Received: 5 May 2021 | Revised: 30 August 2021 | Accepted: 22 September 2021
DOI: 10.1002/fld.5050

RESEARCH ARTICLE

On the Accuracy, Robustness and Performance of High Order Interpolation Schemes for the Overset Method on Unstructured Grids

Sébastien Lemaire*[1,2] | Guilherme Vaz[3,1] | Menno Deij - van Rijswijk[2] | Stephen R Turnock[1]

[1]Dept of Civil, Maritime and Environmental Eng, University of Southampton, Southampton, United Kingdom
[2]R&D Dept, MARIN, Wageningen, The Netherlands
[3]HPC & Data Science Team, WavEC Offshore Renewable, Lisbon, Portugal

Correspondence
*Sébastien Lemaire. Email: sebastien.lemaire@soton.ac.uk

Abstract
A comprehensive study on interpolation schemes used in overset grid techniques is here presented. Based on a literature review, numerous schemes are implemented, and their robustness, accuracy and performance are assessed. Two code verification exercises are performed for this purpose: a 2D analytical solution of a laminar Poiseuille steady flow; and an intricate manufactured solution of a turbulent flow case, characteristic of a boundary layer flow combined with an unsteady separation bubble. For both cases the influence of grid layouts, grid refinement and time-step is investigated. Local and global errors, convergence orders, and mass imbalance are quantified. In terms of computational performance, strong scalability, cpu timings, load imbalancing and DCI overhead are reported. The effect of the overset-grid interpolation schemes on the numerical performance of the solver, i.e. number of non-linear iterations, is also scrutinised. The results show that, for a 2nd order finite volume code, once diffusion is dominant (low Reynolds number), interpolation schemes higher than 2nd order, e.g. *Least squares* of degree 2, are needed not to increase the total discretisation errors. For convection dominated flows (high Reynolds numbers), the results suggest that 2nd order schemes, e.g. *Nearest cell gradient*, are sufficient to prevent overset grid schemes to taint the underlying discretisation errors. In terms of performance, by single-process and parallel communication optimisation, the total overset-grid overhead (with DCI done externally to the CFD code) may be less than 4% of the total run time for 2nd order schemes and 8% for 3rd order ones, therefore empowering higher-order schemes and more accurate solutions.

KEYWORDS:
Overset Grids; Interpolation; Code Verification; 3D URANS Manufactured Solution; Computational Performance

1 | INTRODUCTION

The overset grid method, also known as the chimera or overlapping grid method, is used for a wide range of CFD applications in aerospace, for which it was first designed[1], and maritime[2] fields. By enabling *a priori* unknown motions of an arbitrary number of bodies, the overset grid method allows otherwise impossible computations to be performed, for example a 6 degrees of freedom (DOF) motion of a fully appended ship manoeuvring alongside other vessels or close to a harbour[3]. The overset grid method was first used in finite-difference structured CFD codes back in 1980s[1], the 1990s saw the first implementations for structured-grid finite volume codes for maritime applications[4]. Currently, unstructured-grid finite volume codes[5,6,7,8,9] and higher-order finite element codes[10] often have an overset grid capabilities. In the current work, the overset grid technique is studied for face-based unstructured-grid finite volume viscous-flow codes, particularly for incompressible hydrodynamic flows at high Reynolds numbers.

The overset grid method works by overlapping or oversetting several grids in a single simulation. Each body, or its parts, may have one or more body-fitted meshes that will move with it, and a background grid is used for the remainder of the domain. This also enables the use of simpler, generally higher quality and/or even Cartesian individual meshes, instead of complex unstructured grids. Information transfer between the different independent mesh boundaries is done using interpolation at the so-called *fringe* cells[11]. This interpolation phase has to be fast and accurate. Fast, both in terms of computational and parallel communication performance, because it is performed for multiple cells at several iteration levels, and the geometric basis for interpolation has to be recomputed every time one of the grids changes due to movement, deformation or grid refinement (in CFD codes that possess these features). Accurate, so as not to distort the solution with numerical errors due to interpolation.

Theoretically, it is known that the order of convergence of any interpolation method has to be of the same or one-order higher than related discretisation schemes, if their errors are to be of a comparatively negligible level[12]. Several questions arise here: is this also true for interpolation between overset grids, for all sorts of grid layouts, cell types, orientations, convection or diffusion dominated flows? What are the existing overset grid-enabled CFD codes using for this interpolation phase? And for 2nd order accurate finite volume solvers, do the interpolations have to be of 3rd or even higher orders in complex situations? Due to interpolation of field values at cell centres, rather than interpolation of locally conserved face fluxes, application of the overset grid method may suffer from mass imbalance[8]. Can this mass imbalance be influenced, improved or even deteriorated by the type of overset grid interpolation used?

It is often stated that the usage of overset grid method imposes a large computational performance overhead[9,13,14]. Understandably so, since this potential performance penalties may have many sources: 1) the extra hole cutting and domain connectivity information (DCI) needs to be computed, either by the CFD code itself[8], or by external libraries[15,16]; 2) the calculation of the interpolation itself; 3) additional parallel communication between subdomains; 4) non-ideal parallel load balancing due to non-overset grid partitioning[17], i.e. the number of communication operations, or cells needing communication, is not the same for all parallel subdomains; 5) the structure of the discretised equations is modified when using overset grids, which may influence the iterative performance of the preconditioners and solvers used for the solution of the linear system of equations; 6) when using explicit interpolation techniques, the non-linear iterative performance can deteriorate and more non-linear loops may be needed for reaching a desired level of iterative convergence per time-step. Therefore, a fast, low overhead overset grid method is not a straightforward accomplishment.

The open literature on overset grids has some studies on the topics listed above. But these employ either simple analytical solutions[18,19] which are far from the desired practical applications, or use industrial cases where the emphasis is validation[18,19,8,20]. Due to the inherent complexity, numerical and modelling errors are then entangled and therefore difficult to quantify. Moreover, in these papers, a small number of interpolation techniques are studied and rarely are accuracy, convergence order, robustness, and computational performance simultaneously and comprehensively analysed.

In the current work, after a detailed analysis of several overset grid-capable finite volume codes, a summary of all interpolation schemes used in these tools and related DCIs is made and their theoretical properties are analysed. To completely characterise the overset grid method, verification studies are used that capture the likely mesh overlap structures that would be used in more complex flow features such as turbulence, transition, free-surface, and cavitation.

It should be noted that an overset grid method does not improve, nor should it degrade the modelling errors of a specific solution but only its discretisation, therefore an overset grid method itself cannot be validated but only verified. Nevertheless, for the solution to not be degraded by the overset method, associated numerical errors of an overset grid should be quantified, and remain lower than the numerical errors of the transport equation discretisation.

To accomplish this objective, modern verification techniques[21,22] are here used for two test cases: 1) a simple, analytical 2D steady Poiseuille flow at low-Reynolds number[23,11]; and 2) a more complex 3D unsteady manufactured solution at high-Reynolds number[24]. The latter resembles a flat-plate flow together with a pulsating recirculation bubble and involves a manufactured solution for the unsteady RANS equations, including a 1-eq turbulence model. At this stage, only static grid assemblies are considered, that is, the grids do not move. However, several meshes and cell types, background and foreground relative orientations are considered, always having in mind grid- and time-step refinement, as it is crucial for a proper verification study. In total, more than 14 overset grid interpolation schemes are analysed in terms of accuracy, robustness and performance. The resulting mass imbalance is also quantified. For computational performance, absolute timings and strong scalability are analysed. Finally, as it is possible for the cases studied, overset-grid solutions will be compared to their non-overset counterparts.

The studies presented in this work have been performed using the community-based multi-phase viscous-flow finite volume code ReFRESCO[25,26,11] and the overset domain connectivity information (DCI) is provided by the external library Suggar++[27,15] via an internal coupling. This combination is characteristic of several CFD overset grid-enabled codes available nowadays, and consequently the results presented here are expected to be relevant for all these state-of-the-art tools.

This paper is structured as follows: after this introduction, the implementation and the different interpolation schemes used in this study are presented in section 2; the numerical setup is explained in section 3. Section 4 presents the two test cases and focuses on the accuracy and robustness of the different interpolation schemes, while section 5 measures and discusses all the elements impacting the performance of overset grid computations. Section 6 highlights the major conclusions of the work and emphasises some aspects to be further investigated.

2 | INTERPOLATION FOR THE OVERSET METHOD

In an overset grid computation, multiple meshes are overlapped in the same domain. *Fringe* cells are placed at each grid boundary and they receive interpolated field information from *donor* cells on another mesh. Finally, cells that are no longer part of the simulation due to the overlap, called *hole* cells, are disabled by the solver. Figure 1 summarises the different cell statuses on a Cartesian mesh. Their definition is part of the Domain Connectivity Information (DCI). The coupling is performed by modifying the system of equations. When an explicit interpolation approach is used, the interpolated values corresponding to each *fringe* cell are placed on the system's right hand side, and a unitary diagonal element is used. On the other hand, with an implicit formulation, interpolation weights associated with each *donor* cell are placed on the left hand side. Figure 2 describes the solution workflow with an explicit coupling highlighting the steps specific of an overset computation.

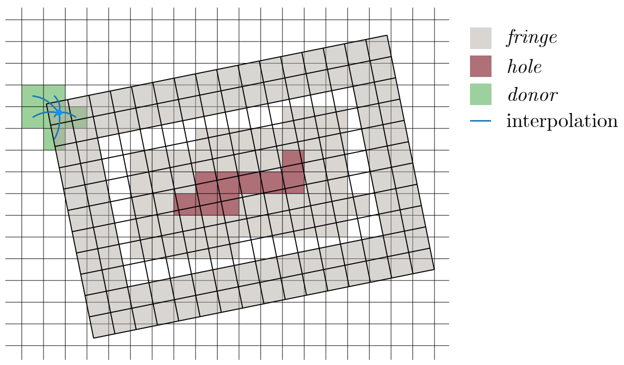


FIGURE 1 Example of overset assembly with its nomenclature. *Donor* cells for the top left *fringe* cell are displayed.

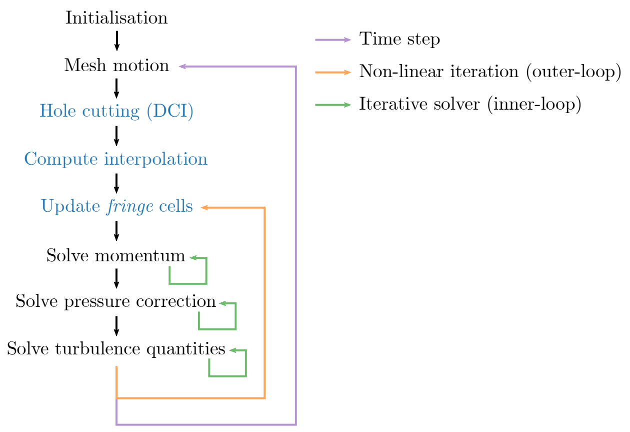

Initialisation
Mesh motion
Hole cutting (DCI)
Compute interpolation
Update fringe cells
Solve momentum
Solve pressure correction
Solve turbulence quantities
Time step
Non-linear iteration (outer-loop)
Iterative solver (inner-loop)

FIGURE 2 Solution workflow with explicit overset coupling. Steps marked in blue are specific to an overset computation.

2.1 | Scheme metrics

A wide variety of interpolation schemes can be and has been used for overset coupling. Comparing them, however, is not always straightforward. This work proposes three metrics to facilitate such a comparison: accuracy, robustness and performance.

2.1.1 | Accuracy

Several factors affect the accuracy of an interpolation scheme. Most interpolation schemes' inherent error can be modelled by $e = \alpha h^p + o(h^{p+1})$ with α and p constants and h a metric of the mesh refinement. The interpolation order, p, defines the rate at which the error decreases with mesh refinement. Since α also plays a role, the order alone does not guaranty an error level. Contrary to the order p, α can not, in general, be computed without tests because it depends on the *donor* cells location or the mesh topology. Therefore tests with multiple refinement levels are needed to assess the error levels fully. Moreover, some interpolation schemes cannot be easily modelled by the equation presented above, requiring tests to quantify their interpolation order which may be mesh-dependent.

Finally, in this study as well as in most overset implementation, interpolation is performed on cell centres or vertex values, which does not guarantee the flux conservation property of the finite volume method. This is particularly important for mass fluxes, for which lack of conservation leads to pressure fluctuations in the entire domain. Methods reconstructing fluxes passing through an overset interface exist and ensure mass as well as momentum and energy conservation. They are, however, not ubiquitous and won't be tested in this study.

2.1.2 | Robustness

The robustness of an interpolation scheme is defined by the probability of giving low error results. Some schemes can, under particular unfavorable circumstances, return very high errors. Depending on the scheme, *donor* cell locations or the number of *donor* cells used can be the cause of these high errors. Such errors are often caused by a poor a conditioning of the system of equations used to solve to perform the interpolation.

On the other hand, some schemes always return bounded quantities by design, meaning that the interpolated value will always be within its *donor* cells values range, thereby guaranteeing robustness. Schemes that do not provide such a guarantee can be made more robust by limiting the returned interpolated value to lie inside the *donor* cell values' range. This *ad hoc* method, however, lowers the interpolation order and can potentially decrease the accuracy. Moreover, if bounding is performed, the interpolated value can not be expressed as a linear combination of the *donor* cells' values anymore, which blocks the interpolation scheme from use in an implicit coupling[28].

2.1.3 | Performance

A computation using an overset method has an additional performance overhead coming from the extra steps that are part of the method. First of all, the computation of the DCI is often complex to accelerate via MPI (Message Passing Interface) distributed-memory parallelisation, and needs to be redone every time a mesh is changing. Secondly, the *donor* search is performed every time a mesh is changing. Finally, during the interpolation phase either interpolation weights are stored and reused within each outer-loop, or the interpolation itself has to be computed every outer-loop.

Each of these steps requires additional computational time, both in terms of CPU operations and communication between different MPI processes. While the performance of each of these overset phases and the overset method itself can vary substantially depending on its implementation, type of discretisation, structure vs unstructured grid code, etc., in this work, we intend to show relative trends and results.

Moreover, in addition to the turnaround time of the interpolation and DCI computation, the coupling method influences the iterative convergence and therefore the performance. Even though an explicit coupling requires no extra parallel communication, it can impact iterative convergence compared to an implicit coupling, leading to a higher number of outer-loops required to obtain a similar level of iterative error. Finally, the interpolation accuracy itself can influence the iterative process.

2.2 | Interpolation Methods

A detailed description of the interpolation methods used in this study is included in Appendix A and Table 1 summarises their main characteristics. *Nearest cell* and *Nearest cell gradient* schemes only require a single *donor* cell and perform first or second-order interpolation, respectively. The *Inverse distance* scheme can use any number of *donor* cells and uses a weighted average based on the distance to the interpolation location, leading to a first-order scheme.

Next, the *Polynomial*, *Polynomial tensor* and *Least squares* schemes are all based on polynomial interpolation. The first two require a fixed number of *donor* cells and work by finding a polynomial function that passes through all of the *donor* cells values. The *Least squares* approach also generates a polynomial function, but over-determining it in a least squares sense (*i.e.* using more *donor* cells than is strictly required) makes the scheme more robust. One of the main characteristics of all these polynomial-based schemes is that they can be designed for any interpolation order, as a polynomial function of any degree can be generated. Higher order *Polynomial* schemes do require more *donor* cells, and tends to be less robust though. One should note that the interpolation order of a polynomial based scheme is one higher than its degree.

Finally, in the *Barycentric* approach coefficients are assigned to the *donor* points, such that this will lead to the interpolation location being the system's barycentre. These coefficients are then used as the *donor* cells' interpolation weights.

TABLE 1 Summary of different interpolation schemes available in ReFRESCO and their main characteristics. Since the number of *donor* cells for the *Barycentric* interpolation depends on the mesh topology it is here given as an example for a Cartesian mesh.

Scheme	Number of *donor* cells		Order (theoretical)	Boundedness
	2D	3D		
Nearest cell	1	1	1	yes
Nearest cell gradient	1	1	2	no
Inverse distance	N	N	1	yes
Polynomial $(degree = 1)$	3	4	2	
Polynomial $(degree = 2)$	6	10	3	no
Polynomial $(degree = 3)$	10	20	4	
Polynomial $(degree = n)$	$\frac{(n+1)(n+2)}{2}$	$\sum_{i=0}^{n} \frac{(i+1)(i+2)}{2}$	$n+1$	
Polynomial tensor $(degree = 1)$	4	8	2	
Polynomial tensor $(degree = 2)$	9	27	3	no
Polynomial tensor $(degree = 3)$	16	64	4	
Polynomial tensor $(degree = n)$	$(n+1)^2$	$(n+1)^3$	$n+1$	
Least squares $(degree = 1)$	> 3	> 4	2	
Least squares $(degree = 2)$	> 6	> 10	3	no
Least squares $(degree = 3)$	> 10	> 20	4	
Least squares $(degree = n)$	$> \frac{(n+1)(n+2)}{2}$	$> \sum_{i=0}^{n} \frac{(i+1)(i+2)}{2}$	$n+1$	
Barycentric (type 1)	6	18		yes
Barycentric (type 2)	6	18		no
Barycentric (type 3)	9	27		yes
Barycentric (type 4)	9	27		no

2.3 | Interpolation methods used in other overset solvers

Interpolation is a core component of an overset implementation, but publications using overset meshes do not always describe in detail the method they use. The *donor* cell collection method is not often mentioned, and the actual interpolation scheme is either not mentioned, or not described in enough detail to be understood or implemented by others[3,29,2,30,31,8]. Table 2 summarises the data found in the literature, and when possible, a nomenclature similar to the one used in the presented work is used. Unstructured and structured solvers are distinguished as the *donor* collection can be done very easily with a structured code, and the interpolation schemes can take advantage of this. This is the case for UP_GRID solver[32], which generates a spline in the entire domain for the interpolation. Suggar++, when used with structured codes, can use a *Polynomial tensor* interpolation which needs n^3 *donor* cells, with the search for these cells being trivial on a structured grid.
Interpolation orders higher than 2 are only achieved with *OPErA* library using a least squares approach. Most of the other solvers implement a *Nearest cell* and an *Inverse distance* scheme (both 1st order). Even though the *donor* cells collection method is not always described, at least 3 of the 6 solvers are using the closest cell's direct neighbours. This means that the number of *donor* cells collected for each *fringe* cell depends on the grid topology and can not be changed. Finally, several schemes make use of the gradient at the *donor* cell centre locations to achieve 2nd order interpolations.

Detailed studies of interpolation methods for overset are not often published. Chandar, on the contrary, did several more thorough tests of the interpolation schemes implemented in OPErA[33,5,34,18]. In 2012, Quon et al. also tested the use of Radial Basis Functions (RBF) for overset interpolation[35,36], and even though positive conclusions were drawn, it seems that no other overset library or CFD solver is using this method, possibly because most overset implementations rely on having only the direct neighbours of the *Nearest cell* as *donor* cells. The RBF method, however, requires more *donor* cells, making its implementation and parallelisation acceleration more challenging for most codes.

TABLE 2 Summary of interpolation methods used in various overset implementations.

Solver	Type	Scheme	Number of *donor* cells	Order
Fresco+[8]	Unstructured	*Nearest cell*	1	1
		Inverse distance	Neigh[a]	1
		Inverse Simplex Volume[b]	4	1 or 2
StarCCM+[7]	Unstructured	*Inverse distance*	4	1
		Shape Function		1
		Gradien Least Squares[f]	Neigh[a]	2
ANSYS-Fluent[5]	Unstructured	Linear[d]	4	1
		Inverse distance		1
		Gradient Least Squares[f]	Neigh[a]	2
FOAM-extend[9,37]	Unstructured	*Nearest cell*	1	1
		Average		1
		Inverse distance		1
ISIS CFD/FINE Marine[6]	Unstructured	*Least squares*	Neigh[a]	2
OPErA (openFOAM)[33,5,34,18]	Unstructured	Linear[d]	4	1
		Inverse distance		1
		Least squares	-[e]	2-4
Suggar++[c] [38,27,15,28]	Both	*Nearest cell*	1	1
		Inverse distance	Neigh[a]	1
		Laplacian	Neigh[a]	2
		Least squares	Neigh[a]	2
		Dual grid	-	2
	Structured	*Polynomial tensor*	n^3	n
LAVA[39]	Unstructured	*Least squares*	Neigh[a]	
	Structured	*Polynomial tensor*	8	2
UP_GRID (NMRI)[40,32]	Structured	Ferguson spline		
Pusan University[41]	Structured	*Barycentric*		

[a] The *donor* cells are the closest cell and its direct neighbours
[b] Interpolation weights based on the inverse simplex volumes, it is made second order by using *donor* cells gradients
[c] Suggar++ is a library being used by several solvers like CFD-Shipflow[13], OVERFLOW[42] and some implementations of OpenFOAM[30,43]
[d] Using vertices to construct a tetrahedron and perform the interpolation
[e] OPErA is not using a fixed number of *donor* cells and therefore the interpolation order is not guaranteed, a pseudo-inverse method is used if not enough *donor* cells are collected and the system is under-determined
[f] The least squares approach is not coupled with a polynomial function here. The gradient at the *donor* cell centre is used to extrapolate a field value at the interpolation location from each *donor* (like the *Nearest cell gradient* scheme), then the least squares approach is used as an averaging method between the values

3 | NUMERICAL SETUP

3.1 | CFD code

3.1.1 | ReFRESCO solver

ReFRESCO (REliable & Fast Rans Equations (solver for) Ships (and) Constructions Offshore)[25] is a research and commercial CFD code that is optimised for maritime applications. It is being developed, verified and validated at MARIN (Maritime Research Institute Netherlands) in collaboration with several other organisations like IST (Instituto Superior Técnico in Lisbon, Portugal), the University of Southampton (Southampton, the UK), TU Delft (Technical University of Delft, the Netherlands) or WavEC Offshore Renewables (Lisbon, Portugal).

ReFRESCO is a viscous-flow CFD code that solves multiphase (unsteady) incompressible flows using the Navier-Stokes equations, complemented with turbulence models, cavitation models and volume-fraction transport equations for different phases. The equations are discretised using a finite volume approach with cell-centered collocated variables, in strong-conservation form, and a pressure-correction equation based on the SIMPLE algorithm is used to ensure mass conservation. Time integration is performed implicitly with first or second-order backward schemes. At each implicit time-step, the non-linear system for velocity and pressure is linearised with Picard's method and either a segregated or coupled approach is used. A segregated approach is always adopted for the solution of all other transport equations. The implementation is face-based, which permits grids with elements consisting of an arbitrary number of faces (polyhedral, hexahedral, tetrahedral, prisms, pyramids, etc.), and if needed h-refinement (hanging nodes).

Moving, sliding and deforming grids, as well automatic grid adaptation are also available. Coupling with structural equations-of-motion (rigid-body 6DOF), and flexible-body fluid structure interaction is possible. For turbulence modelling, both RANS/URANS and Scale-Resolving Simulations (SRS) models such as SAS, DDES/IDDES, XLES, PANS and LES approaches can be used. Code parallelisation is done using MPI and subdomain decomposition computed by parMETIS[17].

3.1.2 | Overset implementation

The implementation of the overset method conducted in ReFRESCO is using the two external libraries Suggar++[15] and DiRTlib[38]. Suggar++ computes the DCI (Domain Connectivity Information), which gives the status for each cell (either *in*, *hole* or *fringe*), and DiRTlib facilitates the use of the output from Suggar++. Suggar++ also has the capability to perform the interpolation itself. In this study, however, the interpolation is done by ReFRESCO. This means that only the so-called IBLANK array containing cell status information is provided by Suggar++, and ReFRESCO does the *donor* search for each *fringe* cell, as well as the interpolation using one of the methods presented in appendix A.

The coupling with the flow solver is done explicitly for all equations[11,23]. Interpolated values are set at the right-hand side of the system, and a unitary diagonal is applied on the lines corresponding to *fringe* cells. The pressure field is also being interpolated on *fringe* cells, employing a zero pressure correction on these cells. This explicit coupling means that *fringe* cells' field data is being updated only every outer-loop and not within inner-loops steps. For schemes that allow it, interpolation weights are computed and stored in memory to be re-used when grids are not moving to decrease computational cost.

3.1.3 | Interpolation implementation

The *donor* cell search was designed and implemented to minimise the number of parallel synchronisation steps and the amount of data being communicated between MPI processes. The algorithm is given in pseudo-code in listing 1. The general idea to select *donor* cells is to first find the cell that contains the interpolation location (*fringe* cell centre) and then gather its neighbours, followed by the neighbours of the neighbours etc., until enough *donor* cells have been gathered for the demanded scheme. The order in which neighbours of the same *layer* are gathered is based on their distance to the *fringe* cell centre location.
First, the first *donor* cell is found for each point where the interpolation is needed. This is done locally on each process, facilitated with an octree search. If no local suitable *donor* cell is found for a particular *fringe* cell, the loop continues without gathering any *donor* cell. Once this is done, a parallel communication step is run to check if each interpolation point has one and only one *donor* cell associated. For each local *donor* cell found at the previous step, its neighbours are gathered as *donor* cells themselves, followed by the neighbours of the second *donor* cell etc. Parallel communication is needed after each set of neighbours gathered for the search to expand to the neighbouring parallel domain when needed. Finally, once enough *donor* cells have been gathered for each interpolation point, a last communication step is used to prepare future exchanges.

Listing 1: Pseudo code representing the *donor* search implementation in ReFRESCO

```
for cell in global_fringe_cells:
  find 1st donor cell

check duplicates & orphans                  # parallel exchange

while more donor cells needed:
  for point in local_fringe_cells:
    find next layer of donor cells
  exchange cell information                 # parallel exchange

exchange info to prepare future communication  # parallel exchange
```

This implementation minimises the amount of data being exchanged and favours a few larger parallel communications over many small communication steps. For example, in a 3D Cartesian mesh (each cell having 6 neighbours), if 20 *donor* cells are needed for each *fringe* cell, the `while` loop will be run 4 times in total, leading to 5 exchange steps. The parallel performance of the implementation is measured and analysed in section 5.

The computation of the interpolation itself can be done in two different modes. Either by directly computing the interpolated data for each *fringe* cell or, for supported schemes, by first computing interpolation weights associated with each *donor* cell and from them computing the interpolated quantity. The advantage of computing interpolation weights being that they can be reused, provided the different grids have not been modified, moved or adapted, leading computational time saving. For each interpolation point, the interpolation is performed by the MPI process on which the first *donor* cell was found. This offers some load balancing and minimises communications as *donor* cells mostly belong to the MPI process that computes the interpolation.

4 | VERIFICATION STUDY

For the code verification study, to highlight the interpolation step behaviour on a wide range of flows, two different test cases were chosen. First, the Poiseuille flow analytical test case is a diffusion-dominated steady pipe flow case, here run on a 2D mesh. Second, a more realistic and complex case, a high Reynolds number (so convection dominant) recirculation bubble RANS manufactured solution is used. This is a 3D unsteady test case which includes turbulence modelling, taking it closer to applications for which overset meshes are commonly used while still permitting an accurate assessment of the numerical errors.

4.1 | Poiseuille flow test case

4.1.1 | Case definition

The *Poiseuille* flow analytical formulation can be derived directly from the Navier-Stokes equations. Considering a 2D domain of length L in the x direction and height h in the y direction with the inlet at $x = 0$, outlet at $x = L$ and walls at $y = 0$ and $y = h$. After taking into account the different boundary conditions (non slip walls, and $P = 1$ at the outlet), the analytical solution is presented in Equation 1.

$$u_x(y) = \frac{Ph}{2\mu} y \left(1 - \frac{y}{h}\right) \quad , \quad u_y = 0 \quad , \quad \frac{\partial p}{\partial x} = -P \quad , \quad p(x) = -P(x - L) + 1 \quad . \tag{1}$$

In this study, the constant pressure gradient is $\partial p/\partial x = -8\mu$, the domain height is unity ($h = 1$) and kinematic viscosity is $\nu = 0.1$, resulting in a height based Reynolds number of $Re_h = 10$.

In order to test the interpolation in different conditions, six overset layouts were designed. These are meant to represent various aspects of real overset meshes, with variation in refinement, relative positioning of the meshes and grid topology. For each layout, 5 grid assemblies were generated with different refinements. Each grid assembly is composed of two grids:

1. a Background grid which is common to all layouts, which is a Cartesian grid of ratio 2:1 (see table 3)
2. and a Foreground grid, also a 2:1 rectangular grid but four times smaller than the Background grid. Its position and type is different for each layout:
    - L1: the Foreground grid is a Cartesian grid on which the cell size is the same as the one of the Background grid, it is also centred in both directions relative to the Background grid. In the overlap region there is a one-to-one match between cells of both grids.

- L2: the Foreground grid is similar to layout L1 but the global positioning of the grid is offset by about 0.694 cells in the x-direction and 0.416 cells in the y-direction (the absolute offset depending on the grid refinement).
- L3: the Foreground grid is similar to layout L1 but it is rotated by about 18.65 degrees around the centre of the domain (the decimals were randomly chosen in order to avoid cell alignments).
- L4: The Foreground grid is still Cartesian but the cells are 1.6 times larger in each direction compared to the Background grid. The grid has the same offset from the centre of the domain as L2.
- L5: The Foreground grid is 1.5 times finer in both directions than the Background grid. The grid has the same offset from the centre of the domain as L2.
- L6: The Foreground grid is unstructured composed of triangles, and its cell count is kept close to the cell count of L1. The Foreground grid has the same offset from the centre of the domain as L2.

It can be noted that, even if some of them are structured, the CFD code considers them as unstructured and does not take advantage of their topology.

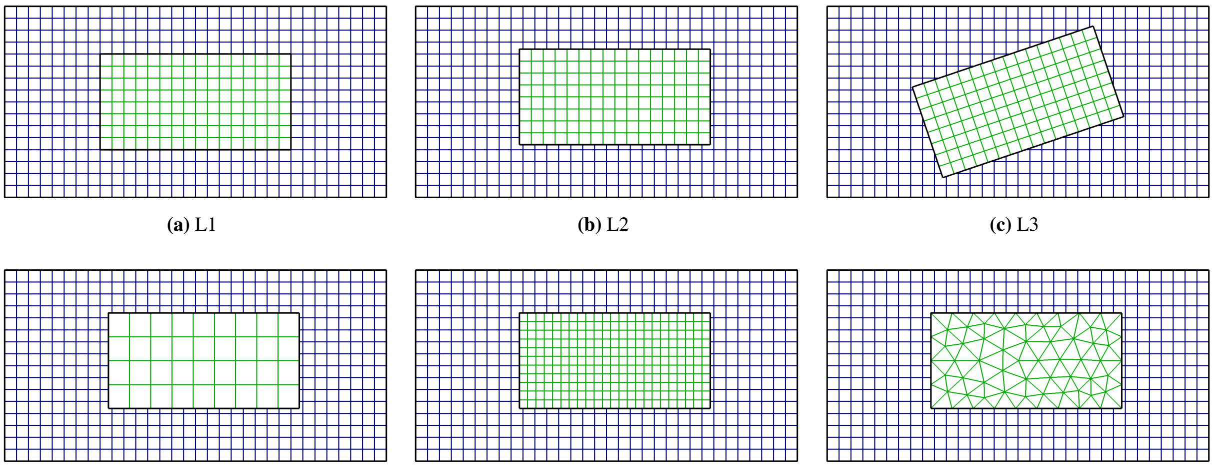
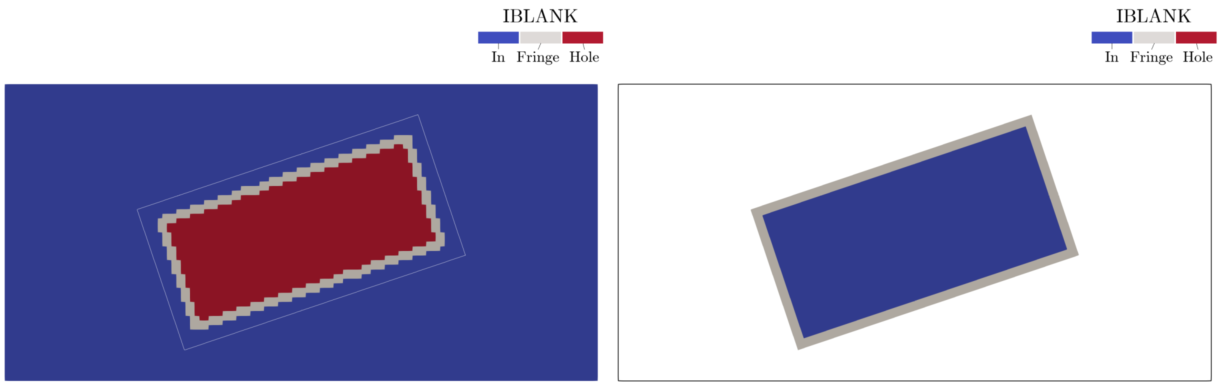


FIGURE 3 Coarsest grid used for each layout. The Background grid is displayed in blue and the Foreground grid in green.

	Background		Foreground							
	L{1-6}		L1 - L2 - L3		L4		L5		L6	
		N_i		N_i		N_i		N_i		N_i
G1	32×16	512	16×8	128	9×4	36	23×11	253	unstr	112
G2	64×32	2 048	32×16	512	19×9	171	47×23	1 081	unstr	518
G3	128×64	8 192	64×32	2 048	39×19	741	95×47	4 465	unstr	2 232
G4	256×128	32 768	128×64	8 192	79×39	3 081	191×95	18 145	unstr	9 234
G5	512×256	131 072	256×128	32 768	159×79	12 561	383×191	73 153	unstr	37 944

TABLE 3 Different grid sizes used in this study.

In this test case, steady computations are performed where only the momentum equation and the pressure correction (via the SIMPLE method) are solved. At the inlet, the analytical velocity profile is set, and pressure is enforced at the outlet. The top



(a) Background grid **(b)** Foreground grid

FIGURE 4 Cell status for the layout L3 and grid G3 computed by Suggar++.

and bottom of the domain use non-slip wall boundary conditions. Iterative convergence is ensured by letting the norm of the residuals fall below 10^{-14} for the infinity norm. Convective and diffusive fluxes are discretised using 2nd order schemes.

Figure 4 shows the IBLANK array for the L3 layout computed by Suggar++. Two layers of *fringe* cells are used at each boundary for ReFRESCO to reconstruct gradients properly on neighbouring *in* cells. Since the test case does not strictly require the use of overset meshes (there is no moving body, etc.) for each refinement, a computation using only the Background grid was additionally done for comparison purposes.
In this study, all the interpolation schemes presented in section 2.2 were tested: *Nearest cell* and *Inverse distance* using 4 and 10 *donor* cells as well as *Barycentric* interpolation of type 2. For polynomial-based interpolation, *Least squares* and *Polynomial* interpolation of degree 1, 2 and 3, as well as *Polynomial tensor* interpolation of degree 1 were used.

It should be noted that since the Poiseuille flow solution is a degree 2 polynomial function, a degree 2 or higher polynomial-based interpolation method should interpolate the field exactly.

The robustness of polynomial-based interpolation is heavily influenced by *donor* cell locations and is particularly sensitive to cell centre alignments. In this test case, *Polynomial* interpolation of order 3 and the *Polynomial tensor* interpolation lead to diverging computations due to their high errors. Results from these computations are not shown in the following study.

4.1.2 | Error level analysis

Figure 5 shows the infinity norm of the velocity field error for a selection of schemes (and the layout L3). For the *no overset* computation, this error is only the space discretisation error, that, as expected with 2nd order schemes, also decreases in 2nd order when refining the grid. On overset computations, however, the error shown is a combination of discretisation and overset interpolation errors. With the *Least squares* 2 scheme (a 3rd order scheme), the error level is similar to the *no overset* computation, suggesting that the interpolation error is lower than the discretisation error. Similarly, Figure 6 shows the infinity norm of the error for each grid per scheme as well as the order of convergence. This order is computed using a linear regression of the log of the errors for the different grid refinements.

- Layout L1: each scheme shows a converging trend with errors similar to the case run without overset except for the degree one *Least squares* and *Barycentric* type 3, which both result in higher errors. This is because neither of them guarantee that an interpolation computed on a cell centre returns the cell centre value. Even though only the velocity error is shown here, the error on the pressure shows similar behaviour.

- Layouts L2 to L5 show very comparable results for each scheme, giving confidence that the conclusions drawn here are not specific to the mesh layout. Every single scheme shows a converging trend when refining the grid with the exception of *Inverse distance* and *Nearest cell* on layouts L3 and L4 (both of them being only 1st order). Increasing the number of *donor* cells for the *Inverse distance* scheme slightly helps to reduce the error but not to the level of other, higher order, interpolation schemes. Then, 2nd order schemes (*Nearest cell gradient*, *Polynomial* 1 and *Least squares* 1), as well as the

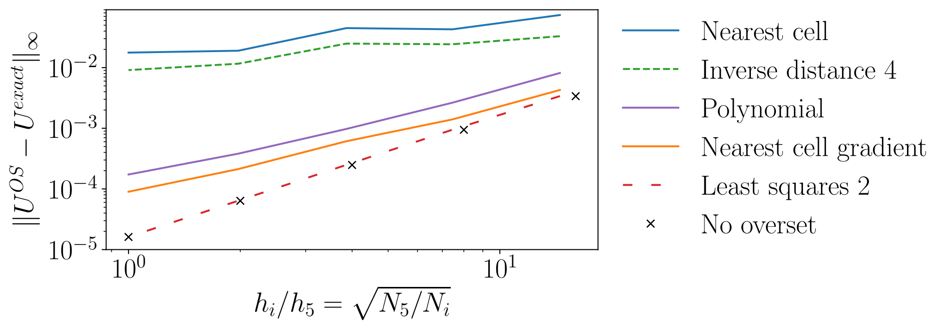


FIGURE 5 Infinity norm of the velocity field error against grid refinement (G5 to G1) for the layout L3. For readability reasons, only a selection of schemes is displayed.

Barycentric ones, show intermediate results, with proper convergence when refining the grids, but still showing errors higher than the discretisation error alone (*no overset* case). The convergence order computed from this error varies between 1 and 1.5. Degree 1 *Least squares* computations show slightly higher errors than the degree 1 *Polynomial* on these layouts, which can be explained by the fact that the *Least squares* approach uses more *donor* cells, thereby using field information further away from the interpolation location. Finally, schemes with theoretical order of 3 or higher (*Polynomial* 3 and *Least squares* 2 and 3) are the only one to show errors levels similar to the case without overset grids, suggesting that the interpolation error is lower than the discretisation error.

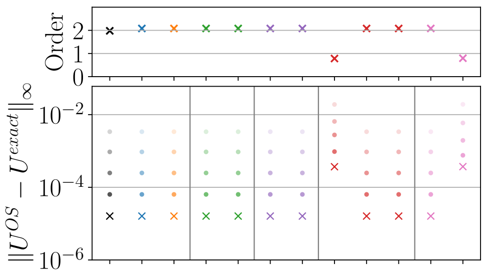
- Layout L6 is the only case introducing an unstructured grid for the Foreground mesh. Since triangular cells reduce the grid quality and increase the intrinsic discretisation error on *in* cells, the comparison with the non-overset case is harder to make. Nevertheless, *Nearest cell gradient* and *Least squares* have similar error levels for the finest grids, though degree 2 and 3 *Least squares* are better for coarser grids.
  The schemes showing the lowest error overall (*Least squares* 2 and 3) have worse convergence order on this layout. As mentioned, this convergence order combines both discretisation errors and overset interpolation error, each of them having their own order of convergence. On these computations, the discretisation error is likely to have a lower order than the interpolation one, thereby lowering the overall order of convergence. However, the *Nearest cell gradient* computation shows a different behaviour as the interpolation error is dominating (with an order of 1.5 like seen on the other layouts). The combined convergence order is 1.5. Upon refining the grids, the discretisation error will likely become dominant, lowering the combined convergence order.
  As stated previously, *Polynomial* based interpolation schemes are sensitive to *donor* cells location. On this layout, both *Polynomial* computations resulted in poor robustness with very high interpolation errors leading to the computation divergence.

4.1.3 | Mass imbalance study

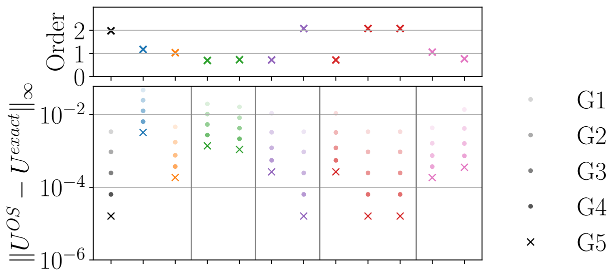
Upon convergence of the SIMPLE/SIMPLER algorithm, mass imbalance is of the same order as the iterative error. When using overset grids, however, the automatic global mass conservation that is inherent to the finite volume method is lost due to the overset errors, which is mainly due to interpolation. This is because the values of the face fluxes are calculated using interpolated values at the *fringe* cells. This source of mass imbalance does not depend on iterative error but rather on the interpolation error and is here sees decreasing with grid refinement.

Figure 7 shows the mass imbalance for each scheme. It was previously seen that on layout L1, overset computations did not show errors higher than the ones without overset. It is visible here, however, that this does not imply that the mass is fully conserved due to numerical errors.

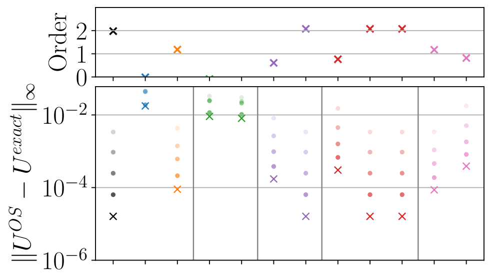
The L2 layout is a translation of L1. Therefore the *Nearest cell* scheme is computing exactly the same solution for the two layouts since the *donor* and *fringe* cells are the same, explaining the similar mass imbalance on these two layouts. The mass imbalance is several orders of magnitude lower than the error level shown in the previous section, which implies that schemes with comparable error magnitudes can have a different mass imbalance. This is, for example, the case for the degree 2 *Least*



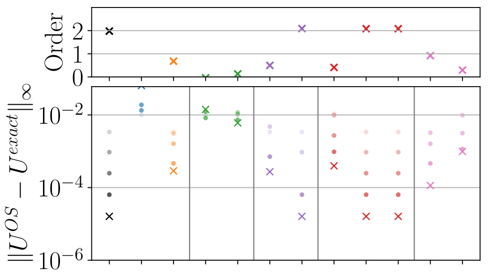
(a) L1



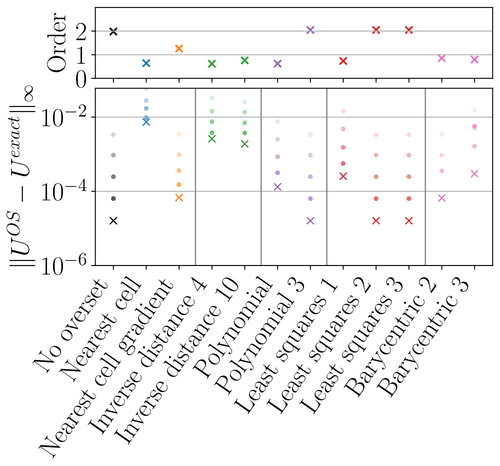
(b) L2



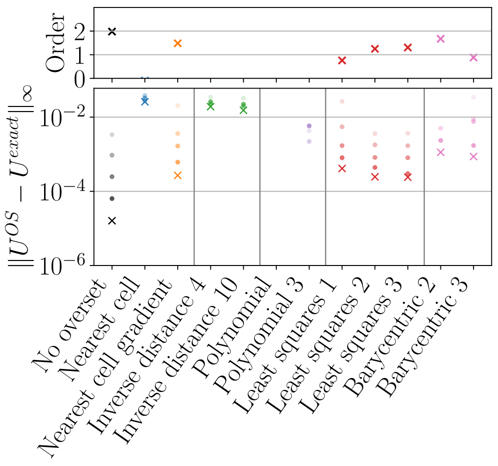
(c) L3



(d) L4



(e) L5



(f) L6

FIGURE 6 Infinity norm of the error and convergence order on the velocity for the Poiseuille case. Cross marker denote the finest grid.

squares scheme. Even if Figure 6 shows similar trends for layouts L2 and L3, the mass imbalance is three orders of magnitude lower on L2 for the finest grids. Similarly to the error plots, it can be seen that schemes with a theoretical order strictly higher than two behave similarly on all grid layouts.

4.1.4 | Flow behaviour and errors location

In this section, only the *Nearest cell*, *Nearest cell gradient*, and degree 2 *Least squares* schemes will be analysed as they can be considered representative of 1st, 2nd and 3rd order interpolation schemes respectively. Field visualisation for the degree 1 *Least squares* being relatively similar to the *Nearest cell gradient* one. Figure 9 shows, for the layout L3 and set of grids G3, the log of the difference between the exact solution and the overset computation for the velocity field. On these figures, only the bottom half of the Foreground mesh solution is displayed to visualise *fringe* cells of the Background grid that the Foreground mesh would otherwise overlap.

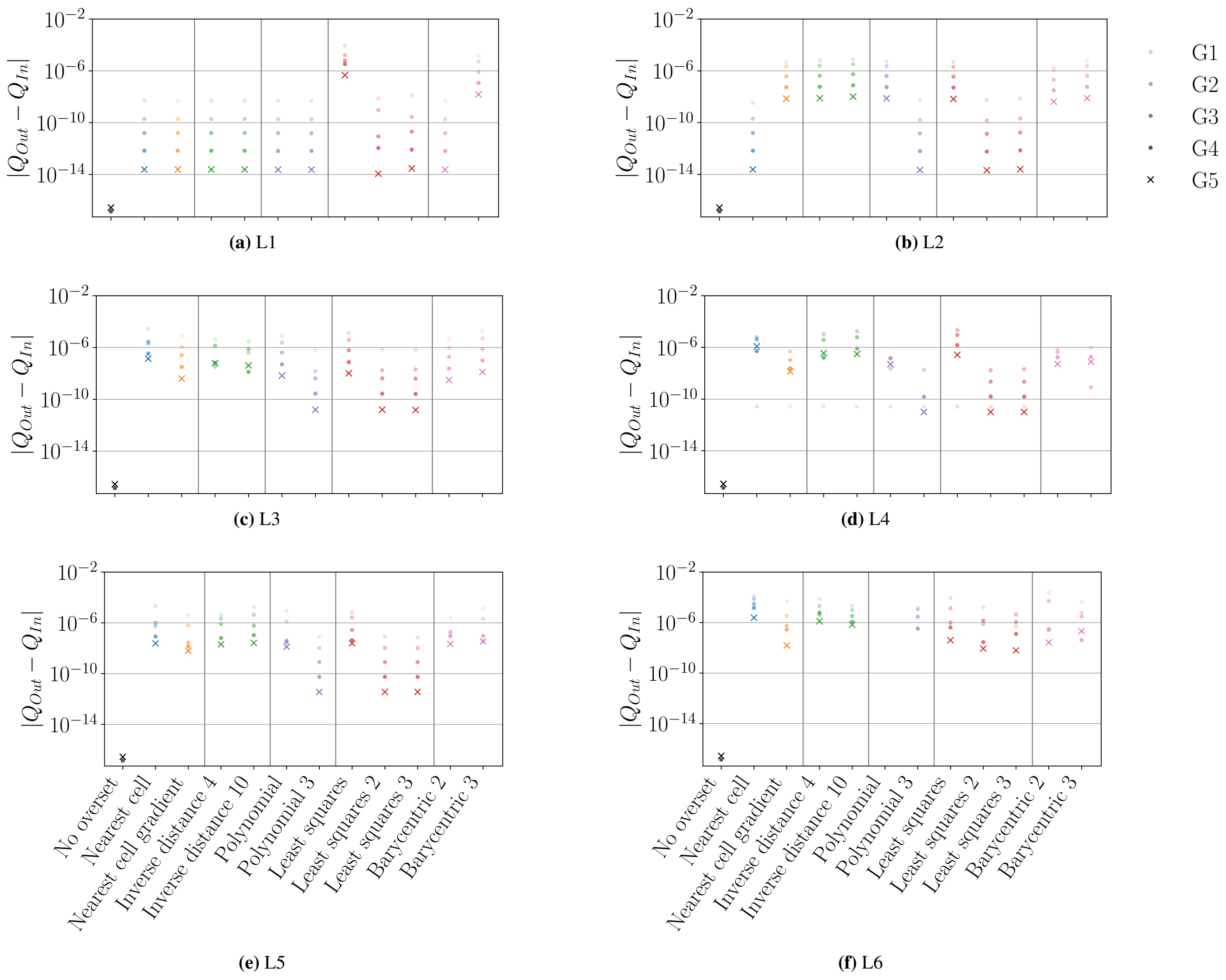
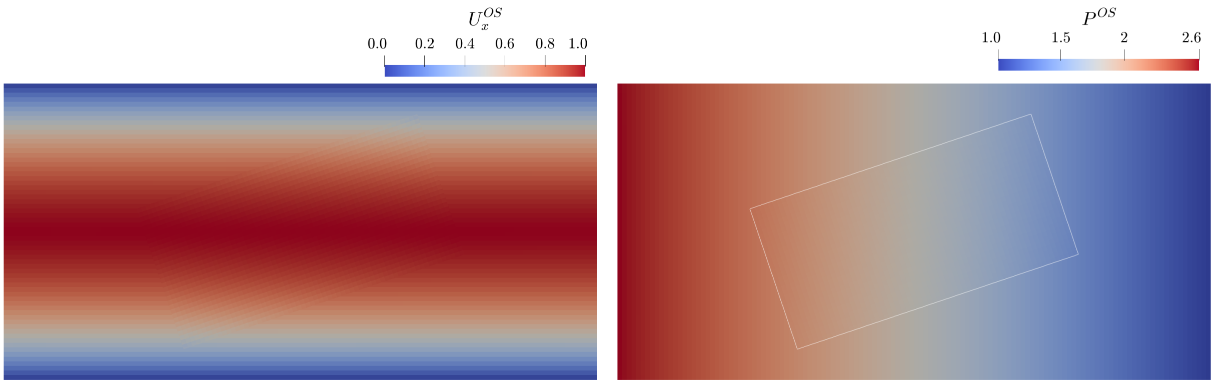


FIGURE 7 Difference between inflow and outflow mass fluxes for the Poiseuille case.

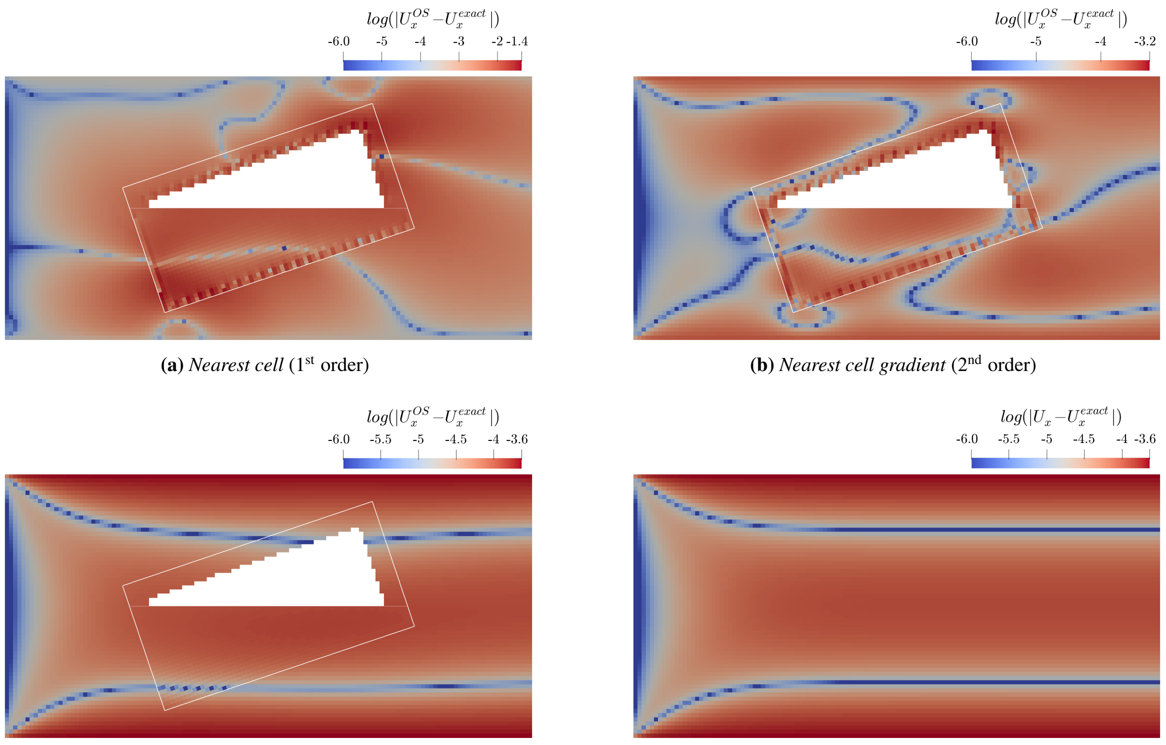
Nearest cell and *Nearest cell gradient* show similar error patterns: *fringe* cells have higher errors and can clearly be distinguished from the rest of the flow. With the degree 2 *Least squares* scheme, on the other hand, no discontinuities are visible even in the overlap region, and the highest errors are at the top and bottom boundaries and not in the overset region. For this scheme, the error is quite similar in terms of magnitude and location to a computation done without overset. Note that the *Nearest cell gradient* scheme has lower error levels than the *Nearest cell* scheme, as shown in Figure 6. The error on the pressure field, Figure 10, shows a comparable trend as degree 2 *Least squares* scheme does not display any artefact related to the overset process. The *Nearest cell gradient* scheme pressure field is smoother than the velocity one, but a step is still present between *fringe* cells and *in* cells, visible on both the Foreground and the Background mesh. Finally, the *Nearest cell* scheme shows point-to-point oscillations on *fringe* cells for both grids and presents higher errors.

The Background grid data can also be directly compared to the computation done without overset as shown in Figure 11 for the velocity field. For the *Nearest cell* and *Nearest cell gradient*, the maximum difference is similar to the error analysed above. In contrast, the *Least squares* 2 figure shows differences one order of magnitude lower than the error (computed when comparing against the analytical solution) and still has a smooth field without visible effects of the overset method. To summarise, artefacts of the overset method, specifically of the interpolation step, are visible for interpolation schemes of order 2 or lower. When using higher order schemes, the error made with overset grids is negligible compared to discretisation error without overset grids.



(a) Axial velocity field **(b)** Pressure field

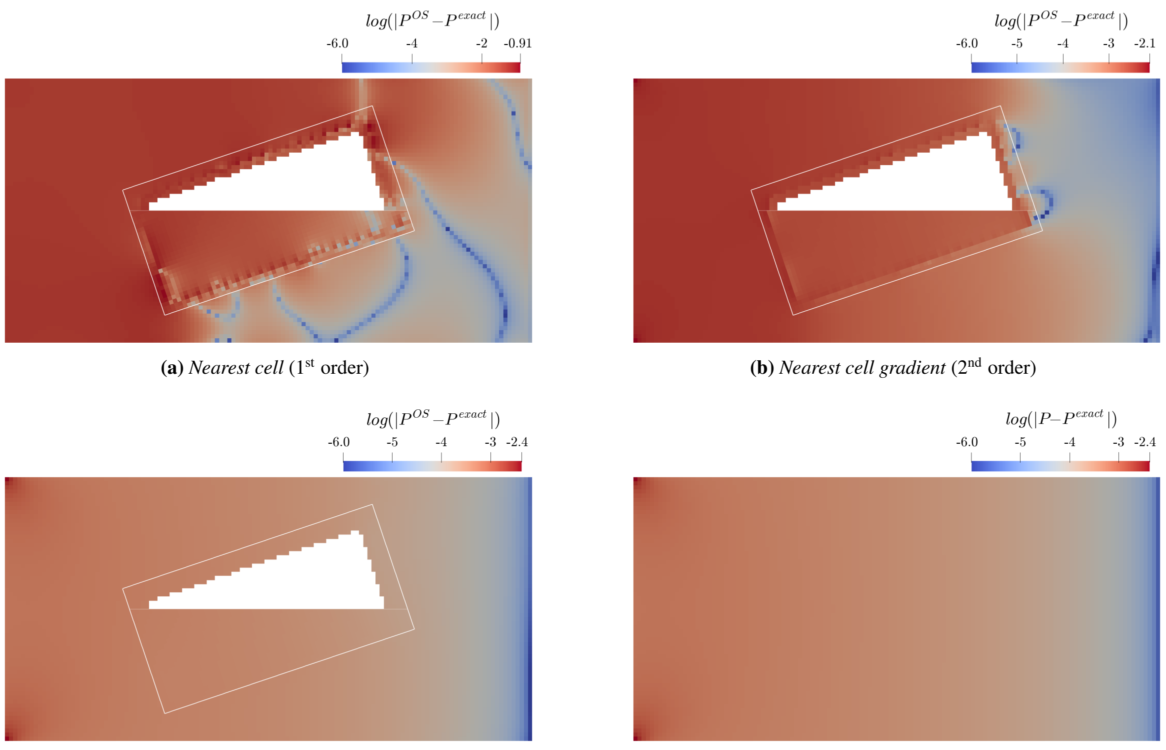
FIGURE 8 Velocity and pressure fields for the layout L3 and grid set G3, using the *Nearest cell gradient* interpolation scheme.



(a) *Nearest cell* (1st order) **(b)** *Nearest cell gradient* (2nd order)

(c) *Least squares* degree 2 (3rd order) **(d)** Computation without overset

FIGURE 9 Log of the error between overset computations and exact solution for the velocity field. Note that the scale is adapted for each plot with the maximum error always being the upper range of the scale. The Foreground mesh is only half visible in order to visualise *fringe* cells of the Background mesh.



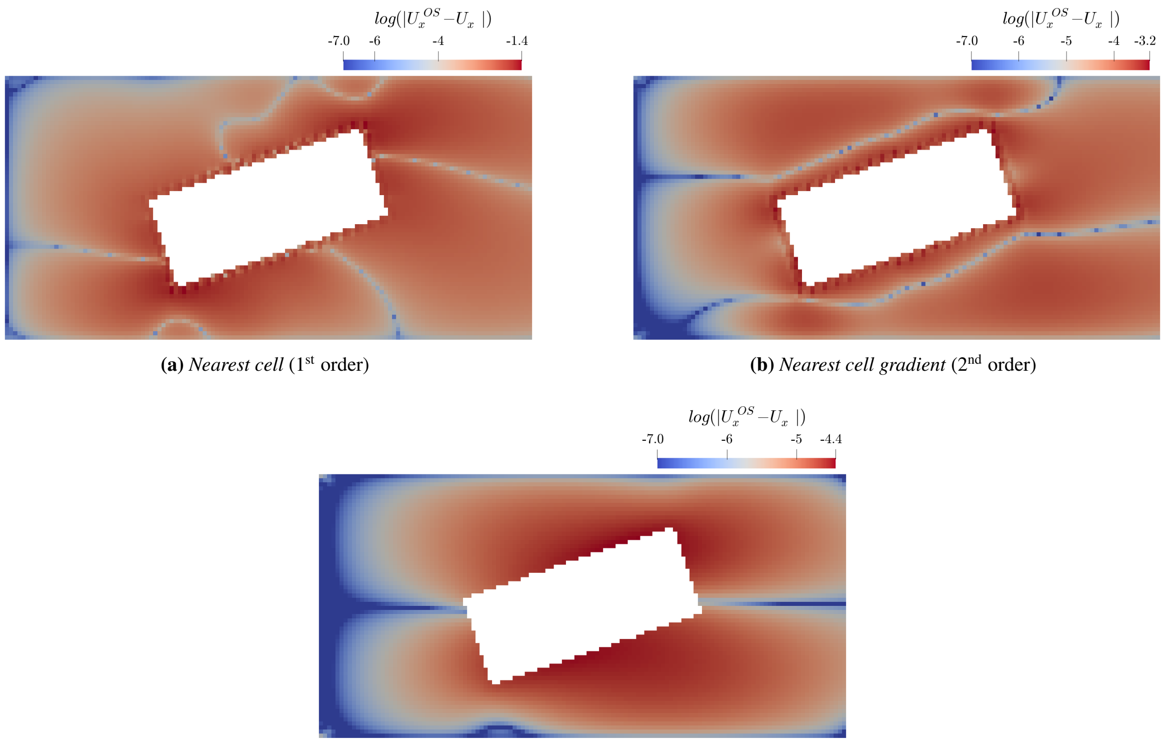
(a) *Nearest cell* (1st order)

(b) *Nearest cell gradient* (2nd order)

(c) *Least squares* degree 2 (3rd order)

(d) Computation without overset

FIGURE 10 Log of the error between overset computations and exact solution for the pressure field.



(a) *Nearest cell* (1st order)

(b) *Nearest cell gradient* (2nd order)

(c) *Least squares* degree 2 (3rd order)

FIGURE 11 Log of the difference in velocity between a computation done with and without overset.

4.2 | Recirculation bubble URANS manufactured solution

4.2.1 | Case definition

This test case, designed by Eça et al.[44], is a 3D unsteady manufactured solution using a high Reynolds number ($Re = 10^7$) recirculation bubble. The manufactured solution defines the velocity, pressure and $\tilde{\nu}_t$ turbulent eddy viscosity in the entire field. The bubble grows and disappears periodically following a sine wave with a period $T = 5$. Figure 12 shows a slice of the velocity field. The inlet is on the left of the domain and outlet on the right.

The source terms needed by the solver for this manufactured solution were computed with the library PyMMS[45], which generates a Fortran source file that can be integrated in ReFRESCO (or any CFD code). A detailed description of the manufactured solution can be found in appendix B and Eça et al.[44] should be read for an explanation of the design decisions that lead to the different fields.

The domain is an empty box of size $x = [0.1; 1]$, $y = [0; 0.4]$, and $z = [0; 1]$. The bottom boundary is a non-slip wall, the inlet a Dirichlet boundary condition with each quantity the analytical formulation, and every other boundary uses a Neumann boundary condition with the exact analytical gradient for each quantity. For this test case, two different overset mesh layouts were tested. In both these layouts, the Background grid of the size of the entire domain is used together with a smaller Foreground grid. The Foreground grid is a Cartesian mesh of size $0.3 \times 0.25 \times 0.8$. The Background grid is a structured mesh with a refinement towards the wall to lower y^+ which was kept below 1 for all grids here. Similarly to the Poiseuille flow test case, structured meshes are considered unstructured by the flow solver. In this study, 5 different refinements were used for both the Background and Foreground meshes, and Table 13 summarises the different dimensions and cell counts. The name of each grid layout (*Grid n*) can be seen as the number of cells in the x-direction for the Background grid, and every other dimension scales with it in order to have a set of geometrically similar grids.
The difference between the two overset layouts is only in the rotation of the Foreground grid, on layout L1, the Foreground grid is in the centre of the domain, and aligned with the global axis. When on layout L2, the Foreground grid is also centred in the domain but two rotations are applied to it. First in the y-direction of 18.90 degrees and then of 12.89 degrees in the z-direction. Both layouts are displayed in Figure 14. The DCI was computed by Suggar++ and is showed in Figure 15 for both grid layouts. Two layers of *fringe* cells are present at the boundaries and the Background grid has *hole* cells also surrounded by two layers of *fringe* cells.

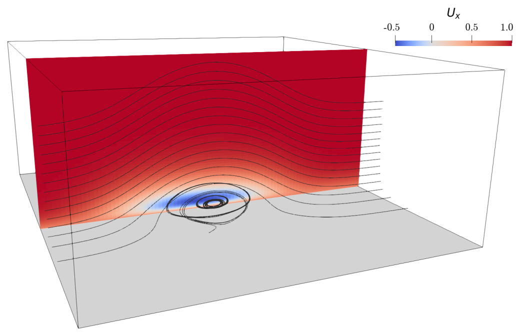


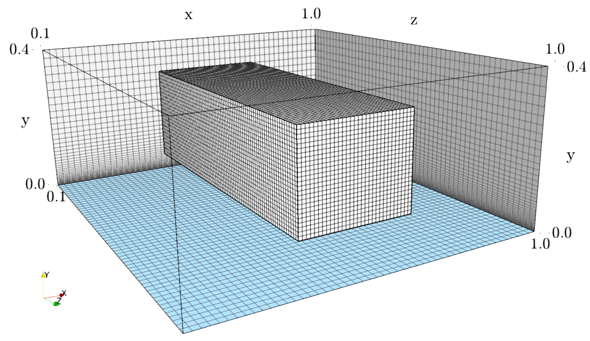
FIGURE 12 Recirculation bubble used as a manufactured solution, the slice is colored by the x axial velocity.

FIGURE 13 Details of the different grids used for the recirculation bubble test case.

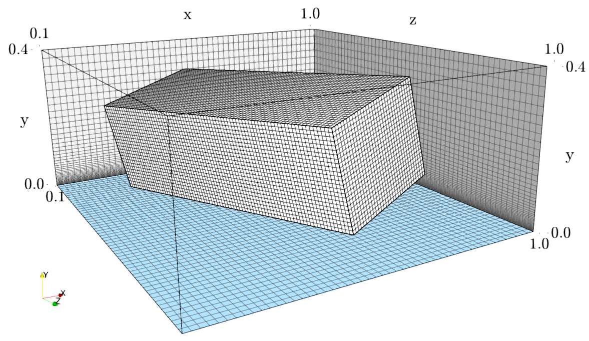
	Background		Foreground	
	$n_x \times n_y \times n_z$	N_i	$n_x \times n_y \times n_z$	N_i
Grid 50	$50 \times 80 \times 55$	220 000	$30 \times 25 \times 80$	60 000
Grid 60	$60 \times 96 \times 66$	380 160	$36 \times 30 \times 96$	103 680
Grid 70	$70 \times 112 \times 77$	603 680	$42 \times 35 \times 112$	164 640
Grid 80	$80 \times 128 \times 88$	901 120	$47 \times 40 \times 128$	240 640
Grid 90	$90 \times 144 \times 100$	1 296 000	$53 \times 45 \times 144$	343 440
Grid n	$n \times \frac{0.4n}{0.25} \times \frac{n}{0.9}$	$\approx \frac{16}{9} n^3$	$0.6n \times 0.5n \times 1.6n$	$\approx 0.48n^3$

For this study, the Spalart-Allmaras turbulence model[46] is used. Furthermore, the pressure-velocity coupling SIMPLER algorithm is used since it showed better iterative convergence characteristics than SIMPLE. Time integration is performed by an implicit three time level scheme (2nd order) and the 2nd order QUICK scheme is used for convective flux discretisation.

The period of the manufactured solution is set to $T = 5$ and the time-step is $t_{step} = 0.01$, leading to a maximum Courant number of 0.3 for the coarsest grid (Grid 50) and 0.7 for the finest grid (Grid 90). The validity of this time-step was checked by doing a time-step refinement study with $t_{step} = 0.05$, $t_{step} = 0.01$ and $t_{step} = 0.001$. With a time-step of $t_{step} = 0.01$, a period is modeled by 500 time-steps.

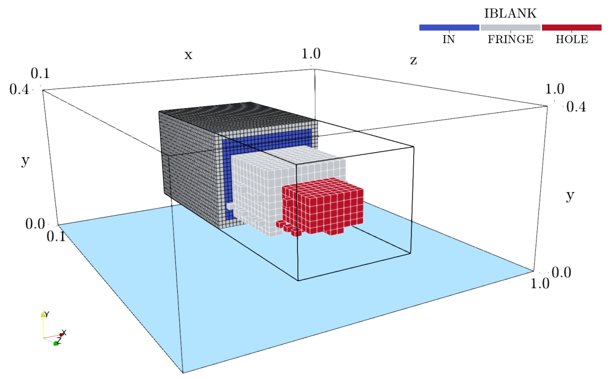


(a) L1

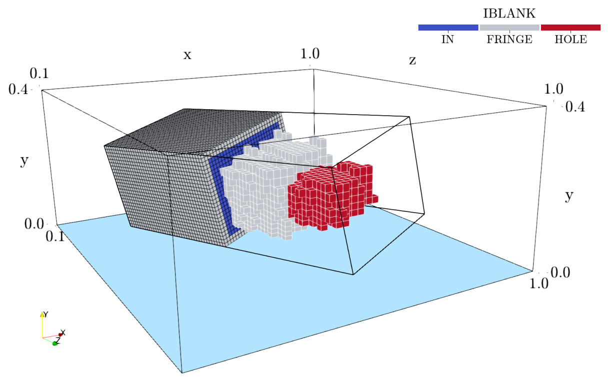


(b) L2

FIGURE 14 Coarsest grid (Grid 50) with the two different layouts used for the recirculation bubble test case.



(a) L1



(b) L2

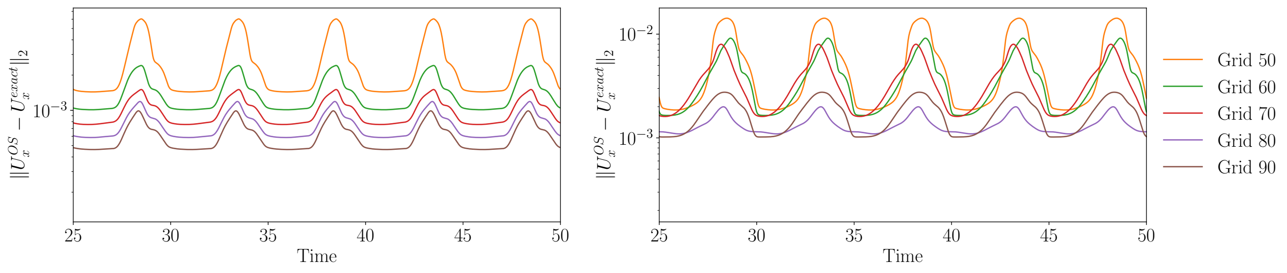
FIGURE 15 Overset domain connectivity as computed by Suggar++ on the coarsest grid setup (Grid 50). Black cells belong to the Foreground grid and white cells belong to the Background grid. One can see that the Foreground grid has two layers of *fringe* cells on its boundaries, and the Background grid has *hole* cells in the middle coated also by two layers of *fringe* cells.

The statistical uncertainty for any measured quantity is kept below 2% by simulating 5000 time-steps (10 periods). This uncertainty is analysed with the Transient Scanning Technique[47] using pyTST[48], and the first two periods of any computation is discarded to remove the startup transient effect (according to the transient scanning technique results).

For this study, the two Layouts L1 and L2, and all the five grid setups were computed using overset. For each grid combination, *Inverse distance* (using 7 *donor* cells), *Nearest cell gradient*, *Least squares* of degree 1, 2 and 3, and *Barycentric* interpolation of type 2 were used. For comparison purposes, a set of computations was also done using only the Background grid without overset grids.

Iterative error was controlled by performing enough outer-loops to keep the infinity norm of the residuals for all equations below 10^{-6}. The same computations were performed with infinity norm of residuals below 10^{-7} but no changes were seen on the computed error fields, therefore 10^{-6} was kept for the remainder of the study.

For this test case, the error norms were calculated every time-step for each field quantity. Figure 16a shows them for the *Nearest cell gradient* on layout L1. On this plot, the five different grid refinements are plotted, all showing similar behaviours. The error is oscillating in time with a period equal to the solution ($T = 5$). From the time traces one can see that differences between periods are negligible. This is also apparent in the low statistical uncertainty of the mean, of the order of 2%. The error also decreases with grid refinement, as one can expect. It is also noticeable that for each quantity, the maximum error occurs just after the middle of the period when the amplitude of the recirculation bubble reaches its maximum. These observations hold for the two layouts and all the quantities as well as the set of computations done without overset meshes. When using the *Inverse* *distance* interpolation scheme on layout L1, however, the behaviour is less clear as seen in Figure 16b. Here grid convergence is not clearly visible, as, for example, Grid 60 and 70 show similar errors. The amplitude is also not consistent between the different grid refinements.



(a) *Nearest cell gradient* **(b)** *Inverse distance*

FIGURE 16 L_2 norm of the error on U_x over 5 flow periods on layout L1. In this plot, the timetrace of each grid refinement is shown. Figure (a) shows a proper converging trend when it is not as clear when *Inverse distance* is used.

4.2.2 | Error level analysis

Figure 17 plots the time-averaged L_2 norm of the error for each quantity, each interpolation scheme, each grid setup and both layouts. In this study, the L_2 norm is used as it was found to be more representative of the situation compared to the L_∞ norm (because the maximum error was often linked to the domain boundary and not the overset assembly). Considering the error regarding U_x for layout L1 (first column and first line), each scheme shows grid convergence, with finer grids leading to lower errors, except for the *Inverse distance* scheme (in green). Computations done without overset show error levels similar to the *Nearest cell gradient*, degree 2 and 3 *Least squares*, and *Barycentric* interpolation approaches. It is noticeable that the errors without overset are even slightly higher than when using the *Nearest cell gradient* on layout L1. This is because the Foreground grid is more refined than the background grid. Hence the overset computations lead to lower discretisation errors when compared to the non overset ones. This is not as true anymore for layout L2, where the Foreground grid is rotated, resulting with part of it within the lower boundary layer. In this region, the Background grid is more refined than the Foreground grid, leading to larger discretisation errors. Overall, for layout L2, the two effects cancel each other out, and the error levels are similar without overset and with a *Nearest cell gradient* scheme.

Consistently for each quantity and both layouts, the *Nearest cell gradient* and *Barycentric* schemes perform as well as the non-overset computations. The degree 2 and 3 *Least squares* interpolation perform here, at best, as well as the *Nearest cell gradient* in layout L1, but slightly worse in layout L2. As mentioned before, on layout L1, the *Inverse distance* scheme shows the highest error levels and no clear grid convergence trend. This implies that the error due to the interpolation is higher than the discretisation error. For layout L2, however, the *Inverse distance* scheme performs better, with lower errors and a converging trend, suggesting that the scheme is more sensitive to the *donor* cells' location than the other tested ones. The degree 1 *Least squares* scheme always performs worse than the degree 2 one, and the degree 2 and 3 *Least squares* schemes perform comparably. Summarising, from the error plots only, the *Nearest cell gradient*, degree 2 and 3 *Least squares* and *Barycentric* schemes are the best candidates as they add no or a minimal amount of error in the field compared to the case without overset being used. This suggests that the interpolation error is lower than the discretisation error.

Regarding the convergence order, the computations done without overset show a 2nd order convergence for each quantity, as expected from the theoretical order of discretisation of the schemes used, and therefore verifying the non-overset solution. The *Nearest cell gradient*, degree 2 and 3 *Least squares* and *Barycentric* computations also show a 2nd order convergence as well. However, the *Inverse distance* scheme shows, depending on the quantity and layout, between 1st and 2nd order convergence. Finally, the degree 1 *Least squares* results in 2nd order convergence for layout L1 but slightly lower order on L2.

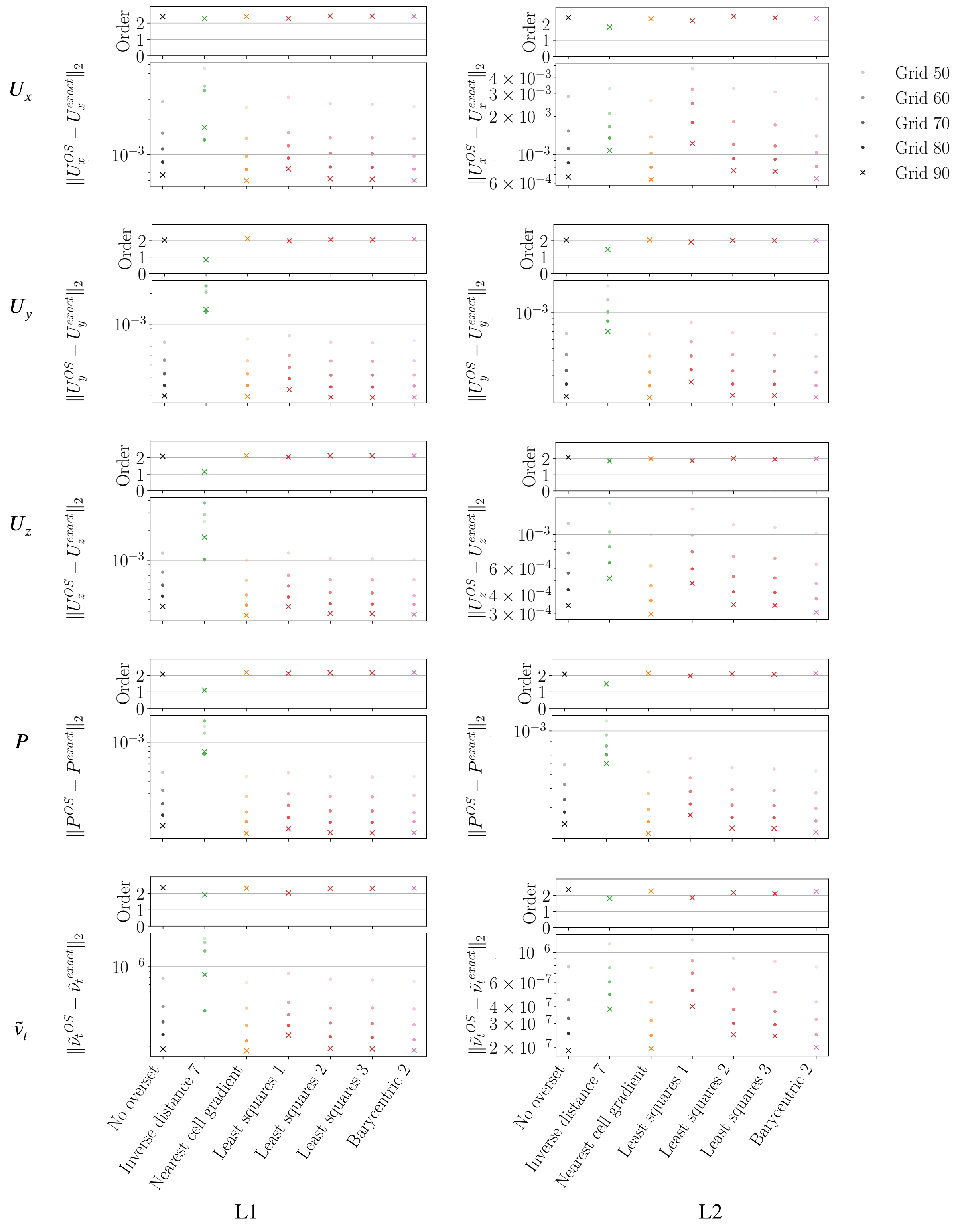


FIGURE 17 Comparison of error levels for each quantity depending on the interpolation scheme used for the recirculation bubble test case. Errors plotted are the time average of the L_2 norm of the error for each quantity. Convergence order is displayed above each quantity.

4.2.3 | Mass imbalance study

Figure 18 shows the mass imbalance for the different computations. Without overset meshes, the imbalance is capped by the iterative error and is around 10^{-7} for every grid. When using overset meshes, if, like in this study, no particular treatment is done to the overset interface, interpolation errors introduce mass imbalance. Independently of the scheme being used, the relation between cell sizes and mass imbalance is complex: finer meshes not always leading to lower errors. For the two second order schemes (*Nearest cell gradient* and *Least squares* 1), Grid 50 to 80 shows a converging trend on layout L1, but Grid 90 has higher mass imbalance. This trend is also not present on layout L2. None of the other schemes show monotonic convergence trend and *Inverse distance* has, in general, the highest errors.

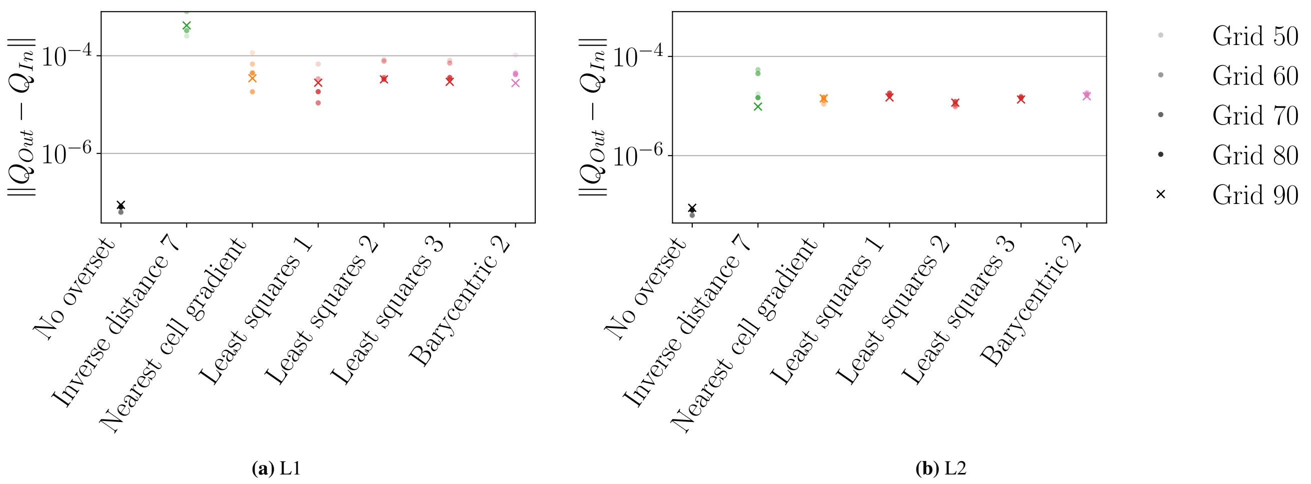


FIGURE 18 Mass imbalance for different interpolation schemes for the two tested grid layouts on the recirculation bubble test case.

4.2.4 | Flow behaviour and error location

Snapshots of the entire field solution were saved at $t = 497.5$ when the recirculation bubble is largest and, as seen in the previous section, when errors are also higher. In this section only the layout L2 is being analysed in detail, though similar conclusions can be drawn from layout L1.

Figure 19 shows the velocity field when using the *Inverse distance* scheme, where no artefacts from the overset interpolation are visible in the field. When looking at the error field for the velocity, however, the effects of the overset meshes become apparent. In Figure 20, the log of the velocity error is plotted for each computation using Grid 80. The lowest errors are visible upstream of the overset region for all of the computations. When using the *Inverse distance* scheme, it is particularly visible that the error made in the overset region is being convected downstream with the flow. Some artefacts of the overset interpolation are also visible downstream of the Foreground mesh when using the *Nearest cell gradient* interpolation, but less than with *Inverse distance*. On the Foreground mesh itself, both the *Inverse distance* and *Nearest cell gradient* computations show a chequerboard error pattern following the flow streamlines. This is because the interpolation error created at the upstream *fringe* cells of the Foreground grid is being convected downstream. Neither scheme produces a smoothed field, meaning that two neighbouring *fringe* cells can have very different interpolated data. This creates artefacts still visible downstream of the layer of *fringe* cells. The *Least squares*, on the other hand, produces a much smoother interpolation, and higher order method makes the field smoother, leading to less visible artefacts of the interpolation, even though the error levels are comparable with the *Nearest cell gradient* scheme. When comparing the *Least squares* and Non overset computation, one can notice that, in the area where the Foreground grid is, the error is lower in the overset computation, which confirms that having a finer overset mesh helps with decreasing the discretisation error. Moreover, the fact that the *inlet* part of the Foreground grid is better aligned with the flow streamlines may also help lower the errors for the overset computations in this region. Finally, even when no overset meshes are used, the recirculation bubble region is where the highest errors are located.

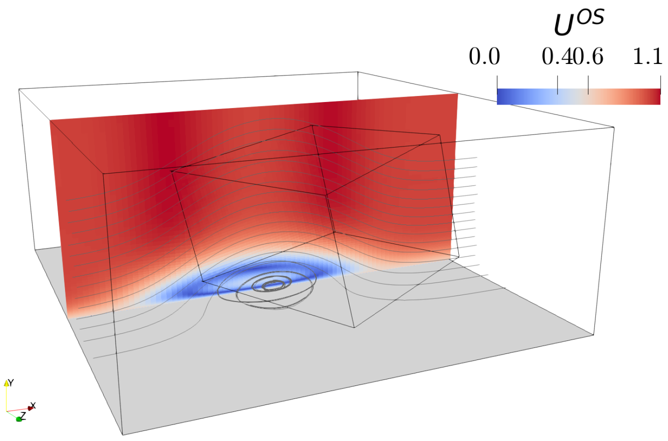
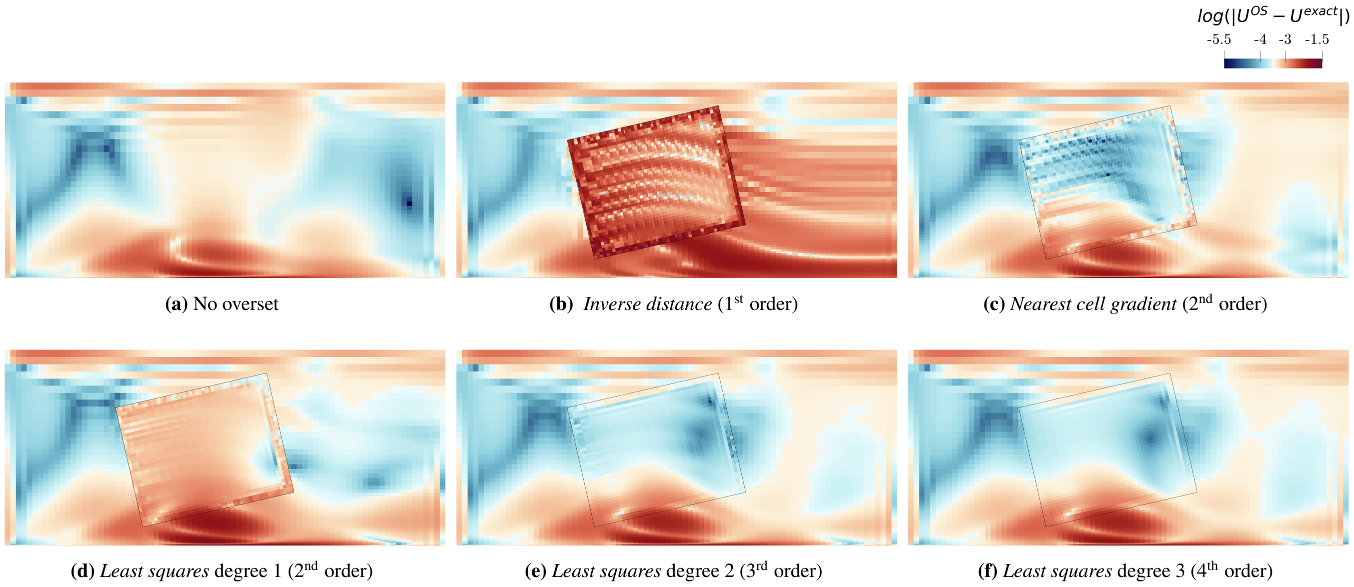


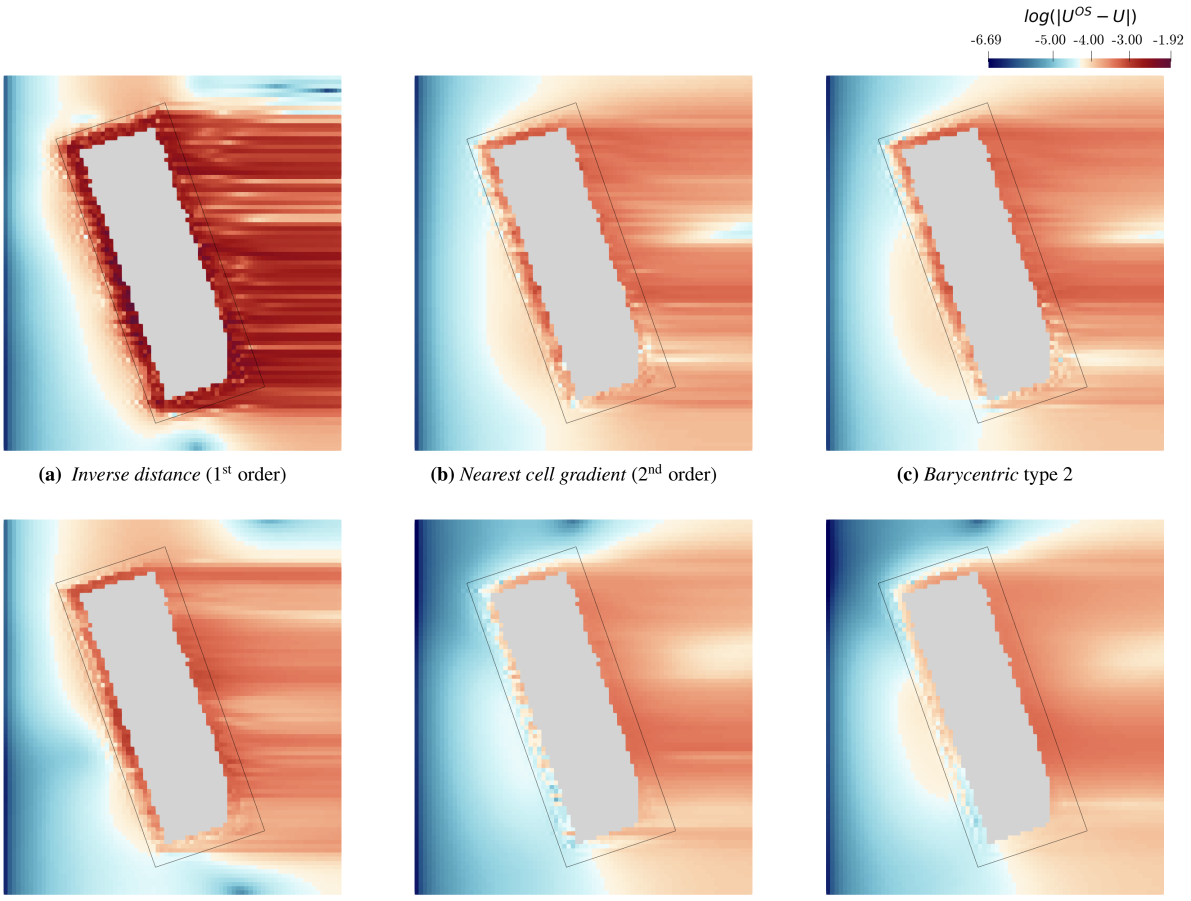
FIGURE 19 Velocity magnitude slice for the *Inverse distance* computation on Grid 80



(a) No overset
(b) *Inverse distance* (1st order)
(c) *Nearest cell gradient* (2nd order)
(d) *Least squares* degree 1 (2nd order)
(e) *Least squares* degree 2 (3rd order)
(f) *Least squares* degree 3 (4th order)

FIGURE 20 Side view of the domain (inlet on the left, wall at the bottom) showing the error made on the velocity magnitude for layout L2. The slice is taken in the middle of the domain ($z = 0.5$).

In order to analyse only the overset method error, Figure 21 shows the difference between overset and non overset computations on the Background grid. Compared to previous error plots, what stands out is that, even with the degree 2 *Least squares* interpolation, interpolation errors are being convected. Even though the *Least squares* produces a smoother field compared to the *Nearest cell gradient* one, errors are still present. Interpolation artefacts are also visible on the *fringe* cells upstream of the Foreground mesh.



(a) *Inverse distance* (1st order)
(b) *Nearest cell gradient* (2nd order)
(c) *Barycentric* type 2
(d) *Least squares* degree 1 (2nd order)
(e) *Least squares* degree 2 (3rd order)
(f) *Least squares* degree 3 (4th order)

FIGURE 21 Top view of the domain (inlet on the left) showing the difference between overset and non overset computations on the velocity for layout L2. The slice is taken in the middle of the domain ($y = 0.2$).

5 | PERFORMANCE ANALYSIS

The overset method has several sources of performance overhead arising from the new routines (DCI and interpolation), additional parallel communication and from deterioration of the iterative convergence.
Both the DCI and interpolation information have to be updated every time the meshes move, deform or are refined. Additionally, each outer-loop, the interpolation requires parallel MPI communication to perform the mesh coupling. In this work, the DCI is computed by the external library Suggar++, and the interpolation, as well as *donor* search, are done by the CFD solver. Due to the modification of the discretisation and the matrices that the overset method requires, especially when explicit coupling is used, the non-linear iterative convergence is influenced. This translates into slower iterative convergence, increasing the number of outer-loops to reach a similar level of iterative error.
This section analyses these three sources of overhead.

5.1 | Interpolation performance

5.1.1 | Methodology and setup

Every scheme available in ReFRESCO and presented in section 2.2 is tested here, as well as the *donor* searching method. Computations were run on MARIN's cluster Marclus4, a 4200 cores cluster hosted inside Marin facilities. Each computational node includes a dual-socket Intel Xeon (E5-2660 v3 @ 2.60GHz) 10 cores CPU, leading to 20 cores per node and Infiniband HBA is used for inter-node communications. The code is compiled using Intel compiler and Intel MPI for the parallelisation

(2018.4). For each computation, the time of each routine was recorded using PETSc[49] solver logging capabilities, and, on parallel runs, the time of the slowest process was used.

Interpolation methods and *donor* searching algorithms are tested on Grid 160 for the recirculation bubble test case (presented in section 4.2.1). To have sufficient statistical convergence on the timing results, each function was run at least 500 times and the timing includes the computation itself as well as all the parallel communication required. To prevent data caching between consecutive runs, functions are not run in isolation inside a real CFD computation (of the recirculation bubble test case). Table 4 shows the interpolation schemes, and parameters examined. The setup reflects a computation that requires a new *donor* search and interpolation every time-step (due to mesh movement, for example) and an update of the interpolation every outer-loop. 90 outer-loops are used per time-step. *Least squares*, *Inverse distance* and *Nearest cell* interpolation use interpolation weights that are computed once per time-step and reused within each outer-loop, while other schemes need the interpolation to be fully recomputed every outer-loop (note that the *donor* search is done only once per time step).

Each computation is repeated using 1-1000 processes, as summarised in Table 5.

TABLE 4 Interpolation schemes and parameter tested in this study

Scheme	Nb *donor* cells
Nearest cell	1
Nearest cell gradient	1
Inverse distance	7
	15
Least squares degree 1	6
Least squares degree 2	20
Least squares degree 3	34
Barycentric type 2	1

TABLE 5 Average number of cells and *fringe* cells per process. Computations up to 20 cores are run on a single node.

N_{proc}	N_i/N_{proc}	N_{fringe}/N_{proc}
1	9 216 000	259 837
2	4 608 000	129 918
10	921 600	25 983
20	460 800	12 991
40	230 400	6 495
80	115 200	3 247
160	57 600	1 623
300	30 720	866
460	20 034	564
620	14 864	419
800	11 520	324
1000	9 216	259

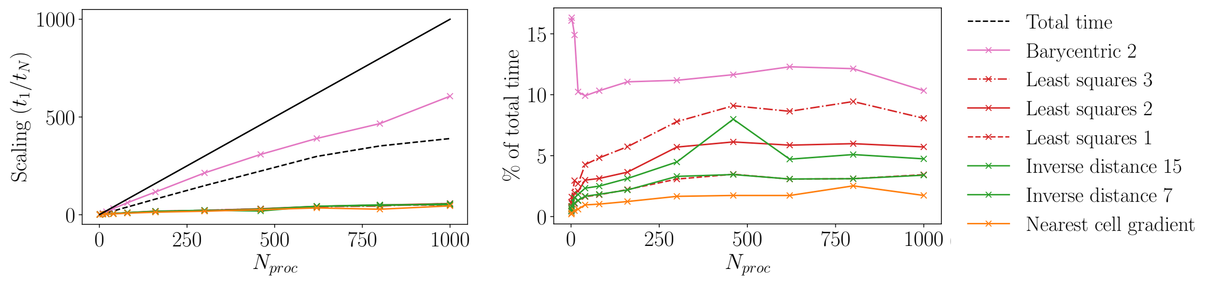
For comparison purposes, individual timings of functions are expressed as a percentage of a baseline set of computations for which the timing of overset and interpolation methods has been subtracted. The ratio between the interpolation time ($T_{interpolation}$) and the time taken by the rest of the computation ($T_{total} - T_{interpolation}$) is used. This means that this number could be higher than 100 if the time taken by the interpolation is higher than the time taken by the rest of the computation. Similar measures are computed for the *donor* searching method.

$$\% \ of \ total \ time = 100 \times \frac{T_{interpolation}}{T_{total} - T_{interpolation}} \tag{2}$$

5.1.2 | Results

Figure 22a shows the strong scalability of interpolation schemes. The wall clock time is scaled based on the serial computation (using a single core). Apart from the scalability of the different schemes, the *Total time* line shows the scaling of the rest of the code. As the solid line denotes ideal scalability, the CFD code is seen to degrade slightly faster after 600 cores.

Amongst the schemes tested, the *Barycentric* interpolation is the one that scales the best, with constant scalability even at a high core count. All the other schemes show similar levels of poor parallel scalability. This scalability behaviour is explained by the way the parallelisation is implemented. Most finite volume method CFD codes use domain decomposition, where each core gets only a part of the full grid. In the current implementation, when using overset meshes, the interpolation is computed by the processor that stores the domain where the first *donor* cell is located. Since the *donor* cells are not scattered evenly in the entire domain, different loads are experienced on different cores, thereby affecting the scalability. This is shown in Figure



(a) Strong scalability **(b)** Timing relative to Total time

FIGURE 22 Scalability and timing of different interpolation schemes compared to ReFRESCO computation without overset (dashed line on scalability plot). The solid black line on shows ideal scalability.

23 where the most loaded core (in number of interpolations to perform) scalability is plotted against the number of cores. This number depends only on the mesh, the layout and the domain decomposition but not on anything hardware specific. From this plot, the close relation between the most loaded core and the scalability of the interpolation is seen, suggesting that the poor scalability is only related to the load balancing of the interpolation but not on parallel communication, for example. This means that distributing the load over a wider range of cores will improve the scalability. One should also note that the scalability is still quite consistent because of this behaviour, forming an almost straight line on this plot.

On the other hand, the *Barycentric* interpolation is using a slightly different parallelisation method compared to the other schemes, which makes it scale very well, but it also makes it more time-consuming. Figure 22b shows the relative time each interpolation takes compared to the full CFD computation. The *Barycentric* interpolation is seen to be the most expensive out of all the tested schemes, especially at low core counts. It can also be seen that the poor scalability of all the other schemes does not impact their timing. Overall, except for the *Barycentric* interpolation, the interpolation does not need more than 8% of the total time, with a minimum of 2% for the *Nearest cell gradient*. This makes using 2$^{\text{nd}}$ and 3$^{\text{rd}}$ order schemes affordable.

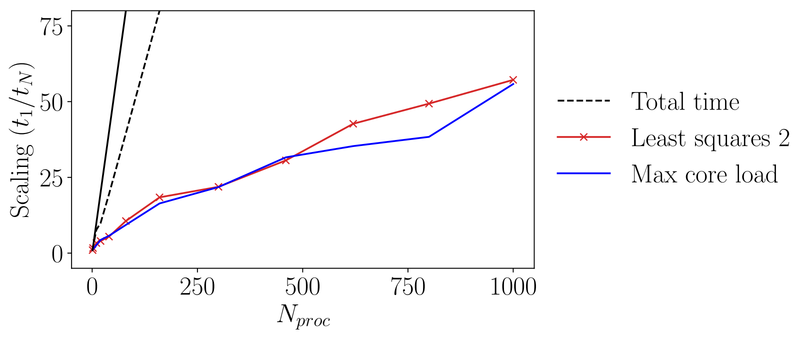
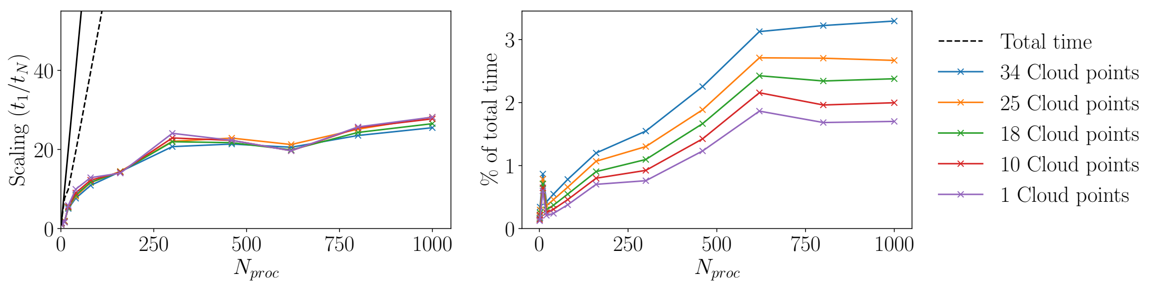


FIGURE 23 Scalability of the *Least squares* interpolation and of the number of interpolation computed by the most loaded core. Due to imperfect load balancing, all the cores are not loaded similarly. The interpolation computations the most loaded core has to perform scales very similarly to the timing of the interpolation. This means that the load balancing can be improved in this implementation.

The computation of the interpolation is only possible if *donor* cells have been collected. Figure 24a shows the scalability of the *donor* searching algorithm. Similarly to the interpolation, the actual time taken by the algorithm is almost negligible, as shown in Figure 24b with a maximum below 3% of the total CFD computation time. Furthermore, once the first *donor* cell is gathered, the subsequent *donor* cells are a lot cheaper to gather. For example, requesting 15 *donor* cells is only about 20% more expensive than requesting a single *donor* cell. This again makes schemes that need more *donor* cells affordable.

Finally, Figure 25 shows the combined time taken by both the *donor* searching and the interpolation itself. Overall, the fastest scheme is the *Nearest cell gradient* since it needs only a single *donor* cell and is trivially easy to compute, i.e. no system to solve



(a) Strong scalability

(b) Timing relative to Total time

FIGURE 24 Scalability and timing of the *donor* searching methods to ReFRESCO computation without overset (dashed line on scalability plot). The solid black line on shows ideal scalability.

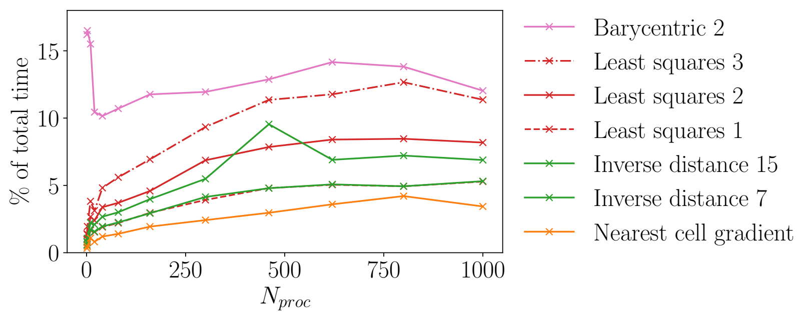


FIGURE 25 Time taken by both the *donor* searching and interpolation for all the tested schemes.

as with polynomial based interpolation. More complex schemes like *Least squares*, however, only take 8% of the total time at most for a degree 2 (3$^{\text{rd}}$ order scheme).

5.2 | DCI computation

The DCI is computed by the serial external library Suggar++. In the computations presented in section 4, Suggar++ was only run as a preprocessing step. Figure 26 shows the time taken if it were run every time-step as a percentage of the CFD computation, similar to what was presented in the previous section. Even if Suggar++ can work with both structured and unstructured meshes, in this section, structured meshes are treated as unstructured by the library. The lack of scalability is apparent from increasing relative time compared to the CFD computation, and goes up to 30% when 1000 cores are used. Figure 27 displays Suggar++ runtime per grid with different refinements, making it clear that it scales linearly with the cell count.
Several methods can be employed to reduce the time taken by the computation of the DCI. Primarily, if the motion is predefined, the DCI can be computed in a preprocessing step since it only depends on the relative location of the different meshes. Otherwise, methods like the one presented by Martin et al.[50] can be implemented, where a couple of processing cores are dedicated to the computation of the DCI. Each time-step, an estimate of the mesh position is made, and a DCI is computed. That way, the CFD solver does not wait for the serial computation of the DCI.

5.3 | Iterative convergence

Using overset meshes modifies the discretisation and inherently the matrices, both left and right hand sides, of the linear system of equations. This has an impact on the performance of the iterative solvers used and also in the non-linear iterative convergence. In this section, the number of outer-loops for reaching a L_∞ norm of 10^{-6} for the recirculation bubble test case is analysed. In this work, the momentum and turbulence equations were solved using the GMRES algorithm, and the pressure correction was solved with a CG solver. A block Jacobi pre-conditioner was used in all cases. It has to be noted that every computation done

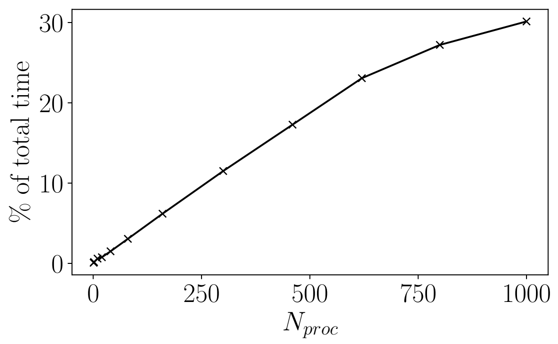


FIGURE 26 Time taken by Suggar++ when computing the DCI at every time-step in percentage of ReFRESCO computational time when running the recirculation bubble test case on Grid 160.

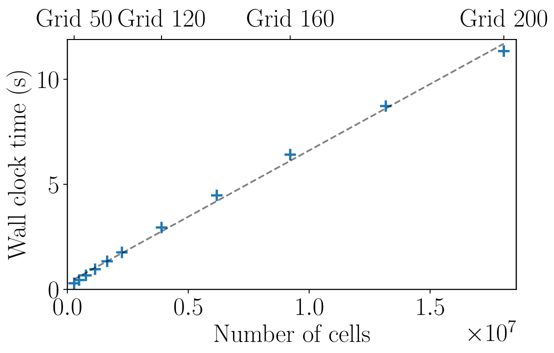


FIGURE 27 Wall clock time taken by Suggar++ to compute the DCI depending on the grid cell count. This plot is using layout L2 of the recirculation bubble test case set of grid.

here uses the same under-relaxation parameters which may not be optimal, and individual runs could potentially be tweaked to have better convergence behaviours. Also, one should note that this implementation uses an explicit coupling, meaning that the *fringe* cell values are being updated only every outer-loop, and do not directly depend on the *donor* cells values. This is known to impact the iterative behaviour compared to an implicit coupling as the different meshes are not coupled as tightly.

Figure 28 shows the average number of outer-loops needed for each computation as well as the difference with the non-overset run. Overall, layout L1 has better iterative convergence as, consistently, the different schemes need fewer outer-loops to converge than on layout L2. Based on these results though, it is hard to draw a clear conclusion for each scheme. It seems that the convergence order or accuracy of the scheme does not play a significant role in the iterative convergence as the *Inverse distance* scheme converges as well as a degree 2 *Least squares*. The degree 1 *Least squares* on layout L2 here shows a poorer convergence though. It was also shown to have higher errors in section 4. Tests done with different under-relaxation coefficients show different trends, with *Inverse distance* and *Nearest cell gradient* performing better than the *Least squares* approach. It is therefore hard to conclude in a general way for each interpolation scheme, but it is clear that the overhead in performance due to the iterative error is not negligible.

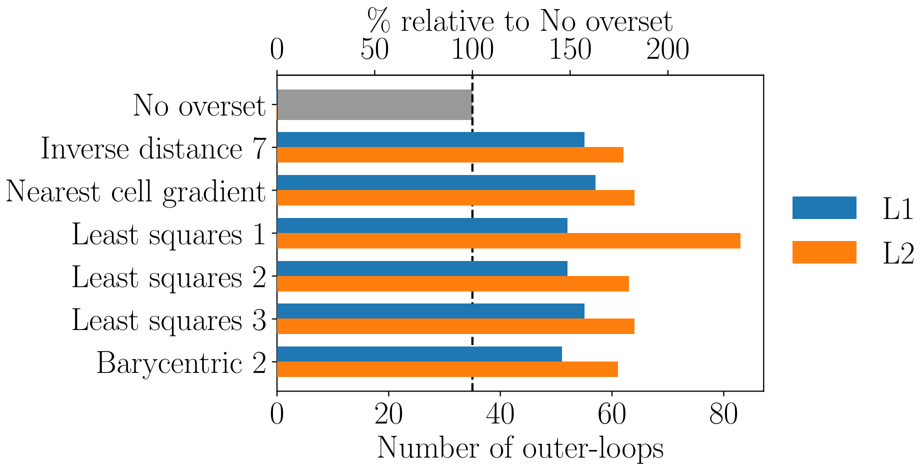


FIGURE 28 Average number of outer-loops needed to converge each time-step to L_∞ residuals of 10^{-6} for each equation. Results are presented for layout L1 and L2 of the recirculation bubble manufactured solution on Grid 80.

6 | CONCLUSIONS

In this work, a wide range of overset-grid interpolation schemes were studied, in the context of unstructured grid discretisation, 2nd order accurate in space and time, finite volume CFD codes. Accuracy, robustness and computational performance have been investigated and quantified by means of code verification exercises. Schemes widely used in the open literature of the subject, and others, ranging from 1st to 4th order were tested. Two test cases have been considered: a low Reynolds number 2D steady Poiseuille laminar flow and a 3D unsteady high Reynolds number manufactured solution of a boundary layer including a recirculation bubble.

For both cases the influence of grid layouts (background and foreground grids), grid refinement, time-step has been investigated. Local and global errors, convergence orders and mass imbalance have been quantified, and a comparison between solutions with and without overset grids was also made. In terms of computational performance, strong scalability, cpu timings, *donor* search and interpolation operations, parallel load unbalancing studies and DCI overhead were reported. Additionally, the effect of the overset-grid interpolation schemes on the numerical performance of the solver, i.e. number of non-linear iterations to achieve convergence, has been also comprehensively analysed.

The following detailed conclusions regarding accuracy and robustness can be drawn:

- Overall, higher order schemes produce lower interpolation errors than lower order ones, as expected. 1st order schemes, like *Inverse distance*, generate errors that affect considerably the underlying discretisation schemes of the flow solver. On the other end of the spectrum, 3rd order schemes, in this study, were always enough to maintain the 2nd order convergence of the global discretisation.
- The interpolation schemes behave differently depending on the flow field smoothness. This is due to the number and location of *donor* cells and the interpolation scheme itself. For example, *Nearest cell gradient* or *Barycentric* interpolation only use local information, whereas a similar order *Least squares* interpolation can use more than 10 *donor* cells. Therefore, the *Least squares* approach was found to produce the smoothest field among all schemes tested. Its robustness, which can be controlled by increasing the number of *donor* cells used, and its accuracy, makes it an ideal candidate for interpolation in an overset context.
- The two test cases were designed to study different features and test the overset implementation and interpolation schemes under different conditions. First, the recirculation bubble manufactured solution is a high Reynolds test case ($Re = 10^7$) compared to the Poiseuille flow one ($Re = 10$). As a result, convection plays a predominant role in the recirculation bubble case, and interpolation errors produced by the overlap are convected downstream. However, on diffusion dominant flows, like the Poiseuille test case, the overset assembly perturbed the entire domain (even upstream of overlap). This difference in the transport of the errors also impacts the smoothness of the field. Indeed, a low Reynolds number leads to a smooth field regardless of the interpolation scheme. However, at high Reynolds number, the wake of the overset meshes is as smooth as the interpolation on *fringe* cells.
- The two test cases also differentiate in how 2nd order interpolation scheme perform. With the recirculation bubble test case, 2nd order interpolation results in interpolation errors lower than the discretisation ones. However, on the Poiseuille flow test case, a 3rd order or higher interpolation scheme is required to achieve accurate results.

Regarding computational performance the following remarks can be made:

- Amongs the different components affecting the performance of an overset computation, in the tested implementation (DCI done externally), the lack of parallel performance of the DCI computation is likely to become the bottleneck on larger grids and more complex grid assemblies.
- The *donor* search and interpolation itself use at most 8% of the computational time for a 3rd order interpolation scheme (*Least squares* of degree 2). And, if a 2nd order interpolation is enough, the *Nearest cell gradient* scheme can be used as it halves the computational cost (3-4% of the total runtime). This makes higher order schemes affordable and recommendable.
- The use of an explicit coupling for the overset assembly degrades the iterative convergence, increasing the number of outer-loops required to reach the desired residuals level. On the recirculation bubble test case, 1.5 to 2 times more nonlinear outer-loops were needed compared to a non overset computation. This performance overhead is significant, and has to be addressed in the short term.

We emphasise that in this work pure code *verification* exercises have been done since an overset grid method does not improve, nor should it degrade, the modelling errors of a specific solution. Therefore the overset grid method cannot be validated in isolation but only verified. For this purpose, close-to-real application manufactured solutions are essential to quantitatively assess an overset grid implementation prior to perform complex validation exercises. Finally, while this paper focused on non-moving overset meshes, for future work a similar study should be done to expand the conclusions for cases with grid and bodies having predefined and/or arbitrary motions.

Acknowledgements

This research was financially supported by the EPSRC Centre for Doctoral Training in Next Generation Computational Modelling (EP/L015382/1) at the University of Southampton including financial support from MARIN. The authors acknowledge the use of the IRIDIS High Performance Computing Facility, and associated support services at the University of Southampton as well as MARIN HPC Marclus4 and WavEC resources and facilities, in the completion of this work.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Benek J, Buning PG, Steger J. A 3-D chimera grid embedding technique. In: *7th Computational Physics Conference* American Institute of Aeronautics and Astronautics; 1985; Reston, Virigina: 10

2. Carrica PM, Ismail F, Hyman M, Bhushan S, Stern F. Turn and zigzag maneuvers of a surface combatant using a URANS approach with dynamic overset grids. *Journal of Marine Science and Technology* 2013; 18(2): 166–181. doi: 10.1007/s00773-012-0196-8

3. Chen HC, Chen M. Chimera RANS simulation of a berthing DDG-51 ship in translational and rotational motions. *International Journal of Offshore and Polar Engineering* 1998; 8(3): 182–191.

4. Hubbard B, Chen Hc, Springs C. A Chimera Scheme for Incompressible Viscous Flows with Application to Submarine Hydrodynamics. In: *25th AIAA Fluid Dynamics Conference* ; 1994; Colorado Springs, CO; United States.

5. Chandar DD, Boppana B, Kumar V. A Comparative Study of Different Overset Grid Solvers Between OpenFOAM, Star-CCM+ and Ansys-Fluent. In: *2018 AIAA Aerospace Sciences Meeting* No. January. American Institute of Aeronautics and Astronautics; 2018; Reston, Virginia

6. Deng GB, Leroyer A, Guilmineau E, Queutey P, Visonneau M, Wackers J. CFD Simulation of PMM motion in shallow water for the DTC container ship. In: *4th International Conference on Ship Manoeuvring in Shallow and Confined Water with Special Focus on Ship Bottom Interaction* ; 2016; Hamburg, Germany: 93–98

7. Schreck E, Peric M. Overset Grids in STAR-CCM+: Methodology, Applications and Future Developments. In: *STAR Japanese Conference* ; 2012.

8. Völkner S, Brunswig J, Rung T. Analysis of non-conservative interpolation techniques in overset grid finite-volume methods. *Computers & Fluids* 2017; 148: 39–55. doi: 10.1016/j.compfluid.2017.02.010

9. Gatin I, Vukcevic V, Jasak H, Lalovic I. Manoeuvring simulations using the overset grid technology in foam-extend. In: *32nd Symposium on Naval Hydrodynamics* No. August. ; 2018; Hamburg, Germany: 10.

10. Brazell MJ, Sitaraman J, Mavriplis DJ. An overset mesh approach for 3D mixed element high-order discretizations. *Journal of Computational Physics* 2016; 322: 33–51. doi: 10.1016/j.jcp.2016.06.031

11. Lemaire S, Vaz G, Turnock SR. On the Need for Higher Order Interpolation with Overset Grid Methods. In: *22th Numerical Towing Tank Symposium (NuTTS)* No. 1. ; 2019; Tomar, Portugal: 1–6.

12. Chesshire G, Henshaw W. Composite overlapping meshes for the solution of partial differential equations. *Journal of Computational Physics* 1990; 90(1): 1–64. doi: 10.1016/0021-9991(90)90196-8

13. Martin JE, Noack RW, Carrica PM. Overset grid assembly approach for scalable computational fluid dynamics with body motions. *Journal of Computational Physics* 2019; 390: 297–305. doi: 10.1016/j.jcp.2019.04.009

14. Carrica PM, Huang J, Noack RW, Kaushik D, Smith B, Stern F. Large-scale DES computations of the forward speed diffraction and pitch and heave problems for a surface combatant. *Computers and Fluids* 2010; 39(7): 1095–1111. doi: 10.1016/j.compfluid.2010.02.002

15. Noack RW, Boger DA. Improvements to SUGGAR and DiRTlib for Overset Store Separation Simulations. In: *47th AIAA Aerospace Sciences Meeting* No. January. ; 2009; Orlando, Florida: 5–9

16. Sitaraman J, Floros M, Wissink A, Potsdam MA. Parallel domain connectivity algorithm for unsteady flow computations using overlapping and adaptive grids. *Journal of Computational Physics* 2010; 229(12): 4703–4723. doi: 10.1016/j.jcp.2010.03.008

17. Schloegel K, Karypis G, Kumar V. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience* 2002; 14(3): 219–240. doi: 10.1002/cpe.605

18. Chandar DD. On overset interpolation strategies and conservation on unstructured grids in OpenFOAM. *Computer Physics Communications* 2019; 239: 72–83. doi: 10.1016/j.cpc.2019.01.009

19. Sharma A, Ananthan S, Sitaraman J, Thomas S, Sprague MA. Overset meshes for incompressible flows: On preserving accuracy of underlying discretizations. *Journal of Computational Physics* 2021; 428: 109987. doi: 10.1016/j.jcp.2020.109987

20. Sitaraman J, Jude D, Brazell MJ. Enhancements to Overset Methods for Improved Accuracy and Solution Convergence. In: *AIAA Scitech 2020 Forum* No. January. American Institute of Aeronautics and Astronautics; 2020; Reston, Virginia

21. Roache PJ. Code Verification by the Method of Manufactured Solutions. *Journal of Fluids Engineering* 2002; 124(1): 4–10. doi: 10.1115/1.1436090

22. Eça L, Hoekstra M. Verification and validation for marine applications of CFD. *International Shipbuilding Progress* 2013; 60(1-4): 107–141. doi: https://doi.org/10.3233/ISP-130083

23. Lemaire S, Vaz G, Turnock SR. Implementation and Verification of an Explicit Overset Grid Method. In: *21th Numerical Towing Tank Symposium (NuTTS)* ; 2018; Cortona, Italy.

24. Eça L, Vaz G, Hoekstra M. Code Verification of ReFRESCO With a Statistically Periodic Manufactured Solution. In: *ASME 33rd International Conference on Ocean, Offshore and Arctic Engineering* American Society of Mechanical Engineers; 2014: V002T08A015

25. Vaz G, Jaouen F, Hoekstra M. Free-Surface Viscous Flow Computations: Validation of URANS Code FreSCo. In: *Volume 5: Polar and Arctic Sciences and Technology; CFD and VIV* ASMEDC; 2009: 425–437

26. Schuiling B, Windt J, Rijpkema D, Terwisga vT. Computational Study on Power Reduction By a Pre-Duct for a Bulk Carrier. *Tokyo Workshop 2015* 2015.

27. Noack RW. SUGGAR: a general capability for moving body overset grid assembly. In: *17th AIAA Computational Fluid Dynamics Conference* . 5117. ; 2005; Toronto, Ontario, Canada: 1–21

28. Noack RW, Wyman NJ, McGowan G, Brown C. Dual-Grid Interpolation for Cell-Centered Overset Grid Systems. In: *AIAA Scitech 2020 Forum* . 1 PartF. American Institute of Aeronautics and Astronautics; 2020; Reston, Virginia: 1–32

29. Boger DA, Dreyer J. Prediction of Hydrodynamic Forces and Moments for Underwater Vehicles Using Overset Grids. In: *44th AIAA Aerospace Sciences Meeting* No. January. American Institute of Aeronautics and Astronautics; 2006; Reston, Virigina: 1–13

30. Shen Z, Wan D, Carrica PM. Dynamic overset grids in OpenFOAM with application to KCS self-propulsion and maneuvering. *Ocean Engineering* 2015; 108: 287–306. doi: 10.1016/j.oceaneng.2015.07.035

31. Windt C, Davidson J, Chandar DD, Ringwood JV. On the Importance of Advanced Mesh Motion Methods for WEC Experiments in CFD-based Numerical Wave Tanks. In: Ringsberg RB, J. ., eds. *VIII International Conference on Computational Methods in Marine Engineering MARINE 2019* International Center for Numerical Methods in Engineering (CIMNE); 2019; Gothenburg, Sweden: 145–156.

32. Kobayashi H, Kodama Y. Developing Spline Based Overset Grid Assembling Approach and Application to Unsteady Flow Around a Moving Body. *Journal of Mathematics and System Science* 2016; 6(9): 339–347. doi: 10.17265/2159-5291/2016.09.001

33. Gopalan H, Jaiman R, Chandar DD. Flow Past Tandem Circular Cylinders at High Reynolds Numbers using Overset Grids in OpenFOAM. In: *53rd AIAA Aerospace Sciences Meeting* American Institute of Aeronautics and Astronautics; 2015; Reston, Virginia: 1–20

34. Chandar DD. Assessment of Interpolation Strategies and Conservative Discretizations on Unstructured Overset Grids in OpenFOAM. In: *2018 AIAA Aerospace Sciences Meeting* No. January. American Institute of Aeronautics and Astronautics; 2018; Reston, Virginia: 1–15

35. Quon EW, Smith MJ. Advanced data transfer strategies for overset computational methods. *Computers & Fluids* 2015; 117: 88–102. doi: 10.1016/j.compfluid.2015.04.023

36. Quon EW, Smith MJ. Advanced Interpolation Techniques for Overset CFD. In: *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* No. January. American Institute of Aeronautics and Astronautics; 2012; Reston, Virigina

37. Vukcevic V, Jasak H. Overset Mesh Library in Foam-Extend. 2018.

38. Noack RW. DiRTlib: A library to add an overset capability to your flow solver. In: *17th AIAA Computational Fluid Dynamics Conference* No. June 2005. American Institute of Aeronautics and Astronautics; 2005; Toronto, Ontario, Canada: 1–20

39. Kiris CC, Housman JA, Barad MF, Brehm C, Sozer E, Moini-Yekta S. Computational framework for Launch, Ascent, and Vehicle Aerodynamics (LAVA). *Aerospace Science and Technology* 2016; 55: 189–219. doi: 10.1016/j.ast.2016.05.008

40. Ohashi K, Hino T, Kobayashi H, Onodera N, Sakamoto N. Development of a structured overset Navier–Stokes solver with a moving grid and full multigrid method. *Journal of Marine Science and Technology* 2019; 24(3): 884–901. doi: 10.1007/s00773-018-0594-7

41. Nguyen VT, Vu DT, Park WG, Jung CM. Navier–Stokes solver for water entry bodies with moving Chimera grid method in 6DOF motions. *Computers and Fluids* 2016; 140: 19–38. doi: 10.1016/j.compfluid.2016.09.005

42. Foster N, Noack RW. High-Order Overset Interpolation Within An OVERFLOW Solution. In: *50th AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition* No. January. American Institute of Aeronautics and Astronautics; 2012; Reston, Virigina: 1–8

43. Boger DA, Noack RW, Paterson E. Dynamic Overset Grid Implementation in OpenFOAM. In: *5th OpenFOAM Workshop* . 21. ; 2010: 24.

44. Eça L, Hoekstra M, Vaz G. Manufactured solutions for steady-flow Reynolds-averaged Navier-Stokes solvers. *International Journal of Computational Fluid Dynamics* 2012; 26(5): 313–332. doi: 10.1080/10618562.2012.717617

45. Lemaire S. PyMMS: Generation of RANS Manufactured Solution for CFD using Sympy. 2021. doi: 10.5281/zenodo.4428181

46. Spalart PR, Allmaras S. A one-equation turbulence model for aerodynamic flows. In: *30th Aerospace Sciences Meeting and Exhibit* American Institute of Aeronautics and Astronautics; 1992; Reston, Virigina

47. Brouwer J, Tukker J, Klinkenberg Y, Rijsbergen vM. Random uncertainty of statistical moments in testing: Mean. *Ocean Engineering* 2019; 182(April): 563–576. doi: 10.1016/j.oceaneng.2019.04.068

48. Lemaire S, Klapwijk M. pyTST: Python library and command line tool performing the Transient Scanning. 2021. doi: 10.5281/zenodo.4428158

49. Abhyankar S, Brown J, Constantinescu EM, Ghosh D, Smith BF, Zhang H. PETSc/TS: A Modern Scalable ODE/DAE Solver Library. *arXiv* 2018; V(212): 1–29.

50. Martin JE, Michael T, Carrica PM. Submarine Maneuvers Using Direct Overset Simulation of Appendages and Propeller and Coupled CFD/Potential Flow Propeller Solver. *Journal of Ship Research* 2015; 59(1): 31–48. doi: 10.5957/JOSR.59.1.140053

APPENDIX

A INTERPOLATION SCHEMES PRESENTATION

In this study, seven different interpolation schemes are tested. For consistency, the location of the interpolation is denoted by $\boldsymbol{x} = (x, y, z)$, the cell centre of a *fringe* cell. N is the number of *cloud points* needed by the scheme, in an overset context, a *cloud point* is a *donor* cell centre. The *cloud point* locations are $\boldsymbol{x_i}$ with $1 \leq i \leq N$. Finally, ϕ is the field to be interpolated and the interpolated field is named $\tilde{\phi}$. The goal of the interpolation is to get $\tilde{\phi}(\boldsymbol{x})$ using $\boldsymbol{x_i}$ and $\phi(\boldsymbol{x_i})$ for $1 \leq i \leq N$.

A.1 *Nearest cell*

This scheme has only one *donor* cell and the interpolated value is equal to the *donor* cell centre value. It is a first order scheme.

$$\tilde{\phi}(\boldsymbol{x}) = \phi(\boldsymbol{x_1}). \tag{A1}$$

A.2 *Nearest cell gradient*

This scheme also has only one cell in its *cloud point*, but it also uses the gradient at the *donor* cell centre to correct the interpolated value. The gradient used here comes from the Gauss method, and does not need to be computed specifically for the interpolation as it already computed by the flow solver. The use of the gradient makes it a second order scheme.

$$\tilde{\phi}(\boldsymbol{x}) = \phi(\boldsymbol{x_1}) + \nabla\phi(\boldsymbol{x_1}) \cdot (\boldsymbol{x} - \boldsymbol{x_1}). \tag{A2}$$

A.3 *Inverse distance*

The *Inverse distance* interpolation scheme is sometimes called weighted average method. The number of *donor* N can be freely chosen and the interpolation is defined as follow:

$$\tilde{\phi}(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} \omega_i \phi(\boldsymbol{x_i})}{\sum_{i=1}^{N} \omega_i}. \tag{A3}$$

$$\omega_i = \frac{1}{||\boldsymbol{x} - \boldsymbol{x_i}||^p}. \tag{A4}$$

The weights ω_i decay with the inverse of the distance to the interpolation location $\boldsymbol{x}$ ($||.||$ being the 2-norm), and this decay is controlled by the power p. In other words, p controls the smoothness of the resulting field, a lower p-value will produce a smoother field as it decreases the relative difference between each weights. The extreme being with $p = 0$ where the *Inverse* *distance* scheme is equivalent to a geometric average. A p value of 3.5 is used in this study. This is a first order scheme, and as the weights $(\omega_i / \sum_{i=1}^{N} \omega_i)$ are bounded between 0 and 1, the interpolated value will be bounded between the minimum and maximum field value of its *donor* cells.

A.4 Polynomial

Polynomial interpolation schemes generate a polynomial function that will pass through all the *donor* cells values. The evaluation of this polynomial function at the interpolation location $(\boldsymbol{x})$ is the interpolated value. A polynomial function can be expressed in the form,

$$P(\boldsymbol{x}) = P(x, y, z) = \sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x}), \tag{A5}$$

where N is the number of coefficients α_k, and F_k are the polynomial basis functions. Two different sets of basis functions are presented in the following sections. The first one named *Complete Polynomial* will simply be called *Polynomial* interpolation in the rest of the paper, and the second one is called *Polynomial tensor* interpolation.

Complete Polynomial

This type of *Polynomial* interpolation, creates a degree n polynomial function that has all the terms of degree n and below, hence the adjective *complete*. Threfore, in 2D, for a polynomial of degree n, the basis functions will be all $F = x^i y^j, i + j \leq n$ with $i, j \in \mathbb{N}$. Similarly, in 3D, $F = x^i y^j z^l, i + j + l \leq n$ with $i, j, l \in \mathbb{N}$

To find the polynomial function that passes through all of the *donor* cells, α_k that solves the system needs to be found:

$$\sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x_i}) = \phi(\boldsymbol{x_i}), \quad 1 \leq i \leq N. \tag{A6}$$

Which can be rewritten as:

$$[A_{k,m}][\alpha_k] = [rhs_k], \quad 1 \leq k, m \leq N. \tag{A7}$$

With

$$[A_{k,m}] = F_k(\boldsymbol{x_m}) \tag{A8}$$

$$[rhs_k] = \phi(\boldsymbol{x_k}) \tag{A9}$$

Once the system is solved and the α_k are obtained, the interpolated value is given by:

$$\tilde{\phi}(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k F_k(\boldsymbol{x}) \tag{A10}$$

Dimension	N (number of *donor* cells)			
	n (degree)	$n = 1$	$n = 2$	$n = 3$
2D	$\frac{(n+1)(n+2)}{2}$	3	6	10
3D	$\sum_{i=0}^{n} \frac{(i+1)(i+2)}{2}$	4	10	20

TABLE A1 Number of *donor* cells N depending on the dimension of the problem and the degree of the polynomial function n.

For a polynomial of degree n, the convergence order of its interpolation is $n + 1$. Table A1 shows the number of *donor* cells needed for the *Polynomial* interpolation.

Polynomial tensor

The *Polynomial tensor* interpolation is similar to the previously presented complete polynomial interpolation; the only difference lies in the basis functions.

The basis functions used by the *Polynomial tensor* interpolation of degree n will be, in 2D, all the terms of the product: $(\sum_{i=0}^{n} x^i)(\sum_{i=0}^{n} y^i)$, and in 3D, the terms of $(\sum_{i=0}^{n} x^i)(\sum_{i=0}^{n} y^i)(\sum_{i=0}^{n} z^i)$. In other words, the basis functions in 2D are $F = x^i y^j$

with $0 \leq i, j \leq n$ and in 3D $F = x^i y^j z^k$ with $0 \leq i, j, k \leq n$. Compared with the complete polynomial basis functions of the same degree, the *Polynomial tensor* adds some crossterms of degree higher than n.

Dimension	N (number of *donor* cells)			
	n (degree)	$n = 1$	$n = 2$	$n = 3$
2D	$(n+1)^2$	4	9	16
3D	$(n+1)^3$	8	27	64

TABLE A2 Number of *donor* cells N depending on the dimension of the problem and the degree for the *Polynomial tensor* interpolation.

In 2D and 3D the *Polynomial tensor* interpolation is sometimes called bi-linear (2D degree 1), bi-cubic (2D degree 2), or tri-linear (3D degree 1), tri-cubic (3D degree 2), etc. This kind of interpolation is often used in finite element methods. In general, to interpolate inside a square in 2D or a cube in 3D, the bi-linear or tri-linear interpolations offer the possibility to use the vertices as *cloud points*.

A.5 Least squares

The *Least squares* interpolation also relies on complete polynomial functions, where more *donor* cells than the number of unknown coefficients α_k are collected. The system is solved by minimising the sum of the squares of the errors at each *donor* cells location. By increasing the number of *donor* points, the method become more robust.

The polynomial function to be found is then:

$$P(\boldsymbol{x}) = \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x}). \tag{A11}$$

The *Least squares* approach is to find α_k that minimise:

$$S = \sum_{i=1}^{N_{cloud}} \left(\phi(\boldsymbol{x_i}) - \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x_i}) \right)^2. \tag{A12}$$

The minimisation of S is computed by finding the roots of its derivative regarding each α_l

$$\frac{\partial S}{\partial \alpha_l} = -2 \sum_{i=1}^{N_{cloud}} F_l(\boldsymbol{x_i}) \left(\phi(\boldsymbol{x_i}) - \sum_{k=1}^{N_{terms}} \alpha_k F_k(\boldsymbol{x_i}) \right) = 0 \tag{A13}$$

Which is equivalent to the $N_{terms} \times N_{terms}$ system:

$$[A_{l,m}][\alpha_l] = [rhs_l], \quad 1 \leq l, m \leq N_{terms} \tag{A14}$$

With:

$$A_{l,m} = \sum_{i=1}^{N_{cloud}} F_l(\boldsymbol{x_i}) F_m(\boldsymbol{x_i}) \tag{A15}$$

$$rhs_l = \sum_{i=1}^{N_{cloud}} F_l(\boldsymbol{x_i}) \phi(\boldsymbol{x_i}) \tag{A16}$$

A.6 Barycentric

The *Barycentric* interpolation goal is to assign to each cloud point a weight that would make the interpolation location the centre of gravity of the system and then use these weights as interpolation weights. However, such weights can be found only if the interpolation location is inside the convex polyhedron made by all the cloud points. In order to make sure the interpolation location is inside this polyhedron, cloud points are not cell centres but cell vertices.

First, the cell that contains the interpolation location is found. This cell is subdivided into tetrahedrons formed by the cell centre and one triangulated face and the tetrahedron sub cell that contains the interpolation location is selected. Barycentric weights are computed for the vertices of this sub cell. The *Barycentric* interpolation finds weights w_i associated with the vertices v_i that will lead to the interpolation location $\boldsymbol{x}$ being the barycentre or centre of mass of the system. With $\bar{\phi}_{vert_i}$ the value at one vertex of the sub-cell one can compute the value at the interpolation location with:

$$\tilde{\phi}(\boldsymbol{x}) = \frac{\sum_{i=1}^{N} w_i \bar{\phi}_{vert_i}}{\sum_{i=1}^{N} w_i} \tag{A17}$$

In order to get the values at the vertices of the sub-cell 4 different methods can be used. First, the values at the vertices of the cell can be obtained in two different ways:

$$\bar{\phi}_{vert} = \frac{1}{N_{adj}} \sum_{j=1}^{N_{adj}} \phi(\boldsymbol{x}_j), \quad (type\ 1\ and\ 3) \tag{A18}$$

$$\bar{\phi}_{vert} = \frac{1}{N_{adj}} \sum_{j=1}^{N_{adj}} \phi(\boldsymbol{x}_j) + \nabla\phi(\boldsymbol{x}_j) \cdot (\boldsymbol{x}_j - \boldsymbol{x}_{vert}), \quad (type\ 2\ and\ 4) \tag{A19}$$

With N_{adj} the number of cells adjacent to the vertex and $\boldsymbol{x}_j$ the cell centre location of an adjacent cell.

The value at the cell centre (which is the last vertex of the sub-cell) is already known, and can be used directly (type 1 and 2), or it can be reconstructed by taking the arithmetic average of the cell vertex values (type 3 and 4). Table A3 summarises the different *Barycentric* interpolation types.

		Face vertices	
		cell centre only	cell centre + gradient
Cell centre	cell centre	type 1	type 2
	reconstructed	type 3	type 4

TABLE A3 Summary of the different ways to get the values at the vertices of the sub-cell.

B RECIRCULATION BUBBLE MANUFACTURED SOLUTION EQUATIONS

The domain is an empty box, with a wall at the bottom ($y = 0$) and the flow going in the x direction. The inlet is at $x = 0.1$ and outlet at $x = 1$. The domain expands in the z direction from $z = 0$ to $z = 1$. Each quantity is composed of two parts, a *base* flow, and a *perturbation* flow. The *base* flow defines a typical turbulent boundary layer flow and stays constant over time and throughout the domain in the z direction while the *perturbation* flow represents the recirculation bubble, and evolves in time and in the z-direction.

f_{time} defines the time variation of each quantity. As presented in Equation B20, it is a sine wave oscillating between 0.2 and 1 with a period T.

$$f_{time}(t) = 0.2 + 0.4\left(1 + sin\left(\pi\left(\frac{2t}{T} - 0.5\right)\right)\right). \tag{B20}$$

The *base* flow for the x component of the velocity is defined in u_x^b (Equation B21) and represents a boundary layer flow. The shape of the recirculation bubble itself is defined using three parameters, a_1, a_2 and a_3 describing respectively the magnitude of the bubble, the location of centre of the bubble ($x = 0.5$, $y = 1/a_2$) and finally the decay of the bubble with distance to $x = 0.5$. u_x^p (Equation B22) defines this 2D recirculation bubble.

$$u_x^b(x, y) = \sum_{i=1}^{3} \alpha_i^u \cdot tanh\left(a_i^u y x^{-b_i^u} Re^{1-b_i^u}\right) \tag{B21}$$

$$u_x^p(x, y) = \left(1 - tanh\left(a_3\left(x^2 - x + 0.25\right)\right)\right) a_1 y e^{-a_2 y} \tag{B22}$$

The x component of the velocity u_x is defined in Equation B23. u_z is defined in Equation B24. And finally, u_y (Equation B25) is defined from u_x and u_z to satisfy the continuity equation.

$$u_x(x,y,z,t) = u_x^b(x,y) + u_x^p(x,y) \cdot sin^2(\pi z) \cdot f_{time}(t) \tag{B23}$$

$$u_z(x,y,z,t) = \frac{\partial u_x^p}{\partial x} \cdot \frac{sin^2(2\pi z)}{4\pi} \cdot f_{time}(t) \tag{B24}$$

$$u_y(x,y,z,t) = -\int_0^y \frac{\partial u_x}{\partial x} + \frac{\partial u_z}{\partial z} dy \tag{B25}$$

The pressure field is defined in order to have a zero gradient normal to each domain boundary, and a pressure of 0 at the outlet ($x = x_{max}$). Equation B28 defines C_p, similarly to u_x, a *base* flow is added to a fluctuating in time and in the z direction component.

$$P_x(x) = x\left(x\left(\frac{x}{3} - \frac{x_{min}+x_{max}}{2}\right) + x_{min}x_{max}\right) + 1 + \frac{x_{max}^3}{6} - \frac{x_{min}x_{max}^2}{2} \tag{B26}$$

$$P_y(y) = y^2\left(\frac{y}{3} - \frac{y_{max}}{2}\right) + 1 + \frac{y_{max}^3}{6} \tag{B27}$$

$$Cp(x,y,z,t) = P \cdot log\left(P_x\right) \cdot log\left(P_y\right) + P_b \cdot cos\left(\frac{3\pi}{2} \cdot \frac{x - x_{min}}{x_{max} - x_{min}}\right)^2 \cdot cos^2\left(\frac{\pi}{2} \cdot \frac{y}{y_{max}}\right) \cdot sin^2(\pi z) \cdot f_{time}(t) \tag{B28}$$

Finally, the eddy viscosity is defined to have a near wall behaviour close to the wall ($y = 0$) and decays exponentially in the outer region as defined in Equation B31.

$$p_{tm}^p = \left(1 + \left(a_1^d + a_2^d \cdot tanh\left(a_3^d\left(x^2 - x + 0.25\right)\right) - 1\right)\right) \frac{sin^2(\pi \cdot z)}{x^{0.8}} f_{time}(t) \tag{B29}$$

$$y^+(x,y,z,t) = \sqrt{Re \cdot \left.\frac{\partial u_x}{\partial y}\right|_{y=0}} \cdot y \tag{B30}$$

$$\tilde{\nu}_t(x,y,z,t) = \left(\left(\kappa y^+ - \tilde{\nu}_{out}\right) + \tilde{\nu}_{out}\right) \frac{e^{-y \cdot p_{tm}^p \cdot Re^{0.2}}}{Re} \tag{B31}$$

The different variables being used in the current study are shown in Table B4, they follow *case A* from[44].

Variable	Value
T	5
Re	10^7
P	500
P_b	0.25
x_{min}	0.1
x_{max}	1
y_{min}	0
y_{max}	0.4
$\tilde{\nu}_{out}$	1
κ	0.41

i	1	2	3
a_i	-140	40	16
α_i^u	0.35	0.4	0.25
a_i^u	0.0792	0.000063	0.005
b_i^u	0.2	0.2	0.2
a_i^d	0.4	0.6	10

TABLE B4 Variables being used in the recirculation bubble manufactured solution (*case A* from[44])