the involved processes. Nevertheless, this sequential part of the algorithm has a small impact on the computation time as it is shown below.

Tab. 4 shows the computation time of the proposed algorithm on both previous examples. Two Cartesian meshes composed of 1 and 100 million cells are considered. For both, the computation time is smaller than 5 seconds, even for sequential runs. Besides, the algorithm is efficiently scalable up to 256 processors. Therefore, the proposed algorithm seems optimised and takes a negligible part in the simulations.

3.3. Immersion of 3D geometries

3.3.1. Algorithm

The 3D algorithm also consists of surface shape reconstruction followed by degenerate cell elimination. However, the detection of degenerate cells is more complex in 3D and is thus further detailed.

Surface shape reconstruction. As in 2D, the ray tracing process is applied on the grid edges. Then the line segments are created on the faces and the 2D contour reconstruction algorithm is applied to the faces using identical rules for the number of intersection points. The surface shape is thus created based on line segments. For each cut cell, the line segments on the six faces are stored. Fig. 16.a displays an example with 10 line segments. The next step is to construct all polygonal faces from the list of line segments. The graph associated with the list of line segments is constructed first. Based on the depth-first search algorithm, all the possible cycles in the graph are stored. From this list, only the simple cycles (cycle that does not contain any other cycle) are kept, without duplicate. This list of simple cycles corresponds to the list of polygons

	Smooth wavy lines				Sharp polygon			
Mesh size	1 million cells		100 million cells		1 million cells		100 million cells	
Number of processors	Time	Efficiency	Time	Efficiency	Time	Efficiency	Time	Efficiency
1	4.4 s	1	-	-	3.95 s	1	-	-
2	2.24 s	0.98	-	-	2 s	0.99	-	-
4	1.15 s	0.96	-	-	1 s	0.99	-	-
8	0.58 s	0.95	-	-	0.51 s	0.97	-	-
16	0.3 s	0.91	-	-	0.25 s	0.99	-	-
32	0.15 s	0.9	-	-	0.15 s	0.82	-	-
64	-	-	7.4 s	1	-	-	6 s	1
128	-	-	3.8 s	0.97	-	-	3.1 s	0.97
256	-	-	1.9 s	0.97	-	-	1.7 s	0.88

Table 4: Comparison of calculation times according to the number of processors, the mesh, and the geometry.

generated by the line segments in the cell. In the example in Fig. 16.a, three polygons are calculated, consisting of the following points $(1, 2, 4, 3)$, $(3, 4, 6, 5)$, and $(5, 6, 8, 7)$.

Detection of degenerate cells. The cells containing several lonely polygons are degenerate, since it is not possible to construct only one continuous boundary face. These cells are therefore gathered in the list of degenerate cells.

Another kind of degenerate cell can appear during the process. If there are subgrid scale parts in the geometry that do not cut any edges, ray tracing in 3D can miss the geometry as shown in Fig. 16.b. Around this cell, if the geometry expands with a characteristic size greater than the cell size, ray tracing will intersect the geometry again, resulting in the construction of isolated line segments. The 3D algorithm introduces these additional difficulties that do not occur in 2D.

Two criteria are used to detect these cells. Since only closed surfaces are considered, if a point of the reconstructed surface shape is connected to less than three line segments, then the cells containing this point are added to the list of the degenerate cells. If a line segment belongs to only one polygon, then the surface shape is open at this point. The cells containing this line segment are added to the list of degenerate cells, which are then subjected to the following elimination process.

Elimination of degenerate cells. In 2D, we had two different ways of dealing with degenerate cells, depending on whether they contained connected line segments. In 3D, the process associated with the connected line segments is no longer considered. In fact, deleting a point in 2D only affected the two connected line segments, but in 3D, each point is linked to more than three line segments. Managing this process in 3D is therefore much more difficult and beyond the scope of this article. We will therefore only consider the process associated with the construction of super-cells, which are initially composed of $3 \times 3 \times 3$ cells. First, the points and the line segments at the boundary of this super-cell are stored (Fig. 17.a). As with the detection of degenerate cells, the list of possible polygons is calculated from the list of line segments. Then, a triangulation of the polygons is performed to create surfaces from the contours. Basically, a polygon embedded in the 3D space can be triangulated in many ways. All the possible triangulations of the polygons are

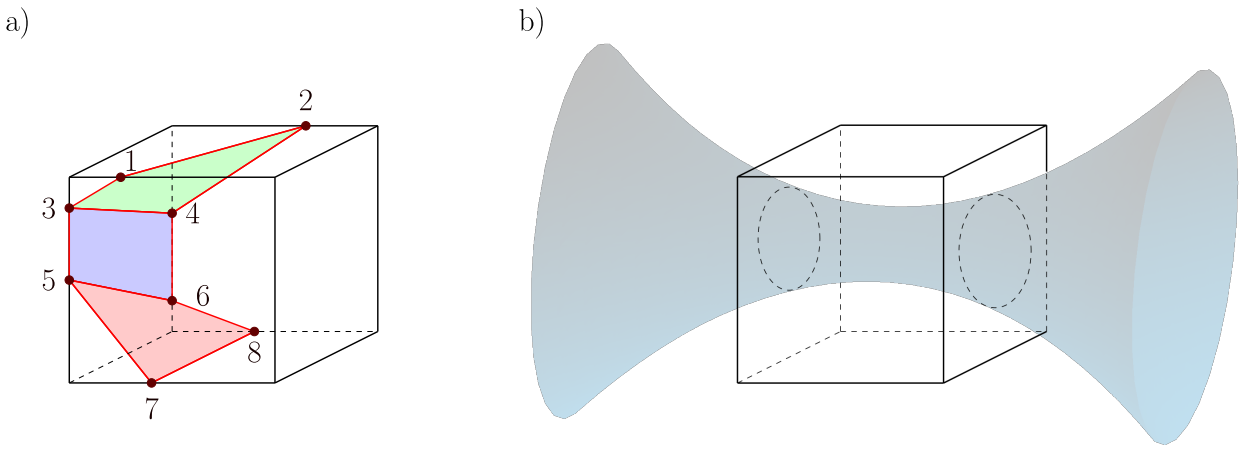

Figure 16: a) Reconstructed line segments on the six faces of the cell. b) Local subgrid-scale geometry that passes through a cell without intersecting its edges.

computed using a recursive subdivision until only triangles remain. Among all the possible triangulations, we choose the one that minimises the surface area, in order to have a surface as flat as possible. Fig. 17.b shows the triangulation of the contours. The intersection points between the created surfaces and the grid edges inside the super-cell are then calculated (Fig. 17.c). As with the surface shape reconstruction, the line segments are constructed (Fig. 17.d) and a detection of degenerate cells in the super-cell is performed. If there are degenerate cells, the super-cell is expanded recursively by one cell in each direction so that in the second step it consists of $5 \times 5 \times 5$ cells. The process ends when there are no more degenerate cell in the super-cell. It should be noted that when the super-cells become large (around $25 \times 25 \times 25$ cells), the triangulation of the contours (as in Fig. 17.b) can lead to severe issues. In these cases, the choice of the triangulation, based on the minimum area criterion, should probably be improved. In particular, triangulations composed of long and thin triangles should be prohibited. If such a case occurs, a loop over the triangulations from the smallest to the largest surface area can be achieved until a suitable configuration is found.

Algorithm 3 outlines the whole 3D procedure of the proposed method.

3.3.2. Examples

We present the reconstruction of the surface shape of a pumpkin for different mesh sizes (Fig. 18). The pumpkin is regular enough that the shape reconstruction is weakly dependent on the cell size. Then, there is almost no difference between the two finest meshes and a slight deviation for the coarsest mesh. Despite the smooth shape of the pumpkin, the coarsest mesh required the creation of five super-cells. Eight and ten super-cells were created for the medium and the finer meshes respectively. The refinement of the mesh therefore does not avoid degenerate cells, as already shown in 2D (§ 3.2.2). As described in algorithm 3, the super-cell approach recursively increases the number of cells composing the super-cells. The super-cells are composed on average of 32, 102, and 51 cells for the coarsest, medium, and finest meshes respectively. As with the number of super-cells, their size does not decrease with refinement. Thus, refinement does not necessarily render the super-cell process useless.

Figure 19 displays the application of the algorithm to a surface representing a rabbit (Stanford bunny).

Algorithm 3 Single-face cut cells

Input Exact 3D triangulated surface

Surface shape reconstruction

apply the 2D contour reconstruction algorithm to the faces of the cells

Degenerate cells detection

for each cut cell **do**
store the list of line segments and intersection points in the cell
construct the graph associated with the line segments and the intersection points
compute the number n_{cycles} of simple cycles in the graph
if $n_{cycles} > 1$ **then**
the cell is added to the list of the degenerate cells
end if
end for
for each intersection point **do**
the point is involved in $n_{linesegments}$ line segments in the surface shape
if $n_{linesegments} < 3$ **then**
the cells containing this point are added to the list of the degenerate cells
end if
end for
for each line segment **do**
the line segment is involved in n_{faces} polygonal faces in the surface shape
if $n_{faces} < 2$ **then**
the cell containing this line segment is added to the list of the degenerate cells
end if
end for

Degenerate cells elimination

for each cell in the list of the degenerate cells **do**
$n = 3$
construct a super-cell composed of $3 \times 3 \times 3$ cells around the cell (Fig. 17.a)
while there is a degenerate cell inside the super-cell **do**
if $n > 3$ **then**
construct a super-cell composed of $n \times n \times n$ cells
end if
store the intersection points and the line segments located on the super-cell
remove the internal line segments
triangulate the contour composed of the line segments on the super-cell (Fig. 17.b)
count the intersection points between the grid edges and the surface
create the new line segments
$n = n + 2$
end while
end for

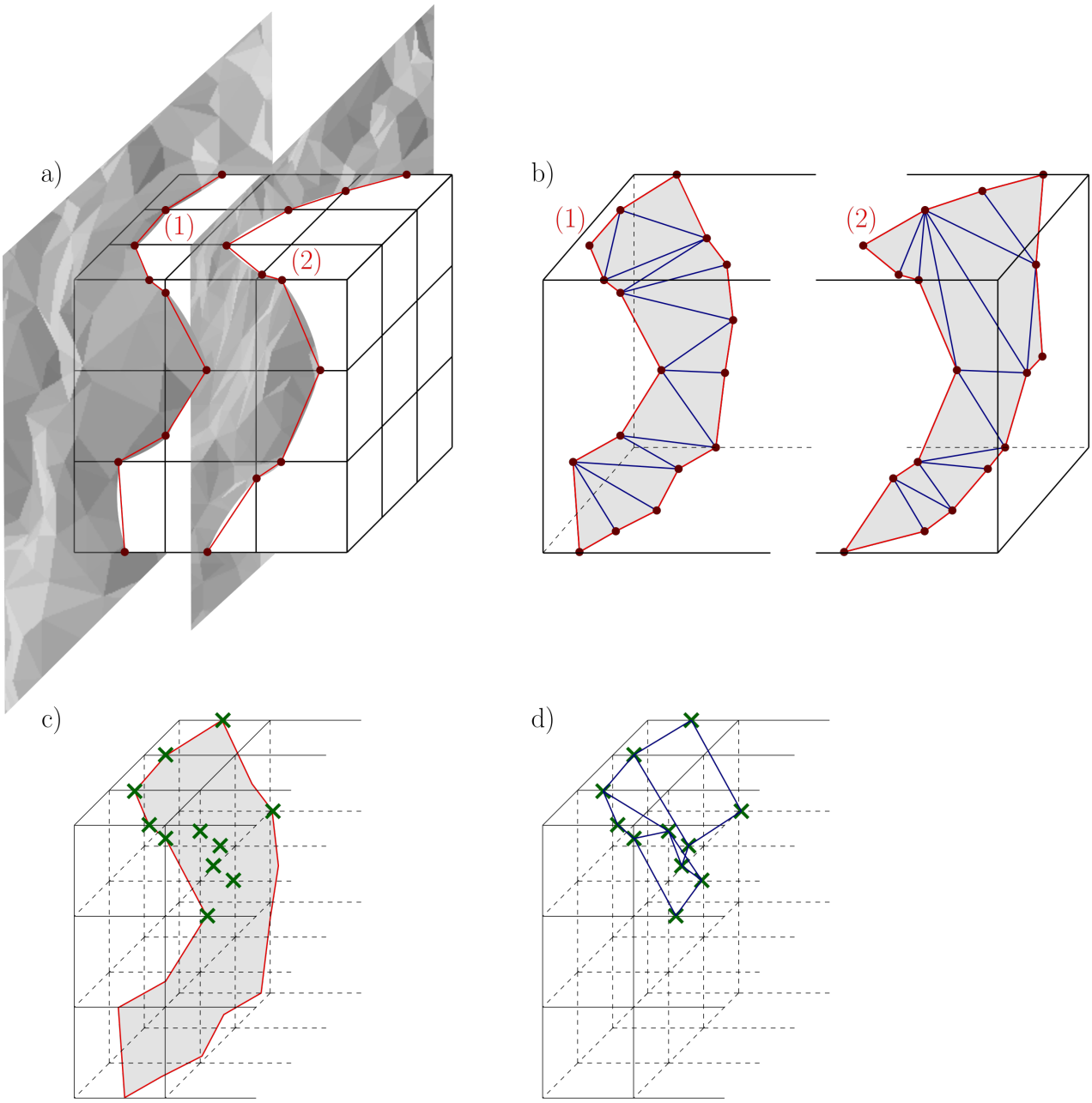

Figure 17: a) Construction of a super-cell composed of 27 cells. The intersection points (dark red dots) and the line segments (red lines) at the boundary of the super-cell are stored. b) Triangulation of the contours and construction of two surfaces intersecting the grid edges inside the super-cell. c) Computation of the intersection points (green crosses) between the triangulated surface and the grid edges. For the sake of clarity, only the intersection points in the top cells are presented. d) Reconstruction of the line segments in each face of the upper cells, which contains at most two points.

For the coarsest mesh, the thickness of the ears becomes smaller than the cell size and they are removed from the final geometry by the algorithm. The number of details increases as the mesh is refined. Seven,

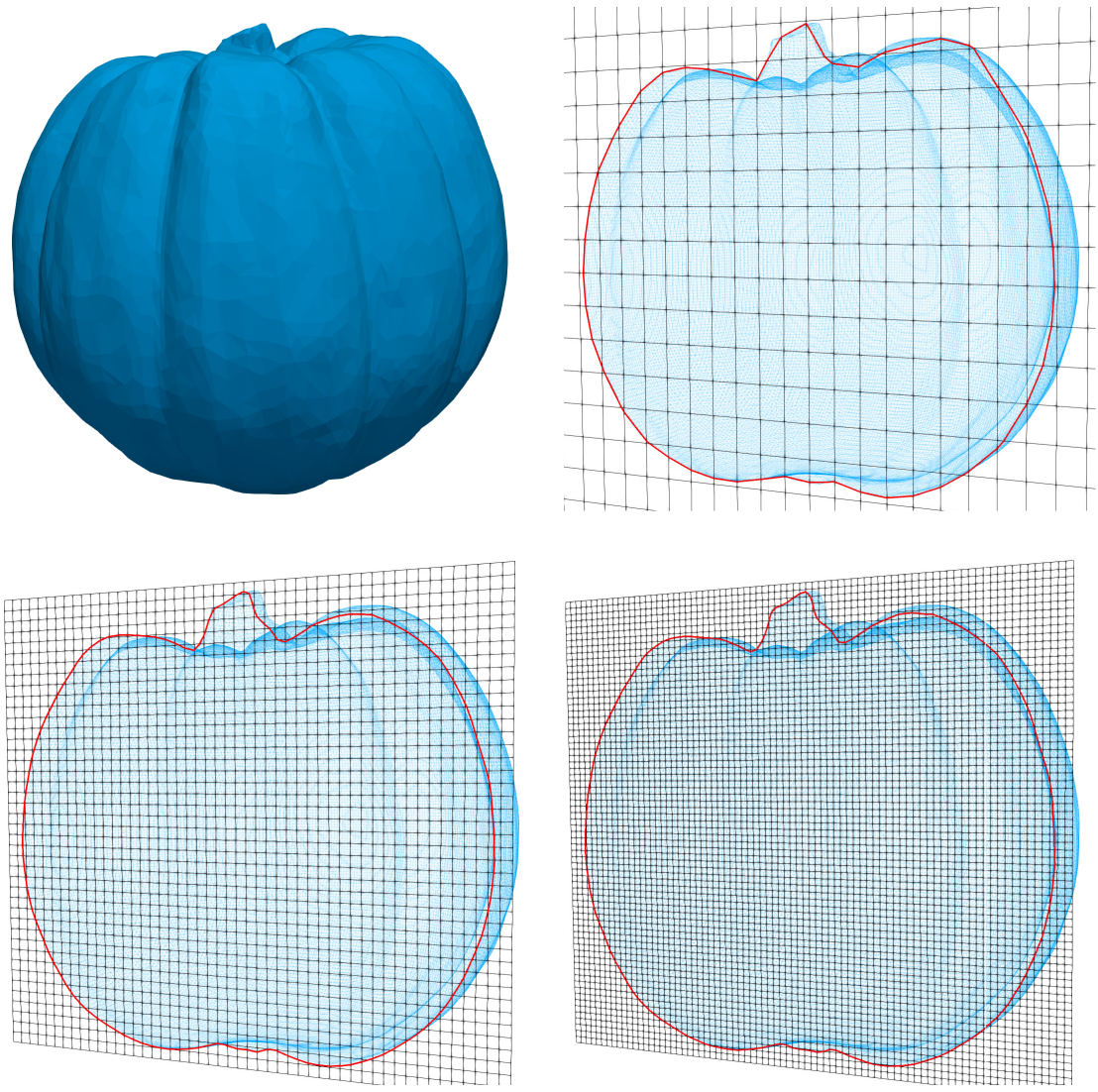

Figure 18: Surface shape reconstruction of a pumpkin according to the grid meshes. The red line is a 2D contour of the reconstructed pumpkin in a slice.

three and twelve super-cells are created from the coarsest to the finest meshes respectively, confirming that refinement is not always a solution to degenerate cells. These super-cells are composed of an average of 27, 68, and 72 cells for the three meshes. The behaviour for the Stanford bunny is similar to that for the pumpkin.

As in the two-dimensional case, Table 5 demonstrates the convergence of the proposed reconstruction algorithm through the progressive reduction of the symmetric difference volume between each mesh and a refined mesh consisting of 3,502,080 cells for the pumpkin and 2,918,400 cells for the rabbit. For both the pumpkin and rabbit geometries, the relative difference decreases by more than one order of magnitude, reaching only 0.08% and 0.2%, respectively, on the finest meshes. The convergence analysis yields an observed convergence order of approximately 2.0 for the pumpkin case and 1.85 for the rabbit case, confirm-

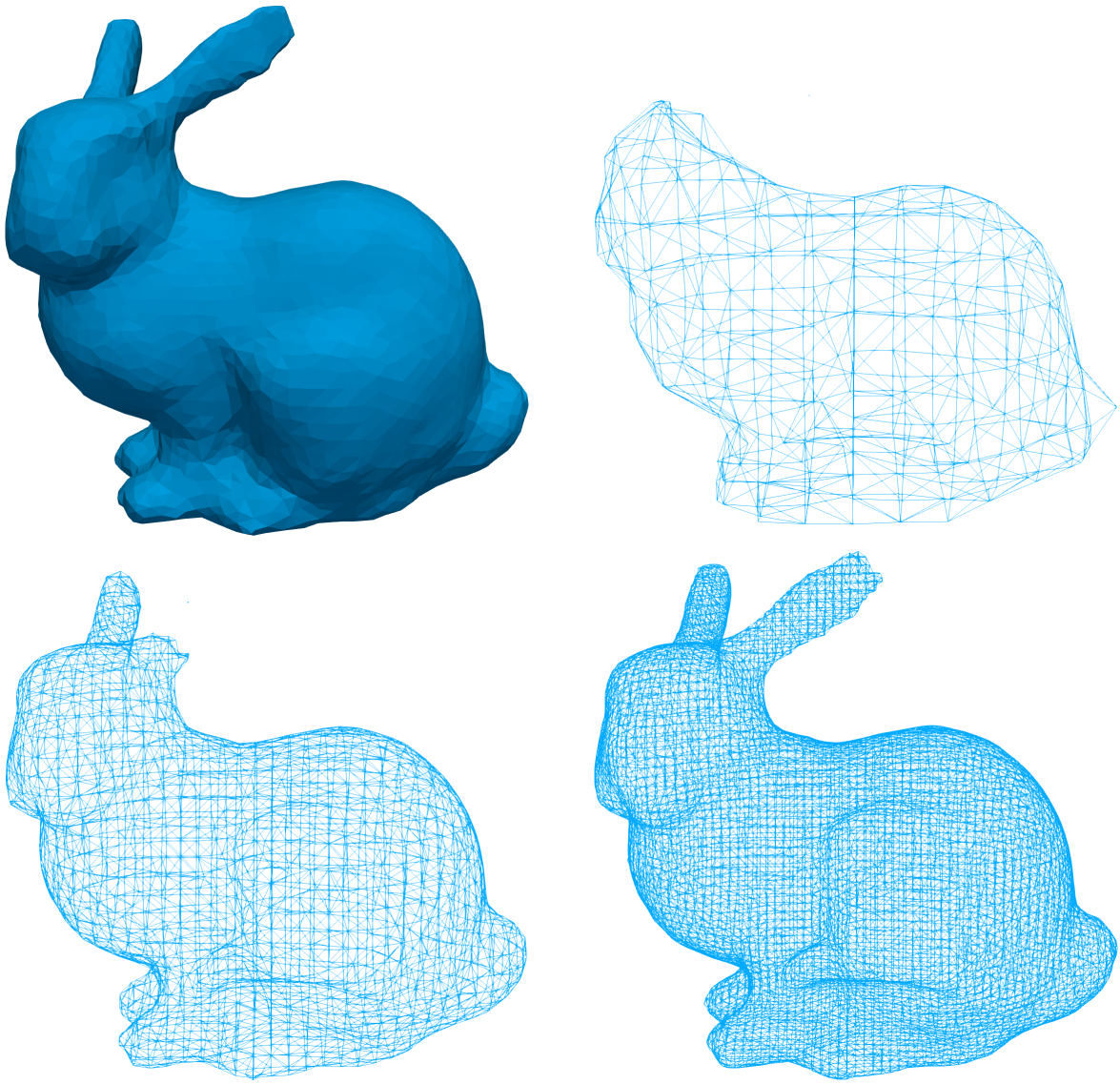

Figure 19: Surface shape reconstruction of a rabbit according to the grid meshes.

ing that the proposed method exhibits near-second-order accuracy for very different 2D and 3D geometries. These results therefore confirm that the reconstruction converges consistently under mesh refinement in three dimensions as well.

3.3.3. Parallelization of the algorithm

As in 2D, the first part of the algorithm can be run in parallel, while the second part is inherently sequential, since the 3D super-cells can spread out in several partitions.

Tab. 6 displays the computation time required by the proposed 3D algorithm to construct four different shapes. Four meshes composed of 150 million and 1.2 billion cells are considered. As in 2D, the compu-

Pumpkin			Rabbit		
Number of cells	Symmetric difference volume	Relative volume	Number of cells	Symmetric difference volume	Relative volume
6,840	2932	1.3%	5,700	10720	2.9%
54,720	746	0.3%	45,600	3204	0.9%
437,760	184	0.08%	364,800	812.4	0.2%

Table 5: Symmetric difference volume between surface shape reconstructions on increasingly refined meshes for the pumpkin and rabbit test cases. The relative volume is defined as the ratio of the symmetric difference volume to the total volume enclosed by the surface.

tation time is very small. The algorithm is efficient for the Stanford bunny geometry (70% between 256 and 4096 processors). However, for the pumpkin, the computation time does not decrease much with the number of processors. This behaviour is due to the sub-ray tracing part of the algorithm. A zone close to the pumpkin stem necessitates a very deep level of sub-ray tracing, which affects the computation time of the associated processor. However, the CPU time is less than a minute for 1.2 billion cells, which does not significantly impact the overall simulation time. The arterial system and the Lascaux Cave need the creation of more super-cells than the first two geometries (82 and 53 respectively). Adding more processors reduces the time to a critical point, which is particularly highlighted by the arterial configuration. For a mesh of 150 million cells, the parallelization efficiency starts to decrease from 2048 processors. The sequential part of the algorithm takes 31 seconds, while the total computation time is 35 seconds with 4096 processors. In such a case, increasing the number of processors has thus only a negligible effect. Given the small amount of computation required by the algorithm, such a limitation is not critical.

	Stanford bunny				Pumpkin			
Mesh size	150 million cells		1.2 billion cells		150 million cells		1.2 billion cells	
Number of processors	Time	Efficiency	Time	Efficiency	Time	Efficiency	Time	Efficiency
256	56 s	1	-	-	94 s	1	-	-
512	34 s	0.82	-	-	88 s	0.53	-	-
1024	18 s	0.78	-	-	75 s	0.31	-	-
2048	9 s	0.77	83 s	1	68 s	0.17	58 s	1
4096	5 s	0.7	36 s	1.15	64 s	0.09	44 s	0.66
	Arterial system				Lascaux Cave			
Mesh size	150 million cells		1.2 billion cells		150 million cells		1.2 billion cells	
Number of processors	Time	Efficiency	Time	Efficiency	Time	Efficiency	Time	Efficiency
256	170 s	1	-	-	194 s	1	-	-
512	91 s	0.93	-	-	150 s	0.65	-	-
1024	53 s	0.8	-	-	95 s	0.51	-	-
2048	41 s	0.52	157 s	1	51 s	0.48	317 s	1
4096	35 s	0.3	84 s	0.93	29 s	0.42	192 s	0.83

Table 6: Comparison of calculation time according to the number of processors, the mesh and the geometry.

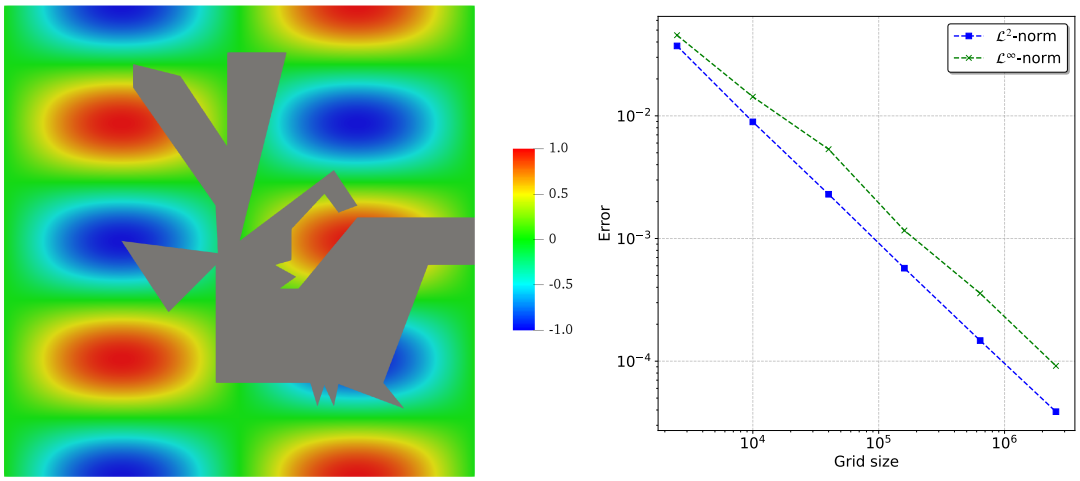

Figure 20: On the left, numerical solution of the Poisson problem for the finest mesh composed of 2,560,000 cells. On the right, error between the numerical and analytical solutions.

4. Validations & Applications

In this section, we present two validation cases and two application examples of the proposed algorithms: a Poisson problem defined on the irregular polygon geometry, flow around a triangular cylinder, two-dimensional flow through a porous medium, and three-dimensional hemodynamic flow in an arterial system. For these simulations, the boundary conditions are applied with the linear ghost-cell method [15] (see section 3.1) based on the proposed algorithm (§3). Both the advection and diffusion terms are discretized with the implicit second-order centered scheme. The velocity-pressure coupled problem is solved with the Timmermans method [39]. The simulation is achieved by Notus CFD [38] where the linear systems are solved with the preconditioned (BoomerAMG) biconjugate gradient stabilized method (BiCGSTAB) with the hypre library [16].

4.1. Poisson problem

This section investigates the convergence of the numerical solution of a Poisson problem defined on the domain associated with the irregular polygon. A Dirichlet boundary condition, $u(x, y) = \sin(\pi x)\cos(2\pi y)$ is prescribed on the polygon boundary, while the source term is $\mathcal{S} = -5\pi^2 \sin(\pi x)\cos(2\pi y)$. Homogeneous Neumann boundary conditions are imposed on the top and bottom boundaries, whereas homogeneous Dirichlet boundary conditions are prescribed on the left and right boundaries [15].

Figure 20 presents the numerical solution on the finest mesh together with the corresponding $\mathcal{L}^2$ and $\mathcal{L}^\infty$ errors. The convergence study confirms the expected second-order convergence rate with respect to the cell size. Therefore, the proposed geometry reconstruction algorithm does not alter the convergence properties of the linear ghost-cell method, demonstrating that the reconstructed geometry is sufficiently accurate to preserve the second-order spatial accuracy.

4.2. Flow around a triangular cylinder

This section validates the proposed ghost-cell immersed boundary method, combined with the contour reconstruction algorithm, by comparison with reference numerical results obtained using a body-fitted mesh

[40]. Three benchmark configurations are considered. The computational mesh is refined such that approximately 38 cells discretize one edge of the triangular cylinder.

Figure 21.a presents the instantaneous vorticity field for a triangular cylinder rotated by 30° (using the same angle convention as Ng et al. [40]) at a Reynolds number of $Re = 200$. The characteristic von Kármán vortex street generated by the periodic vortex shedding is clearly observed. It is also observed in Figures 21b and 21c, which show the instantaneous vorticity field around triangular cylinders rotated by 60° and 0°, respectively, for $Re = 100$.

The mean lift and drag coefficients obtained for the three configurations are compared with the reference results of Ng *et al.* [40]. For the first configuration, the present method predicts $\overline{C_L} \approx -1.50$ and $\overline{C_D} \approx 2.00$, in excellent agreement with the reference values of $\overline{C_L} \approx -1.49$ and $\overline{C_D} \approx 2.00$. In the second and third configurations, the geometry is symmetric with respect to the flow direction, and the mean lift coefficient is therefore expected to vanish. The computed values are of the order of 10^{-3}, confirming this behavior. The predicted mean drag coefficients are $\overline{C_D} \approx 2.12$ for the 60° configuration, compared with the reference range of 2.05–2.12, and $\overline{C_D} \approx 1.72$ for the 0° configuration, compared with the reference range of 1.70–1.76. The Strouhal number can also be compared with the reference results reported by Ng *et al.* [40] for the three configurations. For the first configuration, the predicted Strouhal number is approximately 0.17, in good agreement with the values reported by Ng *et al.* for different mesh resolutions. For the second and third configurations, the computed Strouhal numbers are approximately 0.15 and 0.20, respectively, which compare well with the reference ranges of 0.1529-0.1540 and 0.1946-0.1966 reported in [40]. These results demonstrate that the proposed contour reconstruction algorithm preserves the accuracy of the ghost-cell immersed boundary method and yields results comparable to those obtained with body-fitted meshes.

4.3. Fluid flow through a porous medium

Flow in porous media is involved in many fields such as underground hydraulics [41], oilfields exploitation [42], chemical industry [43] or problems related to groundwater pollution [44]. The geometry is composed of randomly positioned circles with randomly chosen radii. The porous domain is 2 cm long and 1 cm high. The use of the algorithm for the immersion of geometries in Cartesian meshes is particularly needed for this complex geometry. Approximately 2,000 super-cells have been created during the reconstruction of the geometry to avoid degenerate cells. The steady-state incompressible Navier-Stokes equations has been solved with water properties ($\nu = 10^{-6}$ m^2.s^{-1}). The inlet velocity is equal to 1 cm.s^{-1} and is imposed on the right boundary, which is positioned 2 mm upstream of the porous area to avoid significant influences on the inlet velocity field. The left boundary condition corresponds to an outlet which is also located 2 mm beyond the porous zone. Wall boundary conditions are applied to the top and the bottom of the domain. The mesh consists of 80 million cells. Figure 22 displays the simulated velocity field through the porous medium and the simulated pressure.

Preferential paths emerge in the velocity field. In fact, main veins are naturally formed according to the local diameter of the channels. The constriction of the preferred flow areas thus induces significant velocity increases. Despite an inlet velocity of 1 cm.s^{-1}, the flow locally reaches velocities of the order of 10 cm.s^{-1}. The pressure difference between the inlet and the outlet is approximately 200 Pa. As expected, the isobars are not strictly vertical but depend on the local distribution of the grains.

4.4. Haemodynamics in an arterial system

The simulation presented in this section is based on the geometry of the arterial system shown in Fig. 3. This kind of problem is of great interest in the medical field [45, 46]. The geometry is very irregular since

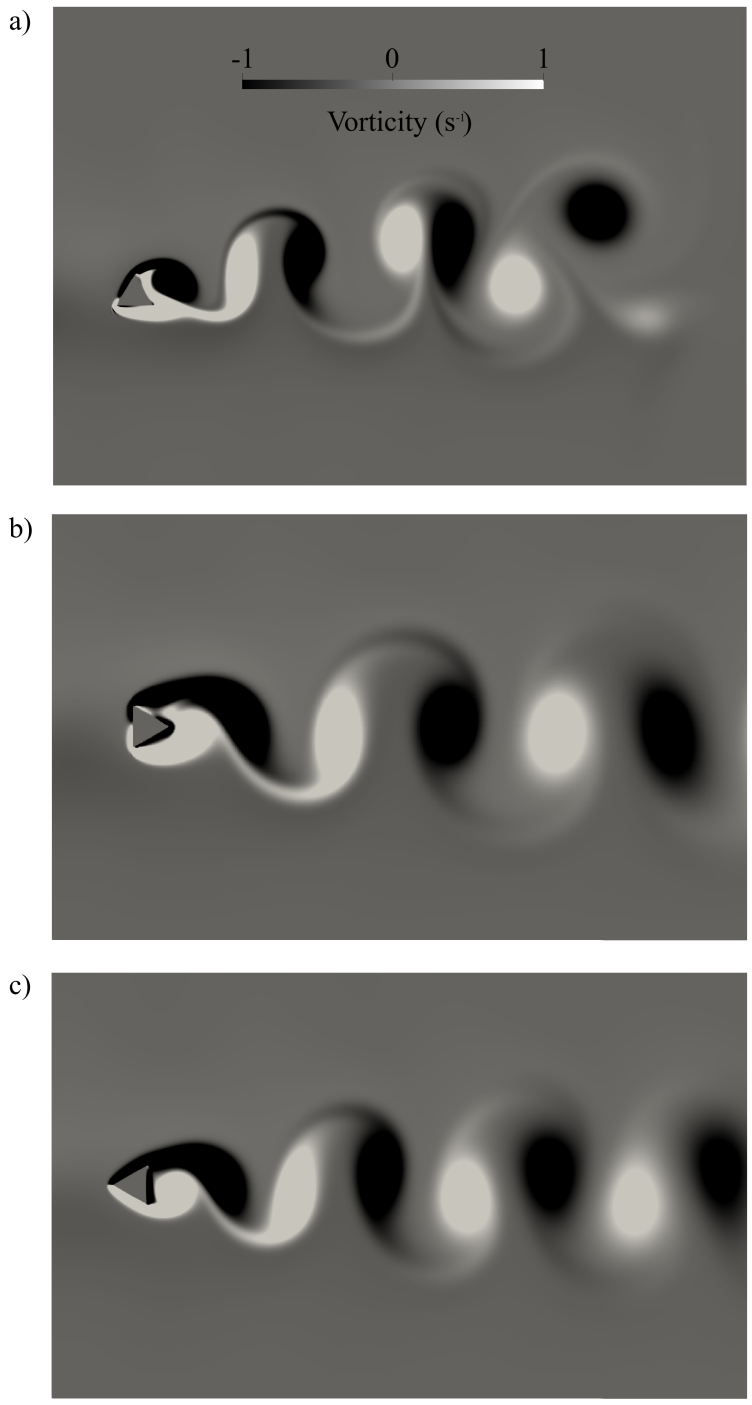

Figure 21: Instantaneous vorticity field for the flow around a triangular cylinder rotated by (a) 30° (Re = 200), (b) 60° (Re = 100), (c) 0° (Re = 100).

it consists of many narrow forks and the special treatments presented in the paper are therefore necessary. We solve the steady-state incompressible Navier-Stokes equations inside the arterial system with the blood

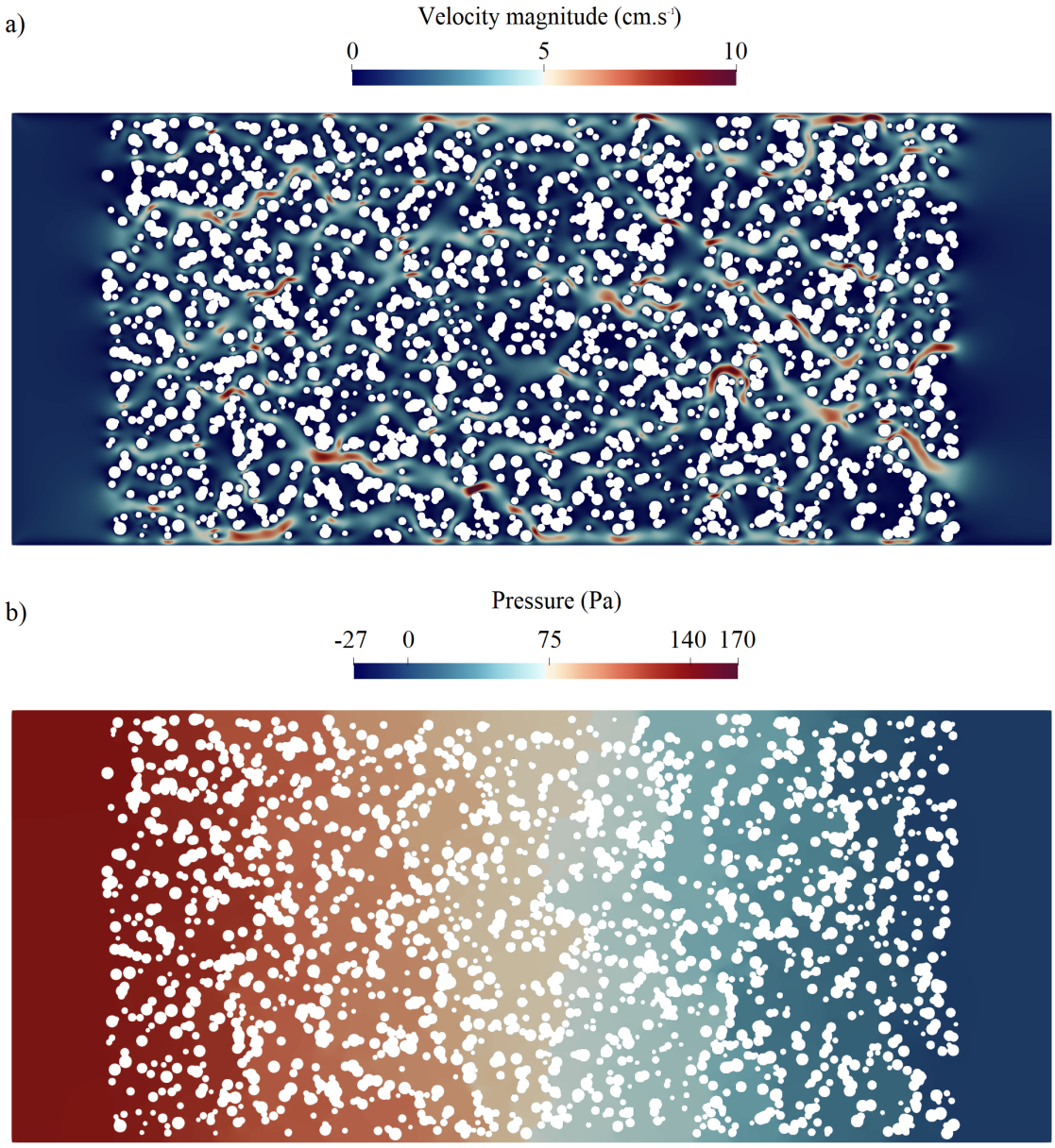

Figure 22: a) Simulated flow velocity through a porous medium. b) Simulated pressure in the porous medium.

kinematic viscosity ($\nu = 10^{-6}$ m^2.s^{-1}). The Reynolds number is approximately equal to 100 in the largest artery with an imposed velocity of 10 cm.s^{-1} at the entrances.

The mesh of the initial numerical domain consists of 526 million cells. After applying the Cartesian decomposition domain algorithm, only 14.4 million cells remain. The simulation is performed with 686 cores on the same supercomputer as in section 2.3. Figure 23 shows both the surface shape reconstruction of the arterial system and the simulated velocity field. The flow inside the medium-sized arteries can reach 60 cm.s^{-1}.

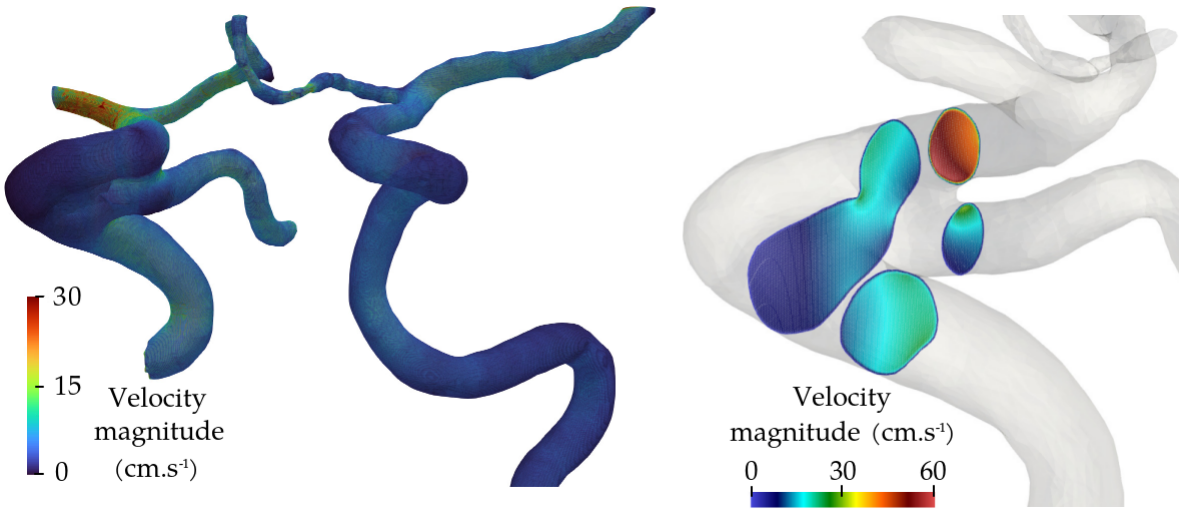

Figure 23: Simulated flow velocity within an irregular arterial system.

5. Conclusion

IBMs are well suited to running large simulations with fine Cartesian meshes. However, the intrinsic nature of this method makes it difficult to apply to complex geometries and very large meshes. To make it more attractive, we have developed two algorithms operating on two critical pre-processing steps. First, when simulating flow around complex geometries, a significant proportion of cells are outside the physical domain. Although unnecessary, they must be included in the computations to allow massively parallel libraries to be used. In such cases, IBMs lose their advantages over the use of an unstructured mesh. To overcome the limitations of this method, we have improved the Anupindi method, which removes the partitions containing only fictitious cells from the computation during the domain decomposition process. Based on the proposed sliding process of partition lines, additional partitions are removed. In the presented examples, the method has eliminated between 20 and 30 % extra partitions compared to the Anupindi approach. In addition, the algorithm maintains cuboidal partitions, which enables the use of massively parallel libraries such as hypre. This novel approach thus reduces the number of processors required to run IBM-based simulations. For meshes consisting of billions of cells, the computation time remains small (less than 20 minutes) compared to other domain decomposition algorithms. We have shown that the algorithm is efficiently parallelizable up to 128 processors.

Second, we have addressed the problem of degenerate cells occurring when irregular geometries are considered. The direct ghost-cell method allows a second order application of the boundary conditions, but is not well suited when applied to a degenerate cell. Irregular geometries often produce degenerate cells if the mesh is not sufficiently refined. Refinement can be prohibitively expensive for Cartesian meshes, and we have shown that it does not necessarily avoid degenerate cells. We have therefore developed an algorithm for reconstructing geometry using line segments in 2D and triangles in 3D, which locally modifies the initial geometry to remove the degenerate cells. The first part of the algorithm, where the contour is reconstructed, only needs ray tracing to be performed. The main method of removing the degenerate cells is based on the creation of a super-cell composed of several cells. This process locally coarsens the geometry with a characteristic size equivalent to the super-cell size. The subgrid parts of the contour are thus removed and

replaced by smoother geometries. The proposed algorithm has been successfully applied to several 2D and 3D irregular geometries. The reconstructed contours or surfaces converge to the input geometries as the mesh is refined. The computation time is very small in 2D, a few seconds for meshes of 100 million cells, and the algorithm shows a good scalability. In 3D, the process takes about 1 minute for meshes of 1.2 billion cells using 2,048 processors, which is negligible compared to the total simulation time. The scalability of the 3D algorithm is also satisfactory.

We have presented the application of the developed methods to such a 2D porous medium and a 3D complex arterial system. Both simulations were managed by Notus CFD, where the approaches were implemented. Both algorithms will benefit IBM users wishing to perform simulations in irregular geometries.

6. Acknowledgment

We thank the Ministry of Culture and the DRAC Nouvelle-Aquitaine for providing funding for the project. We thank the University of Bordeaux and University of Pau and the Adour Region for providing access to MCIA (Mésocentre de Calcul Intensif Aquitain). This work has also benefited from access to TGCC HPC/IA resources through the GENCI grant 2022-A0112A12927.

References

[1] C. S. Peskin, Flow patterns around heart valves: a numerical method, Journal of Computational Physics 10 (2) (1972) 252–271.
[2] R. Mittal, G. Iaccarino, Immersed boundary methods, Annual Review of Fluid Mechanics 37 (2005) 239–261.
[3] B. E. Griffith, N. A. Patankar, Immersed methods for fluid-structure interaction, Annual Review of Fluid Mechanics 52 (2020) 421–448.
[4] P. G. Tucker, Z. Pan, A cartesian cut cell method for incompressible viscous flow, Appled Mathematical Modelling 24 (8-9) (2000) 591–606.
[5] D. M. Ingram, D. M. Causson, C. G. Mingham, Development in cartesian cut cell methods, Mathematics and Computers in Simulation 61 (3) (2003) 561–572.
[6] Y. Cheny, O. Botella, The LS-STAG method: A new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties, Journal of Computational Physics 229 (4) (2010) 1043–1076.
[7] Y. Cheny, O. Botella, The LS-STAG immersed boundary method for non-newtonian flows in irregular geometries: flow of shear-thinning liquids between eccentric rotating cylinders, Theoretical and Computational Fluid Dynamics 29 (1-2) (2015) 93–110.
[8] A. Coco, 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.
[9] R. Mittal, H. Dong, M. Bozkurttas, F. M. Najjar, A. Vargas, A. von Loebbecke, A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries, Journal of Computational Physics 227 (10) (2008) 4825–4852.
[10] K. Zheng, X. Zhao, D. Yan, Numerical simulation of water entry of two-dimensional structures with complex geometry using a CIP-based model, Applied Ocean Research 106 (2021) 102379.
[11] K. Onishi, M. Tsubokura, Topology-free immersed boundary method for incompressible turbulence flows: An aerodynamic simulation for "dirty" CAD geometry, Computer Methods in Applied Mechanics and Engineering 378 (2021) 113734.
[12] W. Noordt, S. Ganju, C. Brehm, An immersed boundary method for wall-modeled large-eddy simulation of turbulent high-mach-number flows, Journal of Computational Physics 470 (2022) 111583.
[13] M. Lauber, G. D. Weymouth, G. Limbert, Immersed boundary simulations of flows driven by moving thin membranes, Journal of Computational Physics 457 (2022) 111076.
[14] J. Picot, S. Glockner, Discretization stencil reduction of direct forcing immersed boundary methods on rectangular cells: the ghost node shifting method, Journal of Computational Physics 364 (2018) 18–48.
[15] A. M. D. Jost, S. Glockner, Direct forcing immersed boundary methods: Improvements to the ghost node method, Journal of Computational Physics 438 (2021) 110371.

[16] R. D. Falgout, U. M. Yang, hypre: A library of high performance preconditioners, International Conference on Computational Science (2002) 632–641.
[17] K. Anupindi, Y. Delorme, D. A. Shetty, S. H. Frankel, A novel multiblock immersed boundary method for large eddy simulation of complex arterial hemodynamics, Journal of Computational Physics 254 (2013) 200–218.
[18] C. Zhu, J.-H. Seo, R. Mittal, A graph-partitioned sharp-interface immersed boundary solver for efficient solution of internal flows, Journal of Computational Physics 386 (2019) 37–46.
[19] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.
[20] G. Karypis, K. Schloegel, V. Kumar, Parmetis. Parallel graph partitioning and sparse matrix ordering library, Technical report (2003).
[21] K. Khadra, P. Angot, S. Parneix, J.-P. Caltagirone, Fictitious domain approach for numerical modelling of Navier-Stokes equations, International Journal for Numerical Methods in Fluids 34 (8) (2000) 651–684.
[22] E. Arquis, J.-P. Caltagirone, Sur les conditions hydrodynamiques au voisinage d'une interface milieu fluide milieu poreux: application à la convection naturelle, Comptes rendus de l'Académie des Sciences 299 (1984) 1–4.
[23] M. Berger, Chapter 1 - cut cells: Meshes and solvers, in: R. Abgrall, C.-W. Shu (Eds.), Handbook of Numerical Methods for Hyperbolic Problems, Vol. 18 of Handbook of Numerical Analysis, 2017, pp. 1–22.
[24] M. J. Aftosmis, M. Berger, J. E. Melton, Robust and efficient Cartesian mesh generation for component-based geometry, AIAA Journal 36 (1998) 952–960.
[25] P. Colella, EBChombo Software Package for Cartesian Grid, Embedded Boundary Applications, Tech. rep., Lawrence Berkley National Laboratory (2014).
[26] M. Tao, C. Batty, E. Fiume, D. I. W. Levin, Mandoline: robust cut-cell generation for arbitrary triangle meshes, ACM Transactions on Graphics 38 (6) (2019) 1–17.
[27] L. Freitag, On combining laplacian and optimization-based mesh smoothing techniques, Proceedings of the 1997 Joint Summer Meeting of American Society of Mechanical Engineers (ASME), American Society of Civil Engineers (ASCE) and Society of Engineers Science (SES) (1997) 37–44.
[28] N. Amenta, M. Bern, D. Eppstein, Optimal point placement for mesh smoothing, 8th ACM-SIAM Symp. on Discrete Algorithms (1997).
[29] S. Canann, M. Stephenson, T. Blacker, Optismoothing: An optimization-driven approach to mesh smoothing, Finite Elements in analysis and Design 13 (1993) 185–190.
[30] S. Popinet, Gerris: a tree-based adaptive solver for the incompressible euler equations in complex geometries, Journal of Computational Physics 190 (2003) 572.
[31] H. Johansen, P. Colella, A Cartesian grid embedded boundary method for Poisson's equation on irregular domains, Journal of Computational Physics 147 (60) (1998).
[32] H. Udaykumar, W. Shyy, M. Rao, ELAFINT: a mixed Eulerian–Lagrangian method for fluid flows with complex and moving boundaries, International Journal for Numerical Methods in Fluids 22 (691) (1996).
[33] M. J. Aftosmis, M. J. Berger, G. Adomavicius, A parallel cartesian approach for external aerodynamics of vehicles with complex geometry, in: Proceedings of the Thermal and Fluids Analysis Workshop, AL, 1999.
[34] M. S. Day, P. Colella, M. J. Lijewski, C. A. Rendleman, Embedded boundary algorithm for solving the poisson equation on complex domains, Tech. rep., Lawrence Berkley National Laboratory (1998).
[35] H. Ji, F.-S. Lien, E. Yee, A robust and efficient hybrid cut-cell/ghost-cell method with adaptive mesh refinement for moving boundaries on irregular domains, Computer Methods in applied Mechanics and Engineering 198 (3-4) (2008) 432–448.
[36] Z. Wang, J. Seo, R. Mittal, Mitral valve regurgitation murmurs—insights from hemoacoustic computational modeling, Fluids 7 (5) (2022) 164.
[37] M. Crowley, H. Sitaraman, J. Klinger, F. Usseglio-Viretta, N. Thornburg, N. Brunhart-Lupo, M. Pecha, J. Dooley, Y. Xia, P. Ciesielski, Measurement of transport properties of woody biomass feedstock particles before and after pyrolysis by numerical analysis of x-ray tomographic reconstructions, Frontiers in Energy Research 10 (2022).
[38] Notus CFD, https://notus-cfd.org.
[39] L. Timmermans, P. Minev, F. V. D. Vosse, An approximate projection scheme for incompressible flow using spectral elements, International journal for numerical methods in fluids 22 (7) (1996) 673–688.
[40] Z. Y. Ng, T. Vo, W. K. Hussam, G. J. Sheard, Two-dimensional wake dynamics behind cylinders with triangular cross-section under incidence angle variation, Journal of Fluids and Structures 63 (2016) 302–324. doi:https://doi.org/10.1016/j.jfluidstructs.2016.04.003.
[41] M. Moradzadeh, S. Boroomandnasab, H. Moazed, J. Alavi, A. Jamalian, M. Khaledian, S. Ruy, A new kinematic–dispersive wave van genuchten (kdw-vg) model for numerical simulation of preferential water flow in soil, Journal of Hydrology 582 (2020) 124480.
[42] H. Zhong, Y. He, E. Yang, Y. Bi, T. Yang, Modeling of microflow during viscoelastic polymer flooding in heterogenous

reservoirs of daqing oilfield, Journal of Petroleum Science and Engineering 210 (2022) 110091.
[43] B. K. Swain, A. K. Nayak, Analytic heat and solute transport of mhd reactive mixed convection flow over a horizontal porous stretching sheet with multiple slips, Journal of Porous Media 26 (2) (2023) 69–87.
[44] M. Bai, Z. Liu, L. Zhan, M. Yuan, H. Yu, Effect of pore size distribution and colloidal fines of porous media on the transport behavior of micro-nano-bubbles, Colloids and Surfaces A: Physicochemical and Engineering Aspects 660 (2023) 130851.
[45] M. Hamdan, H. M. Alargha, E. Elnajjar, A. Hilal-Alnaqbi, A. Elshawarby, W. H. Aziz, Numerical hemodynamics analysis for cerebral aneurysm with porous inserts, Journal of Porous Media 21 (13) (2018) 1439–1448.
[46] K. Chaudhari, H. Patel, Hemodynamics numerical simulation of Stenosis bifurcation, ASME International Mechanical Engineering Congress and Exposition, Proceedings (IMECE) 3 (2015).