UNIVERSITY OF RIJEKA
FACULTY OF ENGINEERING

Ante Sikirica

ADAPTIVE MESH REFINEMENT FOR COMPUTATIONALLY EFFICIENT LARGE EDDY SIMULATIONS

DOCTORAL THESIS

Rijeka, 2025.

UNIVERSITY OF RIJEKA
FACULTY OF ENGINEERING

Ante Sikirica

ADAPTIVE MESH REFINEMENT FOR COMPUTATIONALLY EFFICIENT LARGE EDDY SIMULATIONS

DOCTORAL THESIS

Supervisor: Prof. Lado Kranjčević, PhD

Rijeka, 2025.

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET

Ante Sikirica

ADAPTIVNO UPRAVLJANJE NUMERIČKIM MREŽAMA ZA RAČUNALNO UČINKOVITE SIMULACIJE VELIKIH VRTLOGA

DOKTORSKI RAD

Rijeka, 2025.

Doctoral thesis supervisor: Prof. Lado Kranjčević, PhD

The doctoral thesis was defended on _____ at the University of Rijeka, Faculty of Engineering, Croatia, in front of the following Evaluation Committee:

1. Prof. Jerko Škifić, PhD, University of Rijeka, Faculty of Engineering
2. Prof. Siniša Družeta, PhD, University of Rijeka, Faculty of Engineering
3. Assoc. Prof. Severino Krizmanić, PhD, University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture

ABSTRACT

Computational fluid dynamics simulations often involve a trade-off between accuracy and computational cost, particularly when modelling complex flows with localised phenomena across varying scales. This trade-off becomes especially pronounced in high-fidelity simulations, which require fine grid resolution to capture intricate flow structures. Maintaining such resolution across the entire domain can, however, be computationally prohibitive. Adaptive mesh refinement is a promising methodological alternative that dynamically refines the grid in regions requiring higher accuracy and coarsens it in others. Still, its application for large eddy simulation is challenging since changes in grid resolution can induce instabilities, affect numerical dissipation, and ultimately compromise accuracy.

This thesis aims to address these limitations by developing and validating a computationally efficient adaptive mesh refinement strategy tailored for large eddy simulation within the OpenFOAM 10 framework. The central hypothesis is that an effective and physically grounded refinement criterion can be defined to guide mesh adaptation efficiently. To implement the envisioned strategy, several code contributions have been made. First, native mesh refinement capabilities were extended to handle two-dimensional problems. Second, new classes were developed for two-dimensional and three-dimensional adaptive mesh refinement, enabling refinement decisions based on logical combinations of multiple criteria, including scalar fields and geometric constraints. Third, to address parallel performance concerns, two new load balancing classes were implemented. These distributors rely on direct measurements to assess load imbalance and trigger mesh redistribution.

A core contribution is the formulation of a composite refinement criterion tailored specifically for large eddy simulations. The criterion combines the vortex identification method with a measure of local mesh resolution relative to the turbulent Taylor microscale. This ensures that coherent vortical structures are captured and the mesh is sufficiently fine to resolve the relevant turbulent scales locally.

Validation was performed systematically on a selected set of test cases. These confirmed that the implementation functions as intended and demonstrated the potential of multi-criteria

adaptive mesh refinement. The composite refinement criterion was applied to benchmark large eddy simulation cases, yielding comparable or superior accuracy relative to simulations using a conventional grid generation approach. The load balancing algorithms were also evaluated, demonstrating improved parallel efficiency and reduced simulation times.

The results suggest that the proposed approach effectively balances accuracy and computational cost for large eddy simulations. By integrating targeted refinement criteria with performance-aware load balancing, the adaptive mesh refinement strategy is able to provide high-fidelity numerical results with significantly lower computational demands, making it a practical solution for complex research and industrial problems.

Keywords: Adaptive Mesh Refinement, Load Balancing, Large Eddy Simulation, Computational Fluid Dynamics, OpenFOAM

PROŠIRENI SAŽETAK

Simulacije u domeni računalne dinamike fluida često uključuju kompromis između točnosti i računalnog troška, osobito prilikom modeliranja složenih strujanja s lokaliziranim pojavama različitih skala. Taj kompromis postaje posebno izražen u simulacijama koje zahtijevaju finu razlučivost numeričke mreže, kako bi se obuhvatile složene strukture strujanja. Ipak, održavanje takve razlučivosti na razini cjelokupne domene može biti računalno preskupo. Adaptivno upravljanje predstavlja obećavajuću metodološku alternativu koja dinamički modificira numeričke mreže u područjima koja zahtijevaju veću točnost, dok ih u drugim područjima razrjeđuje. Međutim, primjena ove metode u simulacijama velikih vrtloga predstavlja izazov jer promjene u razlučivosti mreže mogu izazvati nestabilnosti, utjecati na numeričku disipaciju i na kraju utjecati na točnost.

Ova disertacija ima za cilj ponuditi rješenje za navedena ograničenja razvijanjem i validacijom računalno učinkovite strategije adaptivnog upravljanja, prilagođene simulacijama velikih vrtloga unutar OpenFOAM 10. Središnja hipoteza jest da se može definirati učinkovit i fizikalno utemeljen kriterij za upravljanje mrežom, koji će učinkovito usmjeravati adaptaciju mreže. Za provedbu zamišljene strategije ostvareni su brojni programski doprinosi. Prvo, izvorne mogućnosti proširene su kako bi podržale dvodimenzionalne probleme. Drugo, razvijene su nove klase za dvodimenzionalno i trodimenzionalno adaptivno upravljanje, omogućujući donošenje odluka o upravljanju temeljenih na logičkim kombinacijama više kriterija, poput skalarnih polja i geometrijskih ograničenja. Treće, kako bi se riješio problem učinkovitosti pri paralelnom izvođenju, implementirane su dvije nove klase za uravnoteženje opterećenja. Spomenute klase oslanjaju se na izravna mjerenja resursa za procjenu neuravnoteženosti i iniciranje redistribucije mreže.

Ključni doprinos je formulacija kompozitnog kriterija upravljanja posebno prilagođenog za simulacije velikih vrtloga. Kriterij kombinira metodu identifikacije vrtloga s mjerom lokalne razlučivosti mreže u odnosu na turbulentnu Taylorovu mikroskalu. Na taj se način osigurava da su koherentne vrtložne strukture obuhvaćene te da je mreža dovoljno fina za lokalno razlučivanje relevantnih turbulentnih skala.

Validacija je provedena sustavno na odabranom skupu testnih slučajeva. Na temelju dobivenih rezultata utvrđeno je da implementacija djeluje prema očekivanjima te je potvrđen potencijal kompozitnog kriterija za upravljanje numeričkim mrežama. Kompozitni kriterij primijenjen je na referentne slučajeve simulacija velikih vrtloga, pri čemu su ostvareni rezultati pokazali usporedivu ili veću točnost u odnosu na simulacije koje koriste konvencionalni pristup generiranju mreže. Algoritmi za uravnoteženje opterećenja također su evaluirani, pri čemu je zabilježena poboljšana paralelna učinkovitost i smanjenje ukupnih proračunskih vremena.

Ostvareni rezultati sugeriraju da je moguće definirati kompozitni kriterij koji, u kombinaciji s algoritmima za uravnoteženje opterećenja, omogućuje postizanje numeričkih rezultata visoke točnosti uz znatno niže računalne zahtjeve, čime se potvrđuje praktična primjenjivost predložene metodologije u složenim istraživačkim i industrijskim problemima.

Ključne riječi: adaptivno upravljanje numeričkim mrežama, distribucija opterećenja, simulacije velikih vrtloga, računalna dinamika fluida, OpenFOAM

CONTENTS

1 INTRODUCTION

Achieving accurate and efficient computational fluid dynamics (CFD) simulations largely depends on the appropriate discretisation of the computational domain. The traditional discretisation approach is to utilise a uniformly refined computational grid. While straightforward, when high resolution is required, this can become computationally prohibitive, i.e., results in excessive computational costs and memory usage. An alternative, zonally refined approach introduces refinement in specific regions identified beforehand. While more efficient, this static approach still requires a priori knowledge of where high resolution is needed.

Adaptive mesh refinement (AMR) is an effective solution to these challenges. The fundamental principle of adaptive mesh refinement is to dynamically refine the computational grid in regions demanding higher accuracy, typically identified by significant solution gradients or errors, and to coarsen it in areas where the solution is smooth or the error is low [44, 47].

While the theoretical benefits of AMR are well-established, its practical application, particularly in complex simulations such as large eddy simulation (LES), presents significant challenges. LES seeks to resolve the large, energy-containing turbulent eddies directly, making it highly sensitive to grid resolution and numerical dissipation. To apply AMR effectively, it is crucial to establish refinement criteria that can accurately capture and track relevant turbulent structures without introducing excessive numerical dissipation or instability. Additionally, existing AMR implementations, such as that in the widely used open-source toolkit OpenFOAM [29], typically face limitations in terms of functionality and dynamic load balancing.

1.1 Theoretical Foundations and State of the Art

Adaptive mesh refinement is widely used in computational sciences, particularly for problems exhibiting localised phenomena with varying length and time scales [88]. These problems are common in fields such as astrophysics [77], climate modelling [30], and computational fluid dynamics [13, 44]. Over the years, AMR has evolved from a mere tool for enhancing computational efficiency and accuracy to a necessity in increasingly complex simulations [82, 106].

1.1.1 The Importance of Adaptive Mesh Refinement

Numerical solutions to partial differential equations (PDEs) that describe physical phenomena are computed over a problem domain using a finite set of points, elements, or volumes. Established methods for this purpose differ in how they approximate derivatives, enforce conservation laws, and handle boundary conditions, making each suitable for different classes of problems. These methods include the finite difference method (FDM), the finite element method (FEM), and the finite volume method (FVM) [25, 27, 44]:

- **FDM** approximates derivatives in the governing PDEs using function values at discrete grid points, typically derived from Taylor series expansions [25, 27]. It is relatively simple to implement on structured grids due to its straightforward stencil-based formulation. However, it can be less suitable for complex geometries and does not inherently guarantee the conservation of physical quantities [27].

- **FEM** discretises the problem domain by subdividing it into a mesh of finite elements [27, 114]. The solution is approximated within each element using basis or shape functions [84, 114]. FEM is based on the weak formulation of the governing equations and employs a weighted residual approach, where integration over each element yields a system of algebraic equations [27]. The accuracy and stability of the solution are strongly influenced by the quality and resolution of the mesh [31]. It is widely used in structural mechanics, heat transfer, and electromagnetics.

- **FVM** is based on the integral form of the conservation equations applied to discrete control volumes in the computational domain [25, 27]. By balancing fluxes across the faces of each control volume, FVM ensures local and global conservation of physical quantities [27]. Although robust and conservative, the accuracy depends on flux approximation schemes. FVM can handle complex geometries and is widely used in fluid dynamics [27].

The effectiveness of the aforementioned methods is heavily influenced by the computational grid. A common discretisation approach is to use a uniform grid, where grid point spacing or element size remains constant across the domain. For high resolution grids, this implies substantial computational cost. On the other hand, a coarse uniform grid might be computationally efficient but can fail to capture the essential physics and, therefore, lead to unreliable results.

This inherent inefficiency led to the development of adaptive mesh refinement. AMR encompasses a range of techniques designed to adjust the computational grid dynamically. This

adaptive approach improves accuracy while maintaining efficiency [84]. The implementation of AMR varies depending on the underlying numerical method:

- **FDM** variant adjusts grid point density. This implies dynamic adjustment of stencil structures and consistent interpolation across regions with differing resolutions.
- **FEM** variant modifies the grid primarily via element subdivision or adjustment to the polynomial degree of the basis functions.
- **FVM** variant implementation often depends on the grid. The process typically involves subdivisions and merging of control volumes to adapt grid resolution.

1.1.2 Early Developments

Berger and Oliger [14] are often mentioned as central figures in the development of local AMR [90, 102]. Their landmark 1984 paper introduced a foundational AMR concept for hyperbolic PDEs. The proposed approach utilised dynamically generated subgrids within user-defined regions of interest, which were overlaid on top of the initial, coarser computational grid. Subgrids were designed to be independent, allowing for independent numerical integration. Building upon this initial work, Berger and Colella [13] presented an automated local AMR strategy specifically focused on shock hydrodynamics. This method similarly used nested refined subgrids aligned with the underlying coarse grid. Furthermore, the authors elaborated on the grid generation process and error estimation procedure. Blayo and Debreu [15] explored the applicability of the concept presented in [14] for numerical ocean circulation models that use finite difference discretisation. Their findings suggested that AMR substantially reduces CPU time, with reported speedups of approximately threefold, while simultaneously preserving the essential statistical characteristics of the numerical solution [15]. Compared to traditional techniques for local prediction, the authors noted that the adaptive approach delivered comparable or even superior results for the same computational effort. Expanding on the algorithmic foundations laid by Berger and Colella [13], Rendleman et al. [89] presented a parallelisation strategy for structured AMR algorithms. The proposed parallelisation approach utilised dynamic load balancing to distribute the computational workload across multiple processors, relying on message-passing for inter-processor communication.

The concept of grid adaptation, however, has roots in earlier foundational research. Notable contributions include the works of Babuška [8] and Babuška and Rheinboldt [7, 9], who explored the theory and implementation of adaptive approaches within the context of FEM during the 1970s. Gago et al. [35] presented an adaptive refinement strategy based on a posteriori error measurements, which governs the refinement of a finite element grid. A notable contribution to the field was the paper by Quirk and Hanebutte [88]. The paper outlines the structure and implementation of the proposed parallel AMR approach. The authors noted that the block-structured nature of AMR algorithms inherently lends itself to parallelisation, thereby allowing for significant computational efficiency gains. The discussed adaptive grid algorithm for computational shock hydrodynamics has been introduced in Quirk's PhD thesis [87] and is inspired by [13]. Jones and Plassmann [47] introduced a parallel AMR algorithm for unstructured triangular finite element grids, which can be generalised to three-dimensional space. Similarly, Traxler [100] presented a fast algorithm for local AMR in n dimensions, which uses simplex bisection to refine tetrahedral grids. Prakash [84] focused on the development of an integrated AMR tool tailored specifically for finite element flow modelling within three-dimensional geometries of arbitrary complexity. The thesis details the design and implementation of a hierarchical data structure and a cost-effective error estimator.

Arney [6] introduced an adaptive finite volume procedure for two-dimensional Euler equations on grids using quadrilateral cells. The approach combines mesh movement and localised mesh refinement based on solution error. Kallinderis and Vijayan [51] proposed an adaptive algorithm for tetrahedral finite volume grids. The algorithm employs a dynamic cell division process to adapt the grid based on the evolution of the solution. Notably, the authors introduced two distinct refinement approaches to address the issue of hanging nodes that inevitably occur during cell subdivision: centroidal node division and directional division. The former involves introducing a new node at the cell's centroid, followed by its subsequent subdivision. The latter subdivides the cells in question into either two or four smaller cells, depending on whether multiple hanging nodes exist on a single face or a single hanging node exists on an edge. In a subsequent paper, Kallinderis [49] expanded upon earlier work by extending the proposed algorithm to handle hybrid grids composed of tetrahedra and prisms. The adaptation process for these hybrid grids is dual: prismatic cells undergo directional refinement, with tetrahedral cells treated analogously to those in [51]. Müller and Giles [72] proposed a finite volume AMR technique for triangular grids based on adjoint error analysis. In their approach, grid refinement

was driven by an estimation of the residual error, weighted by adjoint variables [72].

Vilsmeier and Hänel [105] investigated adaptive methods on unstructured grids with a focus on two-dimensional Euler and Navier-Stokes equations using the FVM. The authors noted that adaptive methods were still immature for the Navier-Stokes equations. A study by Muzaferija and Gosman [73] presented a solution-adaptive local AMR strategy, a novel spatial discretisation approach and an error estimation technique based on a Taylor series expansion. In his thesis, Ochs [79] presented an adaptive quadtree refinement method for incompressible Navier-Stokes equations within the finite volume framework. Various criteria for identifying cells requiring subdivision were explored, such as total velocity difference and cell-size-weighted velocity derivatives. Jasak and Gosman [46] presented an implementation of an error-based AMR algorithm within the finite volume-based toolkit FOAM. The authors provided a detailed description of the algorithm, the criteria used to determine where adaptation is necessary, and a solution mapping procedure to transfer data between different levels of grid refinement.

Noted works established the basis for AMR development over the following decades. Subsequent studies refined and expanded upon these initial concepts. More recently, the field has seen significant development in machine learning-based refinement strategies [28, 33, 78, 80]. A brief overview of selected key milestone papers related to AMR is given in Table 1.1.

Table 1.1: Selected milestones in the development of AMR.

Reference	Notes & context
[7, 9, 35]	Strategy for adaptive mesh refinement. A posteriori error analysis.
[13, 14, 89]	Algorithmic framework for hyperbolic partial differential equations.
[6]	AMR for Euler equations with mesh movement in two-dimensional space.
[87, 88]	Parallel AMR algorithm for computational shock hydrodynamics.
[105]	AMR for two-dimensional Euler and Navier-Stokes equations on triangular grids.
[49–51, 79]	AMR strategy for three-dimensional finite volume Navier-Stokes equations.
[46, 73, 103]	Algorithm for general computational fluid dynamics.
[84, 91]	Three-dimensional AMR for flow modelling on tetrahedral grids.
[78, 80]	Deep learning-based AMR strategies.
[28, 33]	Reinforcement learning-based AMR strategies.

1.1.3 Adaptive Mesh Refinement Paradigms

AMR paradigms can be classified into structured adaptive mesh refinement (SAMR) and unstructured adaptive mesh refinement (UAMR) based on the underlying grid structure and data

hierarchy [22]. While a brief comparative overview is given in Table 1.2, the core aspects of each are detailed below:

- **SAMR** operates on grids that maintain a logically rectangular structure, even during refinement [22, 24]. It employs a hierarchy of refinement levels. The computational domain is typically covered by the coarsest level, with finer levels embedded within it [22]. This structured arrangement enables implicit and efficient storage of mesh connectivity [22]. Consequently, fast and efficient computational kernels based on finite difference and finite volume methods can be employed [22]. Key challenges in SAMR include interface management between different refinement levels [22]. Furthermore, SAMR struggles with complex geometries [22].

- **UAMR** works on grids where the local connectivity of elements can vary arbitrarily, such as the triangular or tetrahedral grids commonly used with finite element or finite volume methods [18, 22]. Individual elements are directly modified in UAMR. This approach is, therefore, better suited for complex geometries [22, 44]. The adaptation process generally involves two main stages: first, elements are marked for refinement or coarsening based on error indicators or other criteria; subsequently, the grid is modified to ensure conformity [22]. Due to the irregular connectivity, explicit mesh element relationship data storage is required [22].

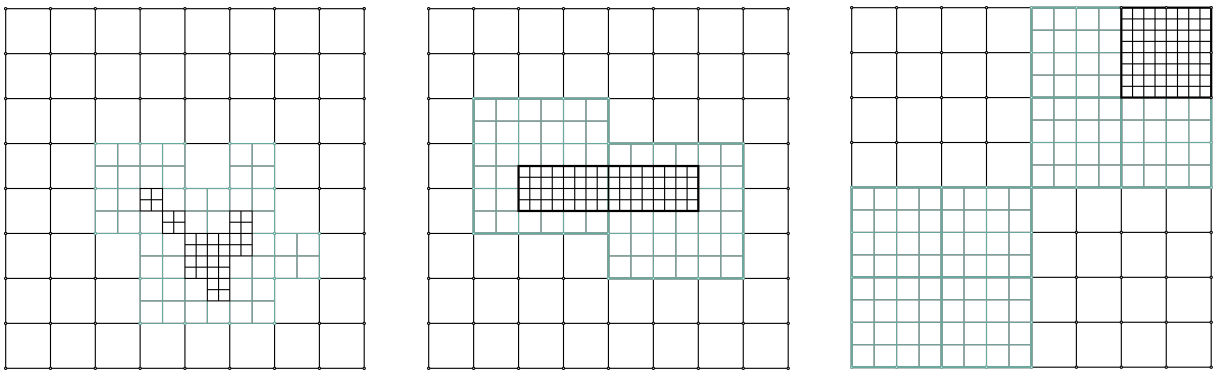
Table 1.2: Comparative overview of SAMR and UAMR [22].

Property	**SAMR**	**UAMR**
Refinement unit	Block/patch with many elements	Individual elements
Data structure	Hierarchy of grids	Element-based
Connectivity	Regular (implicit)	Irregular (explicit)
Discretisation	Usually FDM/FVM	Usually FEM/FVM
Challenges	Internal boundary handling, over-refinement, low-flexibility	Maintaining mesh quality, hanging nodes, data access
Efficiency	Use of highly efficient contiguous arrays	Less efficient due to indirect addressing

1.1.4 Methods and Data Structure

The performance and flexibility of adaptive mesh refinement are closely tied to the chosen refinement method and associated data structures. Broadly, AMR refinement methods can be classified into three main types: cell-based, patch-based, and block-based. Additionally, a fourth

hybrid block-based AMR approach exists, which combines elements of the patch-based and block-based methods. Figure 1.1 provides a brief overview of basic refinement methods.



(a) (b) (c)

Figure 1.1: Main refinement methods: (a) cell-based, (b) patch-based, (c) block-based.

The cell-based method modifies the computational grid at the individual cell or element level [86]. Grid relationships are explicitly maintained, enabling flexibility, i.e., there is no need to adhere to a rigid subdivision pattern. The grid data is organised using a hierarchical tree structure [18]. Each leaf node in the tree corresponds to a single element. This structure inherently captures parent-child relationships between refinement levels and enables efficient search and traversal operations [10, 12].

Grid modifications in the patch-based method are applied within predefined rectangular regions called patches [22, 44]. This method relies on a hierarchy of nested, uniform rectangular grids, where adaptivity is only possible within the bounds of a specific patch [22, 44]. Patch-based implementations were among the earliest AMR approaches but were unsuitable for complex geometries and unstructured grids [10]. The methodology is fundamentally rooted in the pioneering works of Berger and Oliger [14] and Berger and Colella [13].

The block-based method uses non-overlapping blocks of cells [115]. Adaptation is performed at the block level [44]. A hierarchical tree structure is employed where, unlike the cell-based approach, each leaf node represents an entire block. Refinement implies replacing a leaf block with a set of child nodes, each representing a block at the next resolution level. This method leverages the benefits of hierarchical organisational structure while maintaining computational efficiency due to the use of locally structured blocks [18]. Block-based and patch-based methods are sometimes considered equivalent or used interchangeably by some authors [108].

1.1.5 Adaptation Strategies

Grid adaptation strategies modify the grid in order to accelerate computations, improve solution accuracy, or enhance convergence. These strategies involve either changes to the mesh elements or the functions/methods used for approximation or calculation [84, 114]. Adaptation strategies are commonly classified into three main categories [46]:

- ***h*-refinement**: The grid element size h is modified through subdivision (refinement) or merging (coarsening) of elements [20, 41, 114].
- ***p*-refinement**: The polynomial order p (scheme) is modified while keeping the mesh topology unchanged [84, 114].
- ***r*-refinement**: The spatial distribution r of mesh nodes is adjusted without changing the total number of elements or connectivity [11, 59, 64, 84].

These primary strategies and various hybrid approaches offer distinct benefits and introduce specific implementation challenges [114]. Additionally, it is worth noting that the strategy classification itself can vary across the literature. For instance, Zienkiewicz et al. [114] considers r-refinement a subset of h-refinement, whereas other sources adopt stricter definitions that distinguish the methods based explicitly on changes in size, order, or position [46]. A brief graphical overview of noted approaches is given in Figure 1.2.

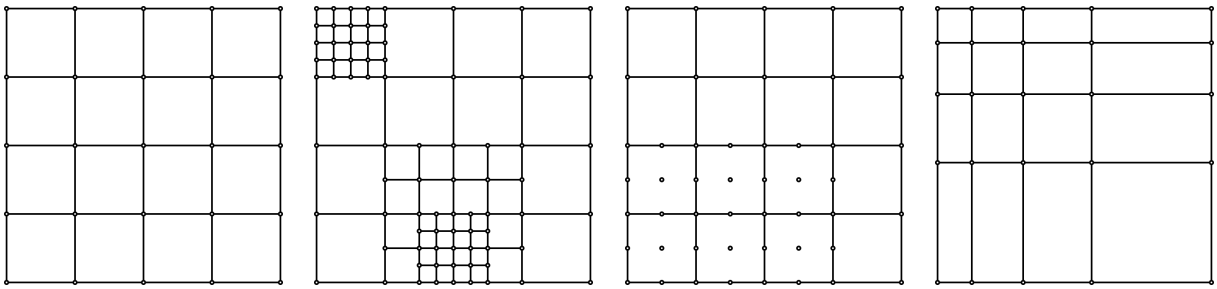


Figure 1.2: Main adaptation strategies: (a) initial grid, (b) h-refinement, (c) p-refinement, (d) r-refinement.

1.1.5.1 *h*-refinement

h-refinement (h-adaptivity) modifies the characteristic size of elements in the computational grid [114] through a series of basic geometric operations [84]. This process combines the

subdivision of existing elements into smaller ones (by introducing additional grid points) if higher resolution is needed (refinement) and the merging of smaller elements into larger ones if fine resolution is no longer required (coarsening) [114]. As a result, the grid connectivity is altered.

A key consideration for *h*-refinement is the choice between isotropic and anisotropic refinement. Isotropic refinement implies uniform subdivision of elements in all spatial directions [52]. While simpler, this can be inefficient for directionally dominated features like thin boundary or shear layers, potentially resulting in excessively fine (thin) elements and consequently leading to increased computational cost [84]. Anisotropic refinement, in contrast, allows preferential subdivision along specific directions, generating elongated elements aligned with dominant features. This can lead to significantly higher computational efficiency and accuracy [52, 84]. However, anisotropic refinement can introduce additional complexity, particularly regarding the directional criteria and grid connectivity [84].

Despite its advantages, *h*-refinement introduces several challenges. Hanging nodes are usually created where fine and coarse elements meet. These nodes require special numerical treatment (interpolation or constraint enforcement) to preserve accuracy and stability and ensure conservation [95, 114]. Furthermore, the dynamic nature of the grid can lead to computational load imbalance across processors during parallel execution. To ensure efficient load distribution, dynamic mesh repartitioning and data migration may be required, introducing additional communication overhead and algorithmic complexity [22, 95].

1.1.5.2 *p*-refinement

When employing *p*-refinement (*p*-adaptivity), the underlying computational grid remains fixed, and its connectivity is unchanged, i.e., there is no change in the grid size or the number of grid points [84]. Adaptation is achieved by selectively changing the local order of the discretisation, i.e., the degree of the numerical approximation, in regions of interest [46, 52, 84]. Consequently, *p*-refinement can lead to improved accuracy on coarser grids compared to those typically used in *h*-refinement [114].

A key advantage of *p*-refinement, particularly for problems with smooth solutions, lies in its capacity to yield exponential convergence rates as the number of degrees of freedom increases [35, 46, 84]. This is in contrast to the algebraic convergence rates typically associated with *h*-refinement [114]. From a methodological standpoint, *p*-refinement is most commonly employed

with FEM [46], where the formulation naturally allows for simple implementation of higher-order discretisation. In contrast, its application within the FVM is less straightforward due to the nature of the discretisation and the challenges associated with maintaining conservation and stability inherent to finite volume formulations.

1.1.5.3 *r*-refinement

r-refinement (*r*-adaptivity) is commonly referred to as mesh movement or redistribution. The core idea is to keep the number of grid elements and their connectivity constant while modifying the spatial distribution of grid points [6, 52, 84, 114]. Grid points are typically clustered in regions with high solution gradients or large estimated errors. Since the total number of elements remains fixed, *r*-refinement does not increase the overall resolution and may, therefore, be inadequate to achieve desired accuracy [46, 114]. Historically, it has been used primarily for two-dimensional problems [84].

A notable advantage of *r*-refinement is its inherent ability to produce directionally stretched elements. This allows for increased resolution in one direction, which is particularly important for resolving anisotropic features such as boundary layers, shear layers, or wakes. Consequently, for problems dominated by strong directional characteristics, *r*-refinement might be more efficient than *h*-refinement [84].

The main challenge associated with *r*-refinement is grid folding, wherein excessive node movement leads to degraded element quality, resulting in inverted elements with negative volumes, ultimately leading to numerical instability [114]. A fundamental limitation of *r*-refinement is that it can only redistribute existing resolution, i.e., it cannot increase the overall resolution to capture finer solution scales if the initial number of nodes is insufficient [114]. Consequently, *r*-refinement alone might not be adequate to meet stringent accuracy requirements. For this reason, *r*-refinement is often combined with *h*-refinement, and some authors have argued that it should not be regarded as a fully independent adaptation strategy [6, 114].

1.1.5.4 Hybrid Strategies

In practice, codes implementing AMR algorithms often go beyond the basic *h*-, *p*-, and *r*-refinement concepts, combining them to take advantage of their respective strengths while addressing their limitations. One of the most widely studied strategies is *hp*-refinement. This strategy modifies the characteristic element size and the polynomial order and is often guided

by a posteriori error estimation [21, 35]. It leverages the complementary nature of its components: *h*-refinement is particularly well suited for resolving singularities or regions with low solution regularity, while *p*-refinement is more efficient in regions with high solution regularity [35, 114]. Despite its theoretical appeal, *hp*-refinement has not seen widespread use in practical settings due to its implementational complexity. The simultaneous use of local *h*- and *p*-refinement introduces challenges, particularly in managing transitions between elements of different refinement levels or polynomial orders. To address these issues, alternative formulations such as the multi-level *hp*-method have been developed [113].

Another important hybrid method is *hr*-refinement, which combines *h*-adaptivity with *r*-adaptivity. This strategy benefits from the relatively low cost of point redistribution while *h*-refinement is used to overcome the connectivity constraints of pure *r*-refinement [23]. In many cases, *r*-refinement is used as the primary mechanism to improve point placement, with *h*-refinement applied selectively when mesh quality deteriorates [114].

Other, less common approaches include *rp*-refinement and the more general *hrp*-refinement. In *rp*-refinement, the grid connectivity remains fixed while the polynomial degree and point locations are adjusted [31]. This strategy is limited by the constraints of fixed grid connectivity. The most comprehensive approach, *hrp*-refinement, integrates all three adaptation types. While this provides the greatest theoretical flexibility, coordinating *h*-, *p*-, and *r*-refinements within a single framework is technically demanding and computationally expensive.

A conceptually different, broader alternative to local refinement strategies is complete remeshing [84, 114]. This process involves generating a new mesh based on the error distribution in the current solution. Some authors consider remeshing a subset of *h*-refinement [114], though its scope is generally broader. The process comes with a considerable computational cost, particularly for three-dimensional problems, and requires careful management of solution transfer between grids [84, 114]. Even so, the improved mesh quality often offsets these drawbacks; hence, remeshing is frequently used as a fallback when other strategies are ineffective.

1.1.6 Refinement Criteria

Refinement criteria are quantitative or qualitative (less common) rules that determine where and how AMR is applied within a computational domain. These criteria are essential to efficiently improve resolution in regions that contribute most significantly to numerical error or to the accuracy of specific output quantities of interest [15, 22, 44].

Historically, the most common refinement criteria were those based on a posteriori error estimation. This approach evaluates discretisation errors after obtaining a numerical solution and employs local error indicators to guide mesh adaptation [7, 9, 23, 35, 46]. A widely adopted concept is the principle of error equidistribution, which seeks to refine the grid such that the estimated local error becomes approximately uniform across the domain [12]. Residual-based estimators measure how well the numerical solution satisfies the governing equations by evaluating residuals within elements and across element interfaces [72, 74, 114]. Recovery-based estimators, such as the Zienkiewicz-Zhu estimator, quantify error based on the discrepancy between computed and smoothed gradient fields [84, 114].

While error estimation strategies aim to reduce global solution error, many engineering applications require targeted accuracy for specific quantities. Goal-oriented or adjoint-based refinement strategies are particularly effective for problems with a specific global quantity of interest. These methods rely on solving an auxiliary adjoint problem to determine how local residuals in the primary solution influence the targeted quantity [72, 74, 103].

Refinement criteria can also be based on spatial features of the solution rather than error estimation. For example, gradient-based criteria can trigger refinement in regions with large spatial gradients of flow variables or derived quantities, such as vorticity [12, 25, 95]. These criteria are relatively easy to implement and effectively capture dynamic structures like shock fronts or shear layers [12, 40]. In compressible flows, refinement is often driven by density gradients or measures of compressibility to resolve shocks and contact surfaces accurately [18]. In turbulent flows, particularly in LES, refinement can be governed by turbulent scales or guided by principles from variational multiscale theory [4, 39, 105].

Some criteria are based on estimates of local truncation error, either derived from terms inherent in the discretisation scheme or through comparisons between solutions on coarse and fine grids [14, 61]. Multi-resolution techniques exploit differences between solutions at various grid resolutions to identify regions requiring additional refinement [13, 14]. Geometric triggers, such as proximity to boundaries or specific features, can also govern refinement.

In practice, complex problems often benefit from the use of multiple refinement criteria to ensure robust and efficient grid adaptation [25, 40, 80]. These choices are highly problem-dependent and require careful consideration to balance accuracy gains with computational cost. Table 1.3 provides an overview of refinement criteria employed for high-fidelity AMR computational fluid dynamics simulations.

Table 1.3: Use of AMR for high-fidelity CFD simulations.

Software	Reference	Refinement criteria	Note
FLUSEPA	[62]	Pressure truncation error, velocity truncation error	Detached eddy simulation
Kestrel	[68]	Q-criterion	Detached eddy simulation
OpenFOAM	[5, 40]	Local error, phase fraction	Detached eddy simulation
Cerisse	[101]	Vorticity magnitude, density and pressure gradient	Large eddy simulation
OpenFOAM	[42, 58, 60, 66, 97, 110]	Gradients, scalars, various	Large eddy simulation
StarCCM+	[96]	Vorticity	Large eddy simulation
TermoFluids	[4]	Vorticity, residual velocity	Large eddy simulation
TFP-AMR	[102]	Mask inclusion method	Large eddy simulation
UMBT	[39]	Turbulent kinetic energy	Large eddy simulation
ML-based AMR (OpenFOAM)	[56]	Dominant balance analysis, Gaussian mixture model	Large eddy simulation
Basilisk	[32]	Phase fraction	Direct numerical simulation
HAMISH	[18]	Gradients of species and temperature	Direct numerical simulation

1.1.7 Impact and Current Research Trends

Adaptive mesh refinement is an important tool in fields where simulations must resolve physical processes spanning multiple orders of magnitude in space and time. In astrophysics and cosmology, AMR is indispensable for capturing complex phenomena related to galaxy formation, stellar collapse, and the evolution of large-scale cosmic structures [22, 25, 34, 82, 99]. In computational fluid dynamics, it is essential in simulations involving compressible flows and reactive fronts as well as eddy-resolving simulations [4, 18, 39, 40, 57, 66]. Additionally, it is used for high-fidelity modelling in complex real-world applications [38, 53, 107]. Beyond fluid dynamics, AMR has become instrumental in climate and Earth system modelling [12, 15, 22], combustion, plasma physics, solid mechanics, and biomedical simulations [16, 58, 82, 112].

An important avenue in current AMR research is the development of frameworks for multiphysics simulations. These frameworks often integrate Eulerian fluid solvers with arbitrary Lagrangian-Eulerian methods [22, 89]. Handling complex, moving, or embedded geometries is another active area. AMR techniques are integrated with immersed boundary and cut-cell methods to accurately capture curved, complex, or evolving interfaces without conformal meshing [109].

Machine learning methods are increasingly being leveraged to improve AMR decision-making. Techniques such as reinforcement learning are shown to be applicable to govern the refinement process [28]. Some studies rely on supervised learning or employ convolutional neural networks to predict refinement needs [78]. These data-driven methods promise improved automation and adaptability but struggle with scalability, generalisation, and seamless integration with existing codes [28, 33, 56].

Still, several fundamental challenges persist in AMR. Refinement criteria are often heuristic, requiring expert tuning or a trial-and-error approach. The use of error estimators is not universally applicable and can introduce substantial computational overhead, particularly in nonlinear or tightly coupled systems [28, 40]. Consequently, the development of robust, problem-agnostic refinement indicators remains an active and important area of research [56]. Algorithmic complexity and parallelism are another challenge. Implementing scalable AMR implies managing hanging nodes, maintaining stability across grids, and ensuring conservation across multi-level grid hierarchies on distributed systems [4, 13].

1.2 Hypothesis and Research Goals

Adaptive mesh refinement is an important tool in computational fluid dynamics. Within OpenFOAM [29, 111], a widely used open-source finite-volume toolkit, AMR capabilities exist but lack standardisation and, in some cases, basic functionality, largely due to code fragmentation across different versions and forks. Although relevant OpenFOAM studies frequently demonstrate and emphasise the efficiency benefits of AMR, they tend to focus narrowly on case-specific implementations, often neglecting broader limitations or unresolved challenges.

Employed refinement criteria are often problem- or domain-specific, as different use cases aim to resolve different physical phenomena. Criteria based purely on error estimation can be overly complex or lack direct physical interpretability for general CFD use. Furthermore, the computational overhead and processor load imbalance resulting from dynamic mesh adaptation are important and often overlooked practical hurdles.

Based on these gaps and challenges, the central hypothesis of this research can be formulated: *an effective and physically grounded adaptive mesh refinement criterion can be defined, specifically tailored for large eddy simulations and based on the size and dynamics of resolved vortices.*

To confirm this hypothesis, the following primary research goals have been defined:

- Implement a set of tools within OpenFOAM 10 [29], supporting initially refined meshes, two-dimensional cases, multi-criteria refinement, and dynamic load balancing.
- Evaluate computational performance, focusing on load balancing efficiency and parallel scalability during dynamic mesh adaptation on local and distributed systems.
- Define and validate an AMR criterion for LES, based on vortex characteristics.
- Apply the approach to problems from environmental engineering and aerodynamics, validating results against available experimental data.

1.3 Scientific Contribution

This thesis aims to define a robust and interpretable AMR criterion and address the limitations of the considered OpenFOAM implementation by developing a set of tools that seamlessly integrate with it. The goal is to create a more flexible and efficient high-performance computing environment for AMR-based simulations.

The main efforts in this work include designing and developing classes capable of handling both two-dimensional and three-dimensional problems, as well as enabling multi-criteria refinement and load balancing. The resulting toolset will be rigorously tested for performance, scalability, and efficiency and compared to conventional, non-AMR approaches, ensuring practical usability.

In addition to the software development, a key scientific contribution of this research is the assessment of the synergy between AMR and large eddy simulations. Given that LES is highly sensitive to numerical errors and dissipation and that AMR can introduce instabilities, this thesis aims to define a complex and well-structured AMR criterion capable of accurately capturing the underlying physical phenomena while avoiding numerical issues. This criterion will be presented and calibrated with careful consideration of computational cost and the frequency of iterative mesh modifications. The impact of the proposed criterion will be tested across different LES models and test scenarios. This is an important step towards making high-fidelity AMR simulations more dependable.

1.4 Thesis Structure

This thesis is organised into seven chapters with corresponding subchapters. The introductory chapter presents a literature review on adaptive mesh refinement, the main research objectives and the hypothesis.

The second chapter details the numerical modelling framework used in this thesis. It covers the governing equations, the finite volume method, and a range of turbulence models. The chapter concludes with the introduction and validation of test cases used later in the thesis.

The third chapter evaluates OpenFOAM's native adaptive mesh refinement capabilities. It starts with an overview of the existing framework, followed by the implementation of a two-dimensional refinement class. The newly implemented two-dimensional and existing three-dimensional refinement classes are then validated using the previously defined test cases.

The fourth chapter introduces a multi-criteria refinement algorithm. This algorithm allows simultaneous use of multiple independent refinement criteria, including geometric constraints, and is validated on two-dimensional and three-dimensional test cases.

The fifth chapter investigates strategies for dynamic load balancing in simulations using adaptive mesh refinement. Two approaches are proposed, and their impact on computational efficiency is assessed on a select test case.

The sixth chapter introduces a composite refinement criterion designed for large eddy simulation. Existing criteria are reviewed, and a new composite formulation is proposed. The criterion is validated on three distinct test cases introduced in Chapter 2.

The final chapter summarises the main findings of the research and reflects on the scientific contributions. The limitations of the current approach are discussed, and suggestions for future research directions are outlined.

2 NUMERICAL MODELLING AND VALIDATION CASES

The motion of fluids, from large-scale weather systems to blood flow in the human body, is governed by physical laws. These laws are described mathematically by complex, non-linear partial differential equations. For most scenarios encountered in science and engineering, finding exact solutions to these equations is often too difficult or even impossible. Consequently, the field of computational fluid dynamics emerged, centred around transforming governing differential equations into a solvable system of algebraic equations.

2.1 Governing Equations of Fluid Flow

Fluid motion is mathematically described by the fundamental physical conservation laws of mass, momentum, and energy. These conservation laws can be written as partial differential equations and form the cornerstone of fluid dynamics [3, 27].

Before deriving the governing equations, a fundamental modelling assumption must be introduced: the continuum hypothesis. This hypothesis asserts that fluids can be treated as continuous media, ignoring their discrete molecular nature [27, 92]. Physical properties such as density ρ, velocity $\mathbf{u}$, pressure p, temperature T, and viscosity μ are treated as smoothly varying field variables defined at every point within the domain. The continuum hypothesis is valid when the characteristic length scale of the flow is much larger than the mean free path of particles in the fluid [92]. For most engineering applications addressed by CFD, including those in which AMR is beneficial, this assumption holds true [27, 92].

To formulate these laws, we adopt the Eulerian perspective. Rather than following individual fluid particles, as in the Lagrangian approach, the Eulerian viewpoint focuses on fixed locations in space and observes how fluid properties change as the fluid passes through them [27]. Central to the Eulerian formulation is the concept of a control volume (CV). The control volume is a finite region of space over which the conservation laws are applied [27]. The general conservation principle for a quantity ϕ within a control volume V bounded by surface S can

be written in integral form as:

$$\frac{\partial}{\partial t}\int_V \rho\phi\, dV + \int_S \rho\phi(\mathbf{u}\cdot\mathbf{n})\, dS = \int_S \Gamma_\phi(\nabla\phi\cdot\mathbf{n})\, dS + \int_V S_\phi\, dV \tag{2.1}$$

where t is time, $\mathbf{n}$ is surface normal vector, Γ_ϕ is diffusion coefficient and S_ϕ represents a source (or a sink).

2.1.1 Conservation of Mass

Mass conservation implies that the mass inside a CV can only change if there is a net flow of mass entering or leaving the CV through its boundaries. For a fixed control volume V with boundary S, the integral form of the mass conservation equation can be derived from the expression for a control volume (Eq. 2.1) by setting the property $\phi = 1$, with diffusion coefficient $\Gamma_\phi = 0$ and no source terms $S_\phi = 0$:

$$\frac{\partial}{\partial t}\int_V \rho\, dV + \int_S \rho(\mathbf{u}\cdot\mathbf{n})\, dS = 0. \tag{2.2}$$

The choice of $\phi = 1$, $\Gamma_\phi = 0$ and $S_\phi = 0$ isolates pure convective mass transport by eliminating diffusive and source terms. The term $\rho(\mathbf{u}\cdot\mathbf{n})$ represents the mass flux through the control volume boundary. By utilising the divergence theorem on the surface integral, the differential form of the mass conservation equation can be derived:

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{u}) = 0. \tag{2.3}$$

This equation holds true for compressible and incompressible flows. In the case of incompressible flow, where the material derivative of density ρ is zero, the continuity equation simplifies to:

$$\nabla\cdot\mathbf{u} = 0. \tag{2.4}$$

The finite volume method is particularly well-suited for ensuring mass conservation in numerical simulations, as its formulation is based on applying the integral form of the conservation laws to each discrete control volume within the computational grid [27]. Mass is conserved exactly in a discrete sense, provided the fluxes are computed consistently across shared faces.

2.1.2 Conservation of Momentum

The momentum conservation law is a generalisation of Newton's second law applied to fluids [3, 27]. This law states that the rate at which a fluid particle's momentum changes is equal to the net force applied to it. Acting forces can be classified into surface forces (pressure and viscous) and body forces (e.g. gravity) [3, 27]. By defining $\phi = \mathbf{u}$ in the general control volume equation [27], analogously to mass conservation, the integral form of the momentum conservation equation can be derived:

$$\frac{d}{dt}\int_V \rho\mathbf{u}\,dV + \int_S \rho\mathbf{u}(\mathbf{u}\cdot\mathbf{n})\,dS = \int_S \sigma\cdot\mathbf{n}\,dS + \int_V \rho\mathbf{g}\,dV + \int_V \mathbf{f}\,dV \tag{2.5}$$

where σ is the stress tensor, $\mathbf{g}$ represents gravitational acceleration, and $\mathbf{f}$ represents other body forces acting per unit volume. The left side of the equation represents the rate of change and convective transport of momentum, whereas the right side includes the surface and body forces acting on the fluid. The differential form can be obtained by applying the divergence theorem and assuming sufficient smoothness. The resulting vector equation is commonly known as the Navier-Stokes equation [3]:

$$\frac{\partial(\rho\mathbf{u})}{\partial t} + \nabla\cdot(\rho\mathbf{u}\mathbf{u}) = -\nabla p + \nabla\cdot\tau + \rho\mathbf{g} + \mathbf{f} \tag{2.6}$$

where $\rho\mathbf{u}$ is the momentum density, $\rho\mathbf{u}\mathbf{u}$ represents the convective flux of momentum, p is the static pressure, τ is the viscous stress tensor, $\rho\mathbf{g}$ represents gravitational body forces, and $\mathbf{f}$ represents other body forces acting per unit volume.

For an incompressible Newtonian fluid with constant dynamic viscosity μ, the viscous stress tensor can be simplified:

$$\tau = \mu\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right)\ . \tag{2.7}$$

Finally, the Navier-Stokes equation can be reduced to:

$$\rho\left(\frac{\partial\mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g} + \mathbf{f}\,. \tag{2.8}$$

The Navier-Stokes equation is challenging to solve primarily due to the non-linear convective term $\mathbf{u}\cdot\nabla\mathbf{u}$, which implies interactions between the velocity components and is responsible for the emergence of complex flow phenomena [92].

2.1.3 Conservation of Energy

The energy conservation equation for a fluid can be derived from the first law of thermodynamics. It includes the effects of convection, heat conduction, work done by surface forces (pressure and viscous stresses), and heat addition from body forces or internal sources. By substituting $\phi = E$ in the general control volume equation, analogously to mass conservation, and including appropriate source terms for work and heat sources, the integral form of the energy conservation equation can be derived:

$$\begin{aligned}\frac{d}{dt}\int_V \rho E\, dV + \int_S \rho E\mathbf{u}\cdot\mathbf{n}\, dS = &-\int_S p\mathbf{u}\cdot\mathbf{n}\, dS + \int_S (\tau\cdot\mathbf{u})\cdot\mathbf{n}\, dS \\ &-\int_S \mathbf{q}\cdot\mathbf{n}\, dS + \int_V \rho(\mathbf{g}\cdot\mathbf{u})\, dV + \int_V \dot{Q}\, dV\end{aligned} \tag{2.9}$$

where E is the total energy per unit mass, $E = e + \frac{1}{2}|\mathbf{u}|^2$, with e being internal energy, $\mathbf{q}$ is the heat flux vector, and $\dot{Q}$ accounts for internal heat sources. By applying the divergence theorem to the surface integrals, the differential form of the energy equation can be obtained:

$$\frac{\partial(\rho E)}{\partial t} + \nabla\cdot(\rho E\mathbf{u}) = -\nabla\cdot(p\mathbf{u}) + \nabla\cdot(\tau\cdot\mathbf{u}) - \nabla\cdot\mathbf{q} + \rho(\mathbf{g}\cdot\mathbf{u}) + \dot{Q}\,. \tag{2.10}$$

The energy equation is particularly important for problems involving compressible flows, where density changes are coupled with temperature and pressure variations via an equation of state, or for flows with significant heat transfer [27]. For isothermal, incompressible flows, the energy equation can be simplified and decoupled [27].

2.2 Principles of the Finite Volume Method

Numerical discretisation implies an approximation of continuous derivatives and integrals in the governing equations at a finite number of points or over a finite number of volumes within the computational domain [27, 92]. The governing partial differential equations in computational fluid dynamics are typically discretised using the finite volume method. The widespread adoption of FVM in CFD stems from its integral-based formulation of conservation laws [27].

In the finite volume method, the computational domain is partitioned into a finite number of non-overlapping control volumes or cells [27]. The integral form of the conservation laws is applied directly to each cell. Surface integrals in the governing equations are approximated as

sums of fluxes across the faces of each cell, with fluxes evaluated using interpolated face values based on neighbouring cell centres. Volume integrals are typically approximated by assuming that the integrand is constant within each cell and using its value at the cell centre [27].

Discretisation converts the integral conservation law into an algebraic equation linking a cell to its neighbours. The resulting discrete equation for a quantity ϕ takes the general form:

$$a_P \phi_P^{n+1} = \sum_N a_N \phi_N^{n+1} + b \tag{2.11}$$

where ϕ_P^{n+1} is the value of ϕ at the centre of the current cell P, ϕ_N^{n+1} are the values at neighbouring cells N, a_P and a_N are coefficients determined by the discretisation of the fluxes and source terms and b accounts for contributions from source terms and boundary conditions.

An important characteristic of FVM is that flux conservation is enforced at the level of each face: the flux leaving one control volume through a shared face is exactly equal (but opposite in sign) to the flux entering the adjacent volume. As a result, conservation of mass, momentum, and energy is guaranteed both locally (at the cell level) and globally (across the entire computational domain) [27]. This property holds true regardless of the mesh size, making FVM robustly conservative even on coarse grids.

2.3 Turbulence Modelling

Turbulence is a complex and inherently chaotic fluid motion fundamental to natural and engineered systems. It is characterised by irregular fluctuations in velocity and pressure [27]. This regime contrasts with laminar flow, which is smooth and orderly and typically occurs at low Reynolds numbers (depending on the problem and conditions). In turbulent flow, inertial forces dominate over viscous forces [27, 92].

Turbulent flows are inherently unsteady, three-dimensional, and rotational and feature unpredictable fluctuations across a wide range of time and length scales [27]. These flows are statistically described rather than deterministically predicted, as the instantaneous state of the flow cannot be precisely determined [92]. Turbulence is characterised by swirling, vortical structures known as eddies, ranging from large scales that extract energy from the mean flow down to very small scales.

Turbulent flows are dissipative. Kinetic energy is transferred from large, energy-containing eddies to progressively smaller eddies through a process called the energy cascade [27, 83]. This energy transfer primarily occurs within the inertial subrange, where energy cascades from large to small eddies with minimal viscous dissipation at intermediate scales. The energy dissipation becomes significant at the smallest scales, the Kolmogorov microscales, where viscous forces dominate and dissipate the kinetic energy into internal energy [83, 92]. These microscales correspond to the smallest turbulent eddies in the flow. Maintaining turbulence requires a continuous energy supply to counteract dissipation [92].

The transition from a laminar to a turbulent flow regime occurs near a critical Reynolds number and is characterised by turbulence intermittently developing within the otherwise laminar flow [27]. The onset of turbulence is sensitive to initial conditions such as surface roughness, free-stream turbulence, pressure gradients, and geometric features, which can induce flow instabilities.

Given the wide range of scales involved, turbulent flows are difficult to resolve numerically. Direct numerical simulation (DNS) is the most accurate simulation approach because it resolves all turbulence scales, but it is computationally prohibitive for practical applications due to the exponential growth in computational effort. Consequently, turbulence models, which model unresolved turbulent motions, are commonly used. The most widely employed models are Reynolds-averaged Navier-Stokes (RANS) and large eddy simulation. RANS offers a computationally efficient approach by time-averaging the effects of turbulence at the cost of reduced physical fidelity, while LES resolves larger eddies and models smaller ones, providing a better balance between accuracy and computational cost [27, 92].

2.3.1 Reynolds-Averaged Navier-Stokes Equations

The core idea of the Reynolds-averaged Navier-Stokes approach is to focus on the mean behaviour of turbulent flows rather than directly resolving all the instantaneous fluctuations. To derive the RANS equations, the Reynolds decomposition must first be applied to the instantaneous Navier-Stokes equations, followed by the application of an averaging operator [27, 92]. In Reynolds decomposition, any instantaneous flow variable $\phi(\mathbf{x},t)$ is split into a mean and a fluctuating part [27, 92]:

$$\phi(\mathbf{x},t) = \overline{\phi}(\mathbf{x}) + \phi'(\mathbf{x},t) \tag{2.12}$$

where $\overline{\phi}$ is the mean (time-averaged or ensemble-averaged) quantity and ϕ' is the turbulent fluctuation about the mean. Different types of averaging are employed depending on the nature of the flow. Time averaging is used when the flow is statistically steady in time [27]:

$$\overline{\phi}(\mathbf{x}) = \lim_{T\to\infty} \frac{1}{T} \int_0^T \phi(\mathbf{x},t)\, dt \tag{2.13}$$

where T is a sufficiently long time over which fluctuations average out. Ensemble averaging is used for statistically non-stationary flows and is defined as [27]:

$$\overline{\phi}(\mathbf{x},t) = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \phi^{(n)}(\mathbf{x},t) \tag{2.14}$$

where the average is taken across many independent realizations of the flow at the same instant in time. A key property of the Reynolds operator is that the average of a fluctuating quantity equals [92]:

$$\overline{\phi'} = 0\ . \tag{2.15}$$

Assuming incompressible flow with constant fluid properties, the conservation equations reduce to:

$$\nabla \cdot \overline{\mathbf{u}} = 0, \tag{2.16}$$

$$\rho \left(\frac{\partial \overline{\mathbf{u}}}{\partial t} + (\overline{\mathbf{u}} \cdot \nabla)\overline{\mathbf{u}} \right) = -\nabla \overline{p} + \mu \nabla^2 \overline{\mathbf{u}} - \rho \nabla \cdot \overline{\mathbf{u}'\mathbf{u}'} \tag{2.17}$$

where $\overline{\mathbf{u}}$ is the mean velocity vector, $\overline{p}$ is the mean pressure, $\overline{\mathbf{u}'\mathbf{u}'}$ represents the normalised Reynolds stress and μ is the dynamic viscosity.

The Reynolds stress tensor $\tau^R = -\rho\overline{\mathbf{u}'\mathbf{u}'}$ captures the momentum transport caused by turbulent velocity fluctuations. Physically, these stresses act like additional turbulence-induced stress on the mean flow field, analogous to molecular viscous stresses, but arising from turbulent eddy motion instead [38]. The introduction of the Reynolds stress tensor results in more unknowns than available equations, creating the so-called closure problem. This prevents direct analytical solution of the RANS equations, hence turbulence models are introduced to approximate the unknown stresses and close the system.

To address the closure problem, most RANS turbulence models rely on the Boussinesq hypothesis, which draws an analogy between turbulent eddy-induced momentum transport and molecular viscosity. It assumes that the Reynolds stresses are proportional to the mean rate

of strain tensor, much like viscous stresses are in a Newtonian fluid [94]. For incompressible flows, the Boussinesq approximation can be used to define the Reynolds stress tensor:

$$-\rho\overline{\mathbf{u}'\mathbf{u}'} = 2\mu_t\overline{\mathbf{S}} - \frac{2}{3}\rho k\mathbf{I} \tag{2.18}$$

where $\overline{\mathbf{S}} = \frac{1}{2}\left(\nabla\overline{\mathbf{u}} + (\nabla\overline{\mathbf{u}})^T\right)$ is the mean strain rate tensor, $k = \frac{1}{2}\mathrm{tr}\left(\overline{\mathbf{u}'\mathbf{u}'}\right)$ is the turbulent kinetic energy, μ_t is the turbulent viscosity, and $\mathbf{I}$ is the identity tensor. Using the Boussinesq hypothesis, the RANS momentum equation can be rewritten to include the turbulent viscosity:

$$\rho\left(\frac{\partial\overline{\mathbf{u}}}{\partial t} + (\overline{\mathbf{u}}\cdot\nabla)\overline{\mathbf{u}}\right) = -\nabla\left(\overline{p} + \frac{2}{3}\rho k\right) + \nabla\cdot\left[(\mu+\mu_t)\left(\nabla\overline{\mathbf{u}} + (\nabla\overline{\mathbf{u}})^T\right)\right] . \tag{2.19}$$

The turbulent viscosity is not a physical property of the fluid but a property of the turbulent state of the flow and needs to be modelled [92]. Typical approaches involve solving additional transport equations for turbulence quantities (N-equation turbulence models).

A fundamental limitation of the Boussinesq hypothesis is the assumption that turbulent fluctuations are isotropic. This assumption often breaks down in complex flows featuring strong streamline curvature, swirl, rotation, stagnation regions, or significant separation [92]. In such cases, advanced (typically non-linear) RANS turbulence models are necessary for accurate predictions [94].

2.3.2 Large Eddy Simulation

Large eddy simulation is a turbulence modelling strategy that aims to provide higher fidelity than RANS by directly resolving a portion of the turbulent scales while modelling the remainder, balancing accuracy and computational cost [92]. The objective is to capture detailed information about turbulent flow structures and their unsteadiness without incurring the computational cost of DNS.

The underlying concept of LES is rooted in Kolmogorov's theories, which describe a wide range of eddy sizes in turbulent flows [83, 92]. The largest eddies are anisotropic, contain most of the turbulent kinetic energy and dominate momentum transport [83, 92]. In contrast, the smallest eddies are more isotropic, universal, and primarily responsible for energy dissipation [27, 92]. LES leverages this scale separation by dedicating computational resources to directly resolve the large, energy-containing eddies while modelling the influence of smaller, unresolved

scales, i.e. subgrid scales (SGS) [38, 92]. A well-resolved LES aims to capture approximately 80 % or more of the turbulent kinetic energy [83].

Mathematically, LES achieves scale separation by applying a spatial low-pass filter with a filter width Δ to the governing Navier-Stokes equations [27, 83]. This is formally noted as a convolution integral [92]:

$$\tilde{\phi}(\mathbf{x},t) = \int_V G(\mathbf{x}-\mathbf{x}',\Delta)\phi(\mathbf{x}',t)\,dV' \tag{2.20}$$

where G is the filter kernel. Filtering averages the flow field, attenuating fluctuations smaller than Δ while retaining larger structures. Any instantaneous variable ϕ can thus be decomposed into a resolved (filtered) component $\tilde{\phi}$, and a sub-filter component $\phi' = \phi - \tilde{\phi}$ [27, 83].

In many practical implementations, filtering is implicit, i.e. the computational grid acts as the filter. Here, Δ is typically related to the local grid size, and discretisation introduces additional implicit filtering effects [92]. Consequently, the quality of an LES is directly linked to the grid resolution, with finer grids resolving more turbulent scales.

The filtered Navier-Stokes equations, governing the resolved velocity $\tilde{\mathbf{u}}$ and pressure $\tilde{p}$ fields, can be derived by applying filtering to the Navier-Stokes equations. Assuming incompressible flow and the absence of external body forces, the filtered Navier-Stokes equations reduce to:

$$\nabla \cdot \tilde{\mathbf{u}} = 0, \tag{2.21}$$

$$\frac{\partial(\rho\tilde{\mathbf{u}})}{\partial t} + \nabla\cdot(\rho\widetilde{\mathbf{u}\mathbf{u}}) = -\nabla\tilde{p} + \nabla\cdot(2\mu\tilde{\mathbf{S}}) \tag{2.22}$$

where $\tilde{\mathbf{S}} = \frac{1}{2}\left(\nabla\tilde{\mathbf{u}} + (\nabla\tilde{\mathbf{u}})^T\right)$ is the resolved strain-rate tensor.

Filtering the non-linear convective term leads to a closure problem since the filtered product of velocity components is not equal to the product of filtered velocities [27, 83]. The difference between these values is the subgrid-scale stress tensor:

$$\tau^{SGS} = \rho\left(\widetilde{\mathbf{u}\mathbf{u}} - \tilde{\mathbf{u}}\,\tilde{\mathbf{u}}\right)\ . \tag{2.23}$$

Incorporating τ^{SGS} into the filtered momentum equation yields:

$$\frac{\partial(\rho\tilde{\mathbf{u}})}{\partial t} + \nabla\cdot(\rho\tilde{\mathbf{u}}\tilde{\mathbf{u}}) = -\nabla\tilde{p} + \nabla\cdot\left(2\mu\tilde{\mathbf{S}} - \tau^{SGS}\right)\ . \tag{2.24}$$

The SGS stress tensor τ^{SGS} is an additional divergence term in the filtered momentum equation [83, 92]. Physically, it models the effect of unresolved scales on the resolved flow, accounting for momentum transfer, the energy cascade, and subgrid-scale dissipation. To approximate τ^{SGS} in terms of resolved quantities, a subgrid scale model must be used.

2.4 Numerical Modelling in OpenFOAM

OpenFOAM (Open Field Operation and Manipulation) [111] is an open-source collection of C++ libraries and applications designed to solve systems of partial differential equations, with a focus on problems in continuum mechanics. The primary numerical method employed is the FVM. Although extensively used in computational fluid dynamics, OpenFOAM's flexible architecture supports a wide range of PDE-driven applications, including solid mechanics, electromagnetics, and chemical processes.

OpenFOAM is provided as free and open-source software under the GNU General Public License. This has led to the development of several major forks. The three most prominent are:

- OpenFOAM maintained by the OpenFOAM Foundation [29],
- OpenFOAM maintained by ESI-OpenCFD [26],
- foam-extend, a community-driven fork [85].

Even though these forks originated from the same codebase, their development paths have diverged. Consequently, they have different syntax variations, offer different feature sets, and are not always cross-compatible.

OpenFOAM uses a modular, object-oriented architecture with core functionality organised into libraries, which handle tasks such as mesh manipulation, discretisation, linear algebra, turbulence modelling, and input/output. On top of these core libraries sits the application layer, which is divided into solvers and utilities:

- Solvers are executable applications designed to address specific classes of physical problems (e.g., `simpleFoam` for steady-state incompressible turbulent flow, `pisoFoam` for transient incompressible flow, and `interFoam` for multiphase flows using the volume of fluid (VOF) method).

- Utilities are supplementary applications designed for preprocessing, postprocessing, and case manipulation.

A typical OpenFOAM case follows a specific directory structure with three primary subdirectories (this can vary significantly):

- **constant** contains data that typically remains unchanged throughout the simulation:
  - Mesh data stored in the `polyMesh` subdirectory.
  - Dictionaries that define material properties, such as `physicalProperties`, `momentumTransport`, and `thermophysicalProperties`.
  - Dictionaries that control specific simulation aspects, such as `dynamicMeshDict` for dynamic mesh operations.
- **system** contains configuration files that control the numerical and algorithmic execution of the simulation. Essential files include:
  - `controlDict` dictionary, which governs the overall simulation execution.
  - `fvSchemes` dictionary, which defines the numerical discretisation schemes for different terms in the governing equations.
  - `fvSolution` dictionary, which specifies the algorithms used to solve the discretised equations, linear solvers, preconditioners, tolerances, and relaxation factors.
  - Additional dictionaries such as those for parallel execution, function objects, or other specialised settings.
- **time directories** named according to the simulation specifics. These directories store the field data (physical quantities). Initial and boundary conditions for all relevant fields are specified in the starting time directory (typically **0**).

A standard case workflow typically begins with the definition of the computational mesh, either by using built-in utilities such as `blockMesh` or `snappyHexMesh` or by importing a mesh from an external source. Physical properties and case-specifics are subsequently specified in the **constant** directory. The initial conditions, including relevant boundary conditions and field values, are set in the **0** directory. Finally, numerical settings, solver controls, and runtime parameters are configured in the **system** directory. Once the setup is complete, the appropriate

solver (application) is executed. Simple problems are typically solved in serial mode. For parallel execution, the mesh must first be decomposed, after which the solver is run using the Message Passing Interface (MPI) wrapper. During runtime, the solver outputs field data (and other results) to time-stamped directories that track the simulation's evolution. In parallel runs, reconstruction is needed for postprocessing.

2.5 Validation of Laminar and RANS Benchmark Cases

This section introduces a set of benchmark problems solved using either a laminar flow assumption or RANS turbulence models. Three well-established test cases are considered: flow around a circular cylinder, bubble dynamics in a rising gas-liquid system, and a dam-break problem. Each case is simulated using conventional (non-adaptive) grid generation on three mesh resolutions. The simulation setups are described in detail, and the results are compared against experimental or high-fidelity numerical reference data.

2.5.1 Flow Around a Cylinder

Two distinct test cases of flow around a cylinder are considered: a two-dimensional test case and a three-dimensional test case. Both cases are based on the benchmark configuration proposed by Schäfer and Turek [93]. The working fluid is incompressible, with constant density $\rho = 1$ kg/m^3 and kinematic viscosity $\nu = 1 \cdot 10^{-3}$ m^2/s. Due to the low Reynolds number, the flow is laminar.

Two-Dimensional Case

The two-dimensional test case is defined in a rectangular domain 2.2 m $\times$ 0.41 m in size. The top and bottom boundaries of the domain are treated as no-slip walls. A stationary cylinder is located 0.15 m downstream from the inlet and 0.15 m above the bottom wall. The inflow velocity varies in time and follows the parabolic profile defined in benchmark case 2D-3 [93], with a mean velocity $U_m = 1.5$ m/s:

$$u(y,t) = \frac{4U_m y(H-y)}{H^2} \sin\left(\frac{\pi t}{8}\right) \tag{2.25}$$

where u is the streamwise velocity component, $H = 0.41$ m is the domain height, y is the vertical coordinate, and t is time. The simulation is inherently unsteady and is run over a period of 8 s.

The Reynolds number varies in time, spanning the range $0 \leq Re \leq 100$. Figure 2.1 provides an overview of the computational domain.

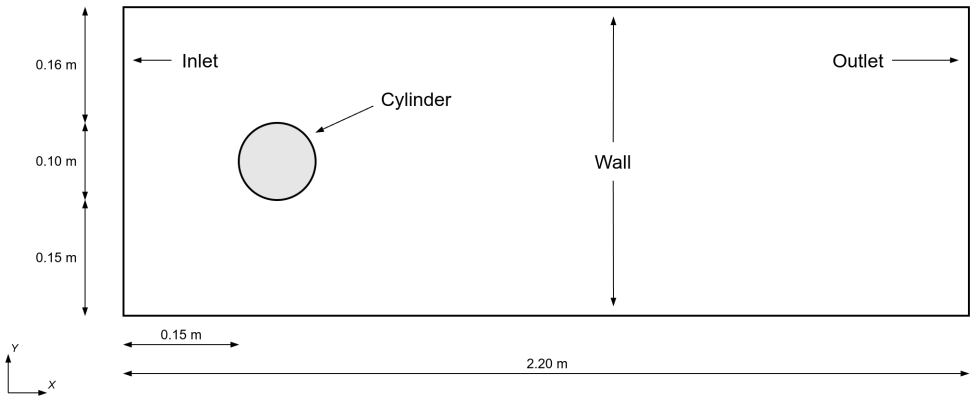


Figure 2.1: Computational domain for two-dimensional flow around a cylinder.

The simulation is conducted using the unsteady, incompressible solver `pimpleFoam`. An adaptive time-stepping strategy is employed to maintain a maximum Courant number of $Co_{\max} = 0.05$. This ensures that the local time step is automatically adjusted according to the local mesh resolution and flow velocity, thereby satisfying the Courant-Friedrichs-Lewy (CFL) stability criterion:

$$\text{CFL} = \frac{u_x \Delta t}{\Delta x} + \frac{u_y \Delta t}{\Delta y} + \frac{u_z \Delta t}{\Delta z} \tag{2.26}$$

where u_x, u_y, and u_z are the velocity components in the x-, y-, and z-directions, respectively, and Δx, Δy, and Δz are the corresponding local grid spacings.

Second-order accurate numerical schemes are employed throughout. Time integration is performed using an implicit, unbounded, second-order accurate scheme. Spatial derivatives, including the gradient, Laplacian, and surface-normal gradient, are discretised using second-order accurate schemes, with corrections applied for the Laplacian and surface-normal gradient. For the convective terms, `limitedLinear` scheme is used.

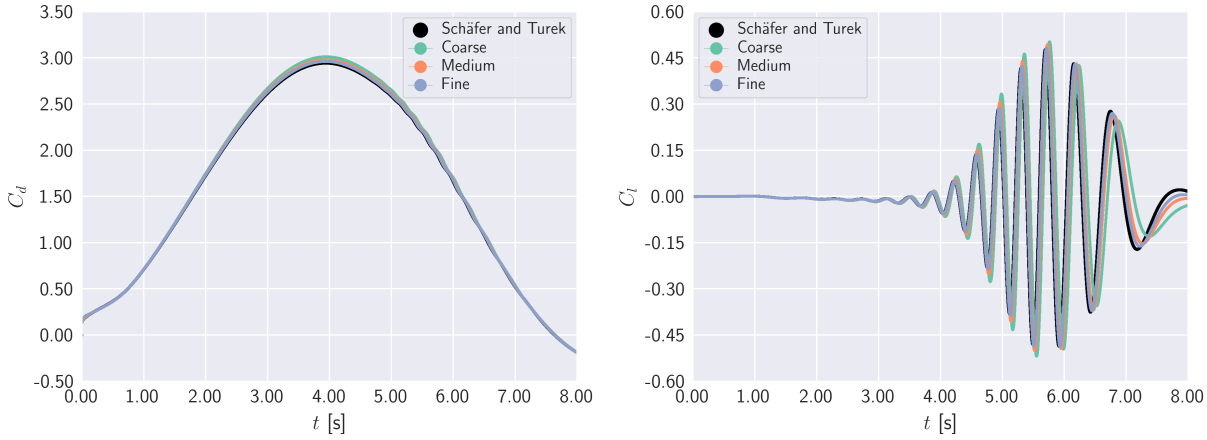
The forces acting on the cylinder are continuously monitored. Based on calculated forces, the non-dimensional drag and lift coefficients, C_D and C_L, are derived to quantify the aerodynamic behaviour of the cylinder under unsteady flow conditions:

$$C_D = \frac{2F_w}{\rho \overline{U}^2 DH}, \quad C_L = \frac{2F_a}{\rho \overline{U}^2 DH} \tag{2.27}$$

where F_w and F_a are the force components in the x- and y-directions, respectively, ρ is the fluid density, $\overline{U}$ is the mean velocity, and D is the diameter of the cylinder.

Two-Dimensional Flow Results

Three simulations were conducted for the two-dimensional case using progressively finer grids, with $r = 1.5$ scaling factor between each level. Drag and lift coefficient results are shown in Figure 2.2. Several key observations can be made.



(a) (b)

Figure 2.2: Resulting drag and lift coefficients for the two-dimensional cylinder case: (a) drag coefficient, (b) lift coefficient.

First, concerning the drag coefficient, C_D, the overall trend is well captured by all grids throughout the simulation. Minor differences appear around $t \approx 4$ s, where the coarser grids tend to overpredict drag. As the mesh is refined, the results, particularly on the fine grid, converge towards the reference data of Schäfer and Turek [93].

In contrast, the lift coefficient, C_L, is more sensitive to grid resolution. Due to its lower magnitude and oscillatory behaviour, discrepancies are more pronounced. Although all grids initially capture the general oscillatory behaviour, error accumulation over time leads to degraded accuracy for coarser meshes. This is especially evident in the final second of the simulation, where coarse grids fail to resolve the oscillations accurately. The finest grid produces results that align well with expectations and show good agreement with the reference data.

Three-Dimensional Case

The three-dimensional test case is defined on a rectangular domain measuring 2.5 m $\times$ 0.41 m $\times$ 0.41 m in the streamwise x, vertical y, and spanwise z directions. A stationary cylinder is

located 0.45 m downstream from the inlet, 0.15 m above the bottom wall, and spans the entire width of the domain. All boundaries, including the top, bottom, and side walls, are treated as no-slip walls.

The inflow velocity for the three-dimensional 3D-3Z [93] test case follows a new time-dependent profile with a mean velocity of $U_m = 2.25$ m/s:

$$u(y,z,t) = \frac{16U_m yz(H-y)(H-z)}{H^4} \sin\left(\frac{\pi t}{8}\right) \tag{2.28}$$

where y and z are the vertical and spanwise coordinates, respectively, and $H = 0.41$ m is the domain height (and width in the spanwise direction). The flow is unsteady and laminar, with the simulation executed for 8 s.

The numerical setup, including solver, temporal and spatial discretisation, fluid properties, and force monitoring, is identical to that used in the two-dimensional case. Throughout the simulation, drag and lift coefficients, C_D and C_L, are continuously monitored to determine the aerodynamic behaviour of the cylinder under unsteady flow conditions. An overview of the computational domain is shown in Figure 2.3.

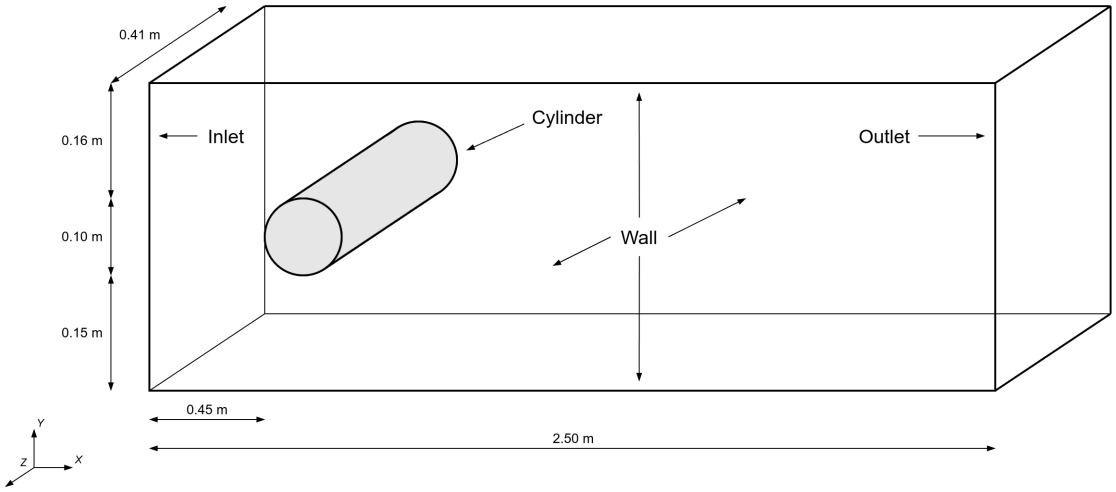
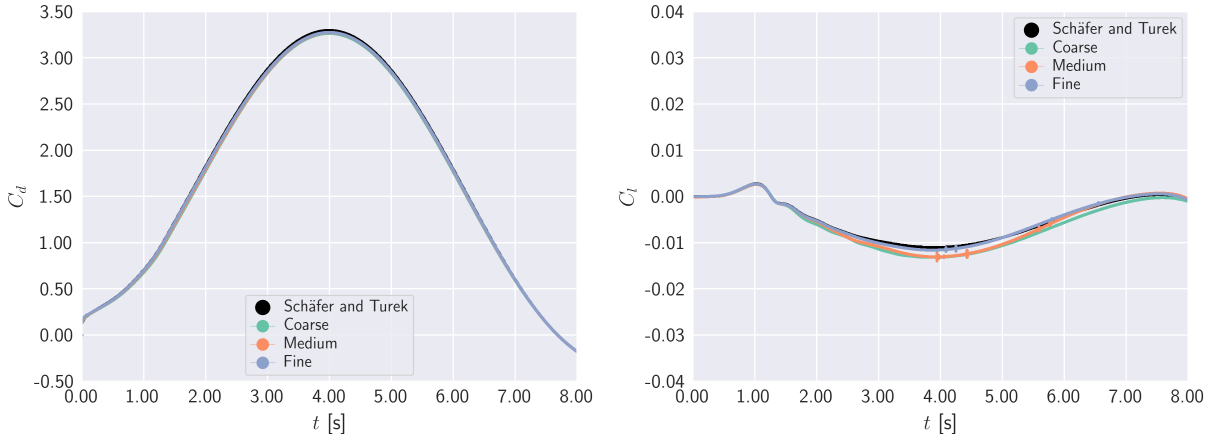


Figure 2.3: Computational domain for three-dimensional flow around a cylinder.

Three-Dimensional Flow Results

Analogously to the two-dimensional case, results for the three-dimensional case were assessed on three progressively finer grids with $r = 1.5$. Drag and lift coefficient results are shown in Figure 2.4.



(a) (b)

Figure 2.4: Resulting drag and lift coefficients for the three-dimensional cylinder case: (a) drag coefficient, (b) lift coefficient.

The findings for the drag coefficient are similar. Compared to the two-dimensional case, the errors are smaller but extend over a longer time interval. All grid types show acceptable agreement with the reference data from Schäfer and Turek [93].

In contrast, the lift coefficient results are significantly worse on the coarser grids. Between $t \approx 2$ s and $t \approx 6$ s, errors can reach up to 15 % on the coarser meshes, although they tend to decrease near the bounds of this interval. Conversely, the finest grid shows acceptable agreement with the reference data [93], with only minor discrepancies around $t = 4$ s.

2.5.2 Rising Bubble Dynamics

The two considered test cases are based on the works of Hysing et al. [45] and Adelsberger et al. [1]. Both involve isothermal, incompressible flow of immiscible fluids and simulate the unsteady rise of a bubble in a water column. The Reynolds number is 35 in both cases, i.e. the flow is laminar. The fluid properties for both test cases are summarised in Table 2.1. Given the nature of the problem, the VOF solver `interFoam` is used.

Table 2.1: Fluid properties employed in the bubble dynamics test cases.

ρ_1	ρ_2	μ_1	μ_2	$\mathbf{g}$	σ
1000 kg/m^3	100 kg/m^3	10 Pa·s	1 Pa·s	0.98 m/s^2	24.5 N/m

The volume of fluid method is a numerical technique designed to simulate the flow of two or more immiscible fluids. Introduced by Nichols and Hirt [43, 75], VOF belongs to the family of

interface-capturing methods. Unlike interface-tracking methods, which explicitly follow the geometric interface between fluids, VOF represents the interface implicitly using a scalar variable known as the volume fraction α, which defines the proportion of a control volume occupied by a given phase [43].

The evolution of the fluid interface is governed by a conservative transport equation for the volume fraction [17]:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{u}\alpha) = S_\alpha \tag{2.29}$$

where $\mathbf{u}$ is the velocity field and S_α is a source or a sink, and is typically zero, $S_\alpha = 0$ [17]. A fundamental constraint of the VOF method is that the sum of volume fractions for all phases in any control volume equals:

$$\sum_{i=1}^{n} \alpha_i = 1 \tag{2.30}$$

where α_i is the volume fraction of phase i, and n is the total number of fluid phases. In systems containing two fluids, it is therefore sufficient to solve for the volume fraction of one phase, with the other inferred from the constraint [19].

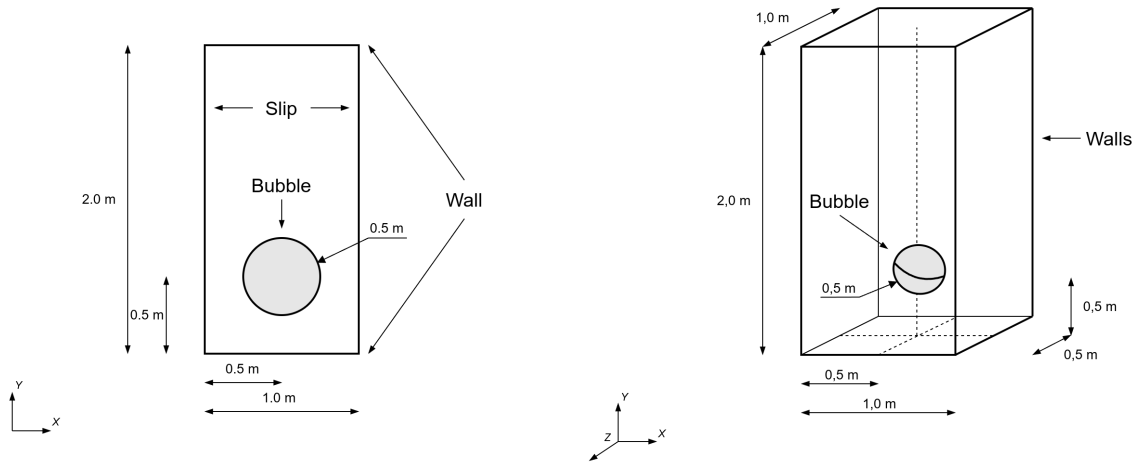
The VOF method employs a mixture approach, solving a single set of momentum equations for the entire fluid mixture [19]. Mixture properties like density and viscosity are calculated as volume-fraction-weighted averages of the individual phases:

$$\rho = \sum_{i=1}^{n} \alpha_i \rho_i, \quad \mu = \sum_{i=1}^{n} \alpha_i \mu_i \ . \tag{2.31}$$

Two-Dimensional Case

The computational domain for the two-dimensional case is a simple rectangle measuring 1.0 m $\times$ 2.0 m in the x- and y-directions, respectively. A cylinder with a radius of 0.5 m is placed at the centre of the domain in the x direction and 0.5 m above the bottom boundary. The mantle of the cylinder defines a sharp interface between two fluids: the fluid with density ρ_1 occupies the region outside, while the fluid with density ρ_2 is inside. The top and bottom boundaries are treated as no-slip walls, while the left and right boundaries are modelled as slip walls. The computational domain is shown in Figure 2.5a.

The simulation runtime is $t = 3$ s. Adaptive time-stepping is employed, and the maximum Courant number is limited to $Co_{\max} = 0.05$. Second-order accurate numerical schemes are used throughout. Time integration is performed with an implicit, bounded, second-order accurate



(a) (b)

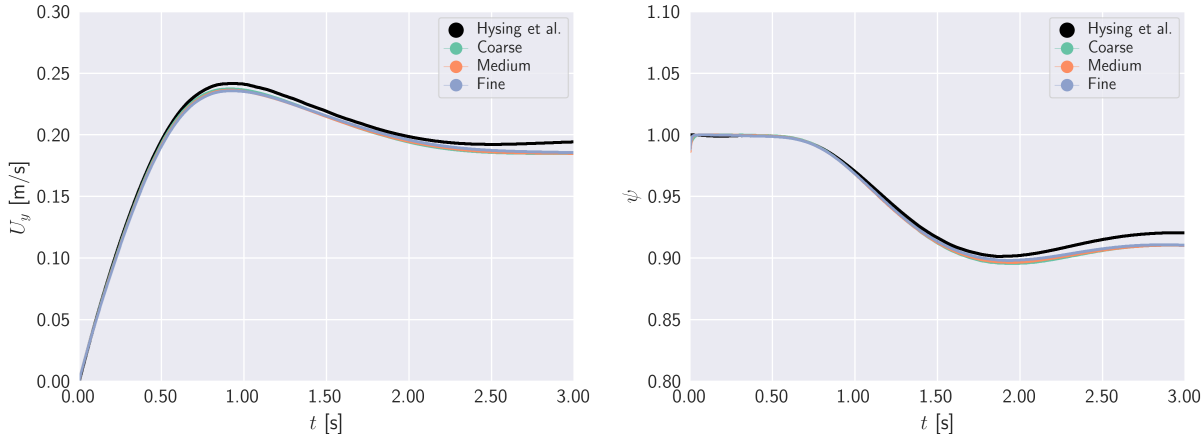
Figure 2.5: Computational domain for the rising bubble test case: (a) 2D domain, (b) 3D domain.

scheme. Spatial derivatives, including the gradient, Laplacian, and surface-normal gradient, are discretised using second-order accurate schemes. Convective terms are discretised using the `limitedLinear` scheme for velocity and the `Gauss interfaceCompression vanLeer 1` scheme for the volume fraction field. During runtime, bubble dynamics are monitored, specifically the centre of mass, rise velocity, and sphericity.

Two-Dimensional Flow Results

Three successively refined grids were used for the two-dimensional case. The refinement factor between levels was $r = 1.333$. The results obtained on each grid, along with the reference data from [45], are shown in Figure 2.6, for both the rise velocity, U_y, and the sphericity, ψ.

The data obtained from all three grids are similar for the rise velocity, with no significant differences observed between the grid types. Compared to the reference data, the maximum error does not exceed 5 %. The values are slightly underpredicted. Sphericity values follow a similarly consistent trend. The finest grid exhibits the smallest overall deviation. The errors remain low across all grid types, suggesting that the employed VOF method can accurately capture the bubble dynamics in the two-dimensional scenario.



(a) (b)

Figure 2.6: Results for the two-dimensional bubble dynamics case: (a) rise velocity, (b) sphericity.

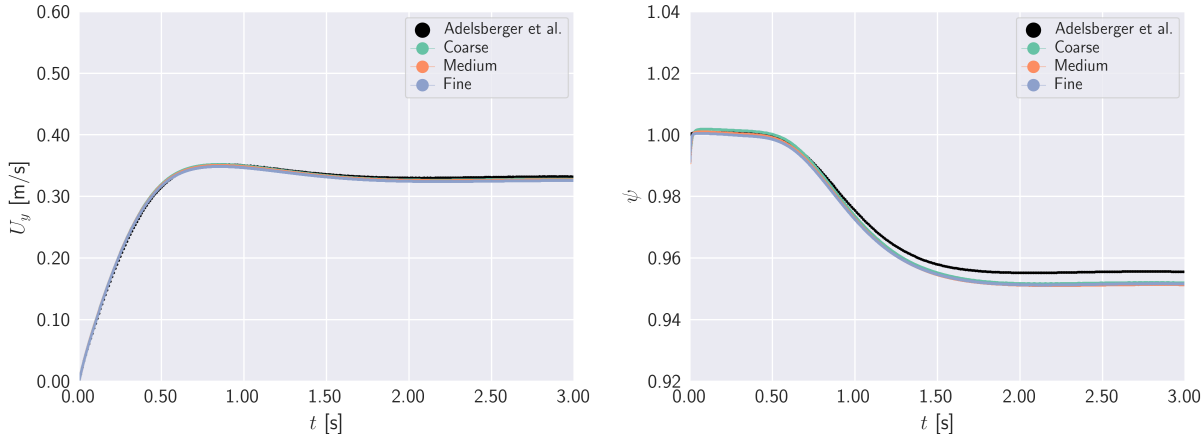
Three-Dimensional Case

The three-dimensional case is analogous to the previously defined two-dimensional case, with the only difference being the addition of the third dimension. The computational domain now measures 1.0 m $\times$ 2.0 m $\times$ 1.0 m in the x-, y-, and z-directions, respectively. The spherical bubble is centred in the x–z plane and placed 0.5 m above the bottom boundary. This added dimensionality implies new boundary conditions. In the three-dimensional case, all boundaries are treated as no-slip walls. All remaining parameters and numerical settings are retained from the two-dimensional setup and are described in the previous subsection.

In addition to the quantities already monitored during runtime, namely the centre of mass, rise velocity, and sphericity, the bubble diameters are now also computed. The three-dimensional computational domain is shown in Figure 2.5b.

Three-Dimensional Flow Results

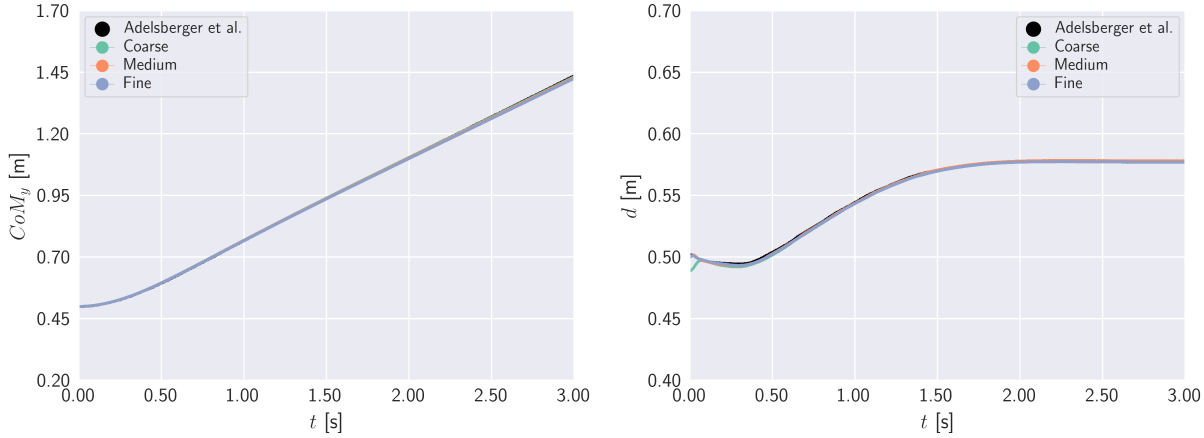
Analogously to the two-dimensional case, progressively finer grids with a refinement factor of $r = 1.333$ were used for the three-dimensional case. Based on the results shown in Figure 2.7, the rise velocity values are in excellent agreement with the reference data [1]. The sphericity, however, shows a measurable underprediction starting at approximately $t \approx 1$ s. Although present, this deviation is effectively small, consistent across all grids, and is in line with results from literature [1].



(a) (b)

Figure 2.7: Results for the three-dimensional bubble dynamics case: (a) rise velocity, (b) sphericity.

The computed position of the centre of mass, CoM_y, and the bubble diameter (measured along the x and z axes) also show excellent agreement with the reference data. Apart from a slight initial deviation in diameter observed on the coarse grid, all three grids - coarse, medium, and fine - closely match the reference data. These results are presented in Figure 2.8.



(a) (b)

Figure 2.8: Additional monitored metrics in the three-dimensional bubble dynamics case: (a) centre of mass, (b) bubble diameter along the x axis.

2.5.3 Breaking of a Dam

The following case replicates the experiment by Kleefsman et al. [54], which models the impact of a water wave on a container positioned on a ship's deck [54]. The computational domain is a rectangular tank measuring 3.22 m $\times$ 1.00 m $\times$ 1.00 m. All boundaries are treated as

no-slip walls except for the top. The top boundary uses a Neumann condition for velocity during outflow and a Dirichlet condition during inflow. A water column measuring 0.55 m in height and 1.228 m in length is initialised on the right side of the domain. The release is instantaneous at $t = 0$ s. A cuboid obstacle measuring 0.403 m $\times$ 0.161 m $\times$ 0.161 m is placed centrally, 0.6635 m from the left boundary. An overview of the domain configuration is shown in Figure 2.9.

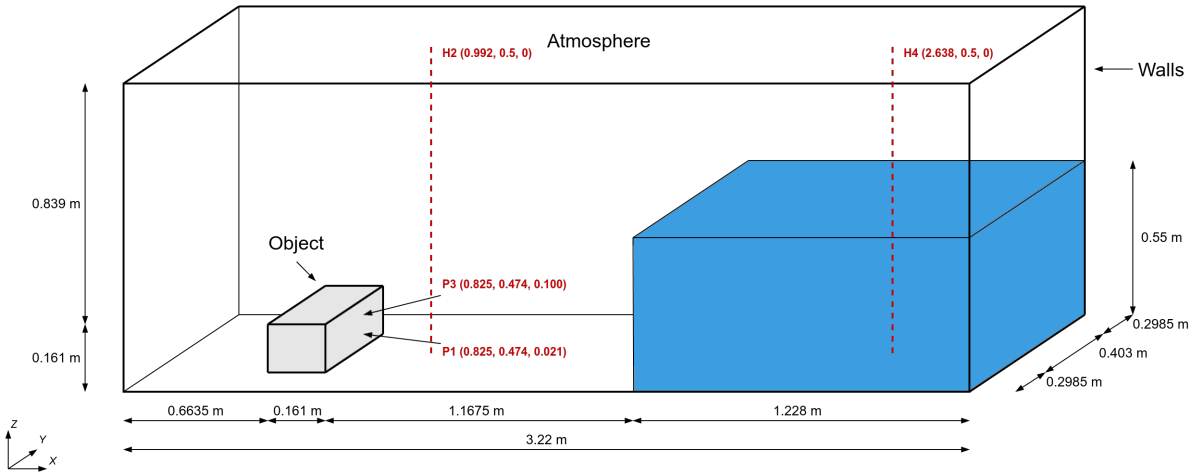


Figure 2.9: Computational domain for the dam break test case.

The problem is inherently multiphase. VOF solver `interFoam` is employed. Fluid properties are defined according to the original paper [54]. The SST k-ω turbulence model is used. Wall functions are applied at the boundaries, specifically `kqRWallFunction`, `nutkWallFunction`, and `omegaWallFunction` for turbulent kinetic energy, kinematic viscosity, and specific dissipation rate, respectively.

The SST model is a two-equation RANS model that solves transport equations for the turbulent kinetic energy, k, and the specific dissipation rate, ω [67]. The transport equation for k is:

$$\frac{\partial(\rho k)}{\partial t} + \nabla \cdot (\rho \mathbf{u} k) = \nabla \cdot \left[\left(\mu + \frac{\mu_t}{\sigma_k} \right) \nabla k \right] + P_k - \beta^* \rho k \omega + S_k \tag{2.32}$$

where P_k is the production term (often defined as $\tilde{P}_k = \min(P_k, C\rho k\omega)$, where C is a constant that limits the production of k), β^* is the model constant for the destruction of k, and S_k is a source term. The transport equation for ω is:

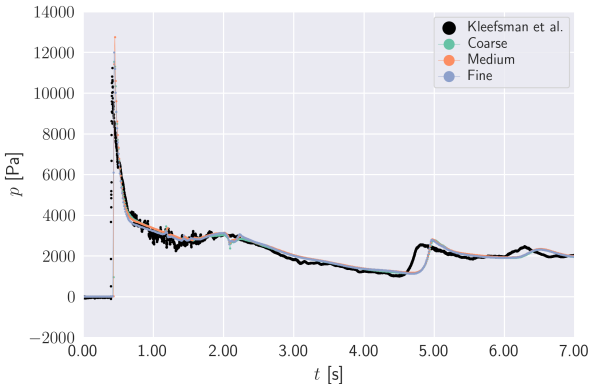
$$\frac{\partial(\rho \omega)}{\partial t} + \nabla \cdot (\rho \mathbf{u} \omega) = \nabla \cdot \left[\left(\mu + \frac{\mu_t}{\sigma_\omega} \right) \nabla \omega \right] + P_\omega - \beta \rho \omega^2 + D_\omega + S_\omega \tag{2.33}$$

where P_ω is the production term, σ_ω is the turbulent Prandtl number for ω, D_ω is the cross-diffusion term and S_ω a source term. The turbulent eddy viscosity is given by:

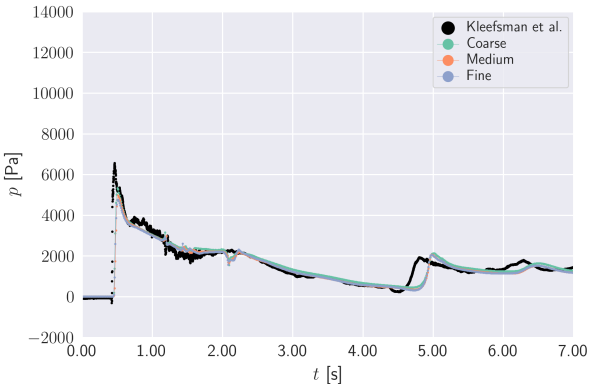
$$\mu_t = \frac{\rho a_1 k}{\max(a_1 \omega, S F_2)} \tag{2.34}$$

where a_1 is a model constant, S is the magnitude of the strain rate tensor, and F_2 is a second blending function.

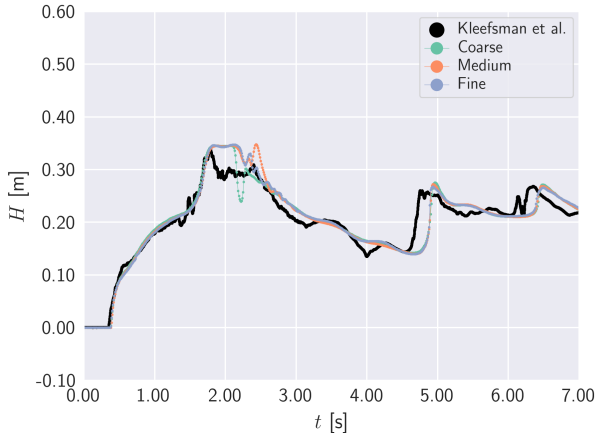
Time integration is performed with an implicit, bounded, second-order accurate scheme. Spatial derivatives, including the gradient, Laplacian, and surface-normal gradient, are discretised using second-order accurate schemes. Convective terms are discretised using first-order accurate schemes, except for the volume fraction, which uses `Gauss interfaceCompression vanLeer 1` scheme. Adaptive time-stepping is employed with a maximum Courant number limited to $Co_{\max} = 0.5$. The total simulation time is set to $t = 7$ s.



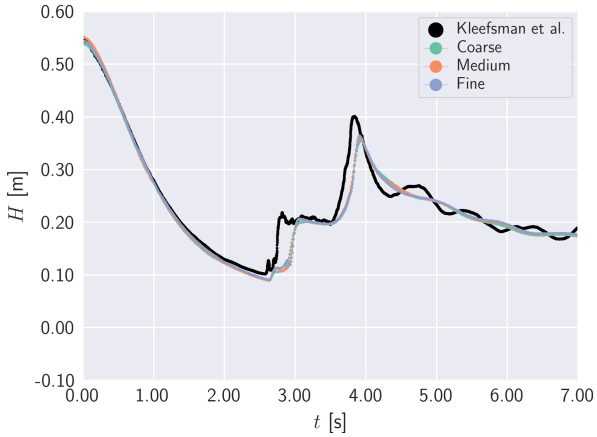
(a) Pressure at point P1.



(b) Pressure at point P3.



(c) Water height at point H2.



(d) Water height at point H4.

Figure 2.10: Selected results for the dam break test case using conventional (non-adaptive) grids.

Three distinct grids are considered - coarse, medium and fine. Grids are progressively finer with a refinement factor $r = 1.333$. A total of eight pressure probe locations and four water level monitoring locations are defined [54]. The water surface level is determined using the built-in `interfaceHeight` utility. Results are evaluated at four selected measurement locations. Selected results for water level and pressure are presented in Figure 2.10.

Pressure results show a slight overprediction and delayed response at point P1. However, the subsequent pressure wave aligns well with experimental data, with notable deviations only appearing after $t = 4$ s. At point P3, an initial underprediction is observed. The results agree well with experimental data up to $t = 4$ s. This holds true for all grids.

The water level is reasonably well captured at locations H2 and H4 up to $t \approx 1.5$ s. At H2, a plateau can be observed. While the general trend is well captured, it remains offset after $t \approx 2.5$ s and is not fully aligned with the experimental data. Similar behaviour is seen at H4, with a general underprediction of the water height. All grids exhibit similar trends, suggesting that discrepancies may also stem from limitations in the measurement utility.

2.6 Validation of LES Benchmark Cases

This section introduces a set of benchmark problems simulated using LES. Selected cases represent canonical turbulent flow scenarios: turbulent channel flow, flow around a square cylinder and turbulent mixing of a jet in crossflow. Each case is simulated using conventional (non-adaptive) grid generation, with numerical setups described in detail. Simulation results are compared against experimental measurements or high-fidelity numerical reference data. These benchmarks establish a baseline performance reference for evaluating the impact of adaptive mesh refinement on LES, as explored in later sections.

2.6.1 Turbulent Channel Flow

Turbulent channel flow is modelled for a friction Reynolds number $Re_\tau \approx 395$, where $Re_\tau = \delta U_\tau / \nu$, with $U_\tau = \sqrt{\tau_w / \rho}$. Here, δ denotes the channel half-height. The computational domain is rectangular, measuring 20π m $\times$ 2 m $\times$ π m in the x-, y-, and z-directions, respectively. The top and bottom boundaries are no-slip walls. The remaining walls are treated as periodics (`cyclic`). An overview of the computational domain is given in Figure 2.11.

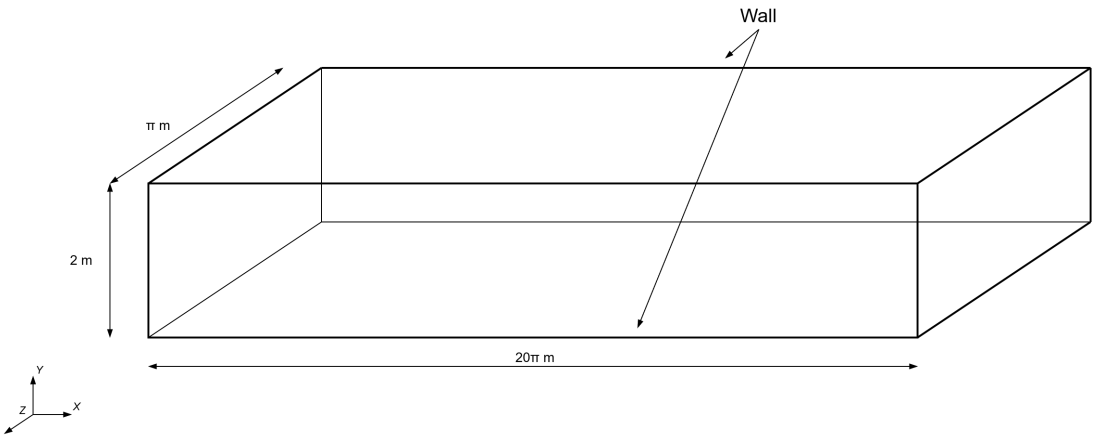


Figure 2.11: Computational domain for the turbulent channel test case.

The fluid properties and general case setup follow the DNS configuration by Moser et al. [70]. Flow perturbations were introduced using the `perturbU` utility [104]. As the utility was originally developed for an older version of OpenFOAM, it was ported to OpenFOAM 10 for use in this thesis. Starting from $t = 0$ s, the simulation was allowed to develop for 10 flow-through times, after which time-averaging was conducted over a period of 50 flow-through times. The simulation was conducted with a fixed time step of $\Delta t = 0.001$ s.

The computational mesh consists of $502 \times 64 \times 100$ cells, with grid grading applied in the wall-normal direction y. The corresponding dimensionless cell sizes are $\Delta x^+ \approx 49$, $\Delta y^+ \approx 1.8$ near the wall, increasing to $\Delta y^+ \approx 18$ in the channel core, and $\Delta z^+ \approx 12$. These values are generally consistent with the grid resolution reported in [104]. The unsteady incompressible solver `pimpleFoam` was used, with second-order accurate schemes applied to all terms, including time and convective terms.

Turbulence was modelled using the Smagorinsky subgrid-scale model [98] with van Driest damping to improve near-wall behaviour. Additionally, a test case was conducted using the dynamic Smagorinsky model proposed by Lilly [63]. The model implementation for OpenFOAM, originally developed by Passalacqua [2], was ported to OpenFOAM 10 for use in this thesis. The subgrid filter width was defined as the cube root of the local cell volume.

The Smagorinsky model [98] employs the eddy-viscosity hypothesis to close the filtered Navier–Stokes equations. It assumes a linear relationship between the deviatoric part of the subgrid-scale stress tensor and the rate of strain tensor of the resolved velocity field:

$$\tau^{SGS} = \tau - \frac{1}{3}\mathrm{tr}(\tau)\mathbf{I} = -2\nu_S\overline{\mathbf{S}} \tag{2.35}$$

where $\mathbf{I}$ is the identity tensor, ν_S is the eddy viscosity, and $\overline{\mathbf{S}}$ is the resolved rate of strain tensor. The Smagorinsky eddy viscosity ν_S is given by:

$$\nu_S = (C_s \Delta)^2 \left|\overline{\mathbf{S}}\right| \tag{2.36}$$

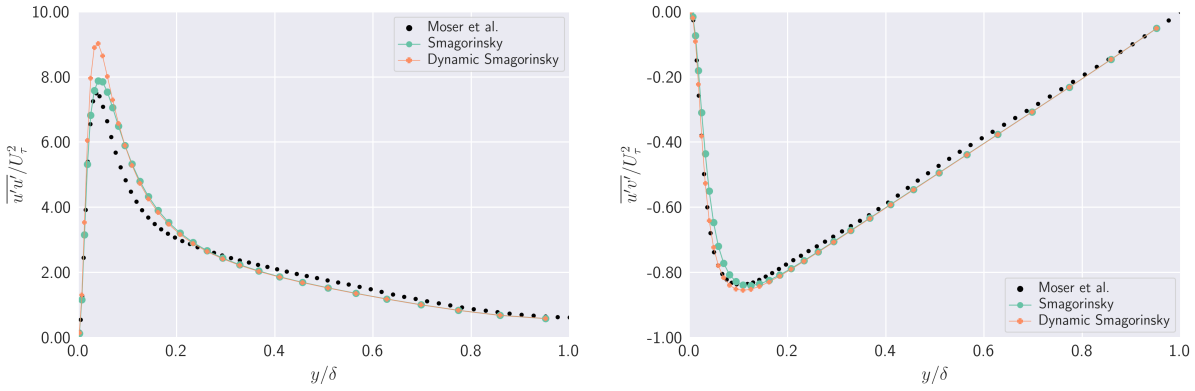
where C_s is the Smagorinsky constant, Δ is the filter width, and $\left|\overline{\mathbf{S}}\right|$ is the magnitude of the rate of strain tensor.

The dynamic Smagorinsky model [63] calculates the Smagorinsky coefficient, C_s, dynamically rather than using a fixed value, in order to minimise the error in the modelled Germano identity. The dynamic coefficient C_s is calculated by applying averaging as follows:

$$C_s^2 = \frac{\langle \mathbf{L}^d : \mathscr{M} \rangle}{\langle \mathscr{M} : \mathscr{M} \rangle} \tag{2.37}$$

where $\mathbf{L}^d$ is the deviatoric part of the filtered rate of strain tensor $\mathbf{L}$, and $\mathscr{M}$ represents the difference between the modelled and exact subgrid-scale stresses.

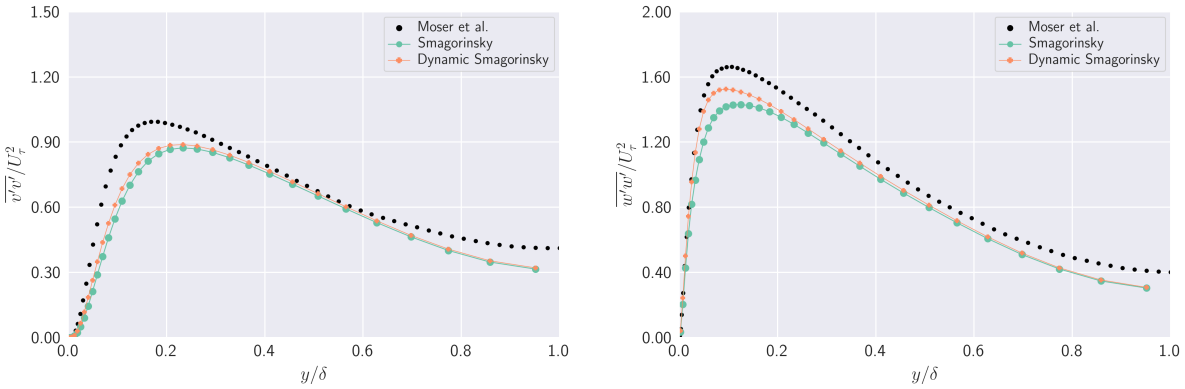
The results obtained for both models are generally similar, though some notable discrepancies exist. Since the underlying grid and overall setup are consistent, these differences can be attributed to the models. For $\overline{u'u'}/U_\tau^2$, a sharp spike is observed for the dynamic model, which quickly diminishes, after which the results align with the trend of the Smagorinsky model. The values for $\overline{u'v'}/U_\tau^2$ remain largely consistent, though the dynamic model's results show a better agreement with the DNS data. Figure 2.12 illustrates noted differences.



(a) (b)

Figure 2.12: Results for the channel flow case using a conventional (non-adaptive) grid: (a) normalised streamwise Reynolds normal stress $\overline{u'u'}/U_\tau^2$, (b) normalised Reynolds shear stress $\overline{u'v'}/U_\tau^2$.

The results for $\overline{v'v'}/U_\tau^2$ and $\overline{w'w'}/U_\tau^2$ exhibit larger discrepancies when compared to the DNS data, with notable underpredictions in both the spanwise and wall-normal directions. These discrepancies can be partially attributed to the models themselves, as well as the refinement of the underlying mesh. The errors are more pronounced in the spanwise direction, although the general trend is captured. The dynamic model tends to follow the DNS data more closely. The overall behaviour is expected and consistent with observations in the literature [104]. The results are shown in Figure 2.13.



(a) (b)

Figure 2.13: Results for the channel flow case using a conventional (non-adaptive) grid: (a) normalised wall-normal Reynolds normal stress $\overline{v'v'}/U_\tau^2$, (b) normalised spanwise Reynolds normal stress $\overline{w'w'}/U_\tau^2$.

2.6.2 Flow Around a Square Cylinder

Flow around a bluff body, a square cylinder, at a Reynolds number $Re = 21400$ was modelled using LES. The configuration is based on the work by Lyn et al. [65]. The square cylinder (a prism) is placed within a rectangular domain measuring 20.5D $\times$ 2D $\times$ 14D, where $D = 0.04$ m is the edge length of the square. The cylinder is located $4.5D$ downstream from the inlet and $6.5D$ from the bottom boundary. The only no-slip wall in the domain is the cylinder surface, which uses appropriate wall functions. An overview of the computational domain is shown in Figure 2.14.

The LES relies on a RANS precursor case, which uses the SST k-ω model. The overall setup and flow properties are maintained across both simulations to ensure consistency and data transfer. Unlike the LES, the RANS case employs symmetry boundary conditions for the outer domain. The working fluid is water, with kinematic viscosity $\nu = 1 \cdot 10^{-6}$ m^2/s. In both cases, the `pimpleFoam` solver is employed. The computational mesh for the RANS case consists of

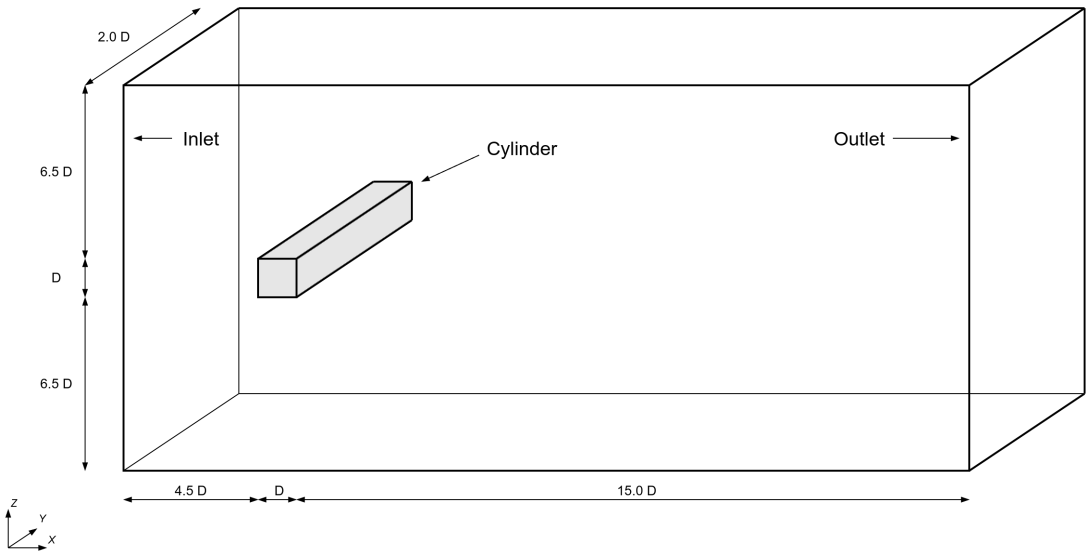


Figure 2.14: Computational domain for the square cylinder test case.

approximately 184 $\times$ 36 $\times$ 126 cells, while the LES mesh uses approximately 410 $\times$ 40 $\times$ 280 cells, with additional refinement near the cylinder. The first grid point in the wall-normal direction for LES corresponds to $\Delta z/D = 0.0015$, while the spanwise resolution is $\Delta y/D = 0.05$, which is finer than in similar studies [69]. $\Delta x/D$ is equivalent to the wall-normal spacing.

For the RANS simulation, adaptive time-stepping is used with a maximum Courant number $\mathrm{Co}_{\max} = 0.9$. A blended second-order/first-order accurate scheme is used for time. Spatial derivatives, including gradients, Laplacians, and surface-normal gradients, are discretised using second-order accurate schemes. The convective term for velocity uses `limitedLinear` scheme, with remaining terms using first-order accurate schemes.

The LES uses a fixed time step $\Delta t = 5 \cdot 10^{-5}$ s. The simulation was allowed to develop for 10 flow-through times, after which time-averaging was conducted over a period of 20 flow-through times. A second-order accurate, unbounded, implicit scheme is used for time. All spatial terms are discretised using second-order accurate schemes. The convective term for velocity is discretised using the `filteredLinear3V` scheme. The employed LES model is WALE [76], with the subgrid filter width defined as the cube root of the local cell volume.

The wall-adapting local eddy-viscosity (WALE) model was introduced by Nicoud and Ducros [76]. WALE is a zero-equation subgrid-scale turbulence model. It adapts to near-wall conditions and models unresolved stresses via an effective turbulent viscosity. The subgrid-scale eddy viscosity, ν_{SGS}, is calculated as follows:

$$\nu_{SGS} = (C_w \Delta)^2 \frac{\left(\mathbf{S}^d : \mathbf{S}^d\right)^{3/2}}{\left(\overline{\mathbf{S}} : \overline{\mathbf{S}}\right)^{5/2} + \left(\mathbf{S}^d : \mathbf{S}^d\right)^{5/4}} \tag{2.38}$$

where $C_w = 0.325$ is a model constant, and Δ is the filter width. The tensors $\overline{\mathbf{S}}$ and $\mathbf{S}^d$ are defined as:

$$\overline{\mathbf{S}} = \frac{1}{2}\left(\nabla\overline{\mathbf{u}} + (\nabla\overline{\mathbf{u}})^T\right), \tag{2.39}$$

$$\mathbf{S}^d = \frac{1}{2}\left(\mathbf{G}^2 + (\mathbf{G}^2)^T\right) - \frac{1}{3}\mathbf{I}\,\mathrm{tr}(\mathbf{G}^2) \tag{2.40}$$

where $\mathbf{G} = \nabla\overline{\mathbf{u}}$ is the velocity gradient tensor.

In addition to the aforementioned LES case, an additional case using the wall-modelled large eddy simulation (WMLES) approach is defined. Due to the lack of an available implementation, the `libWallModelledLES` library [71] was ported to OpenFOAM 10, with the LES model and setup remaining consistent across both cases.

WMLES divides the boundary layer into two regions: the outer region, where large eddies are resolved using LES and standard subgrid-scale models, and the inner region, where small eddies are modelled due to their resolution [81]. In the inner region, a wall-stress model estimates the wall shear stress, τ_w, typically based on the velocity field in a sampling layer within the logarithmic region:

$$\tau_w = \mu \left(\frac{\partial \overline{\mathbf{u}}}{\partial n}\right)_{\text{model}} \approx \tau_{\text{model}}(\overline{\mathbf{u}}, y_s, \nu)\ . \tag{2.41}$$

This stress is then imposed as a boundary condition for LES, linking the inner and outer regions.

Based on the Figure 2.15, results for u'/U agree well with the available experimental data. The profiles match closely across most cross-sections, with only minor deviations. The WMLES case shows slightly better agreement with the reference data.

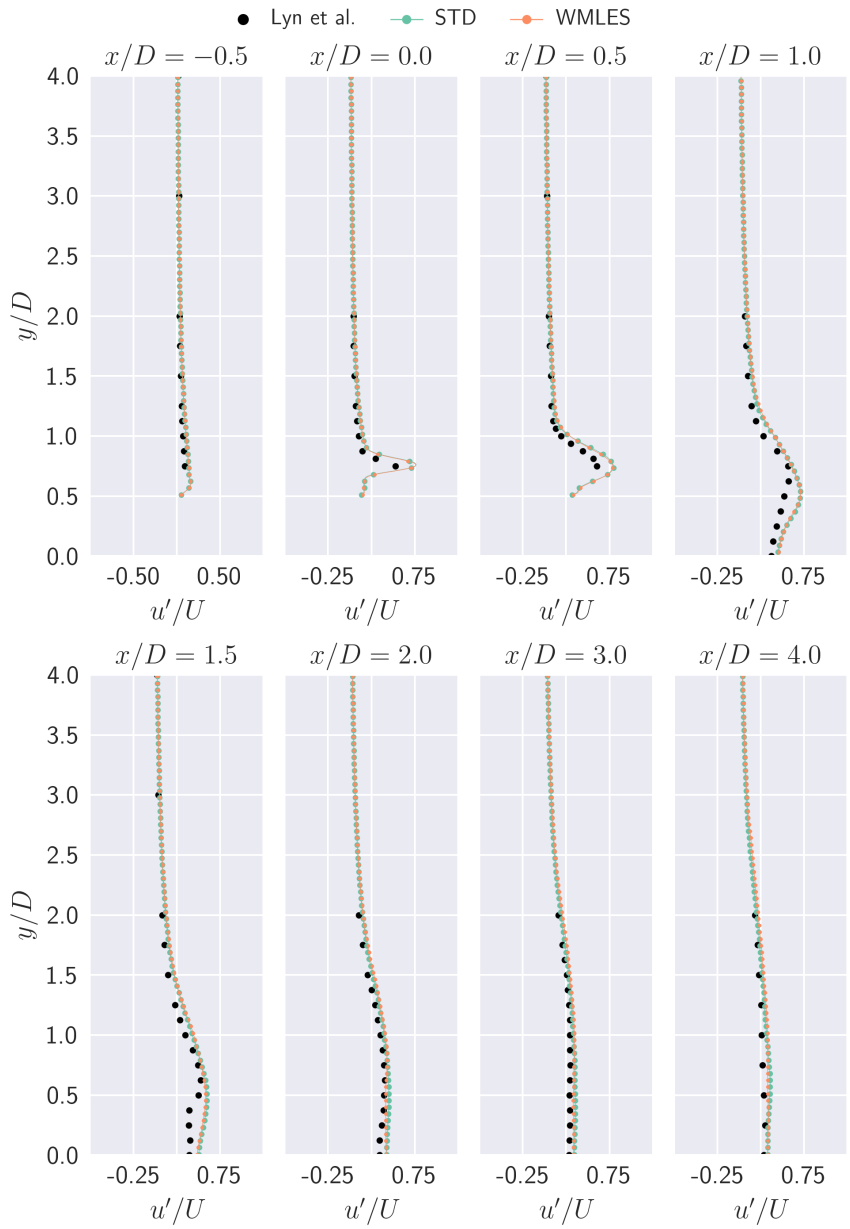


Figure 2.15: Profiles u'/U at various cross-sections for the square cylinder test case using a conventional (non-adaptive) grid.

For v'/U, as shown in Figure 2.16, at cross-sections $x/D = 0$, $x/D = 1.0$, and $x/D = 1.5$, irregular trends appear below $y/D \approx 1$ and $y/D \approx 0.5$ for the latter two. However, the results converge towards the reference data as y/D increases. These discrepancies are primarily due to limitations in near-wall resolution and modelling. Overall, the results remain in good agreement with the measurements.

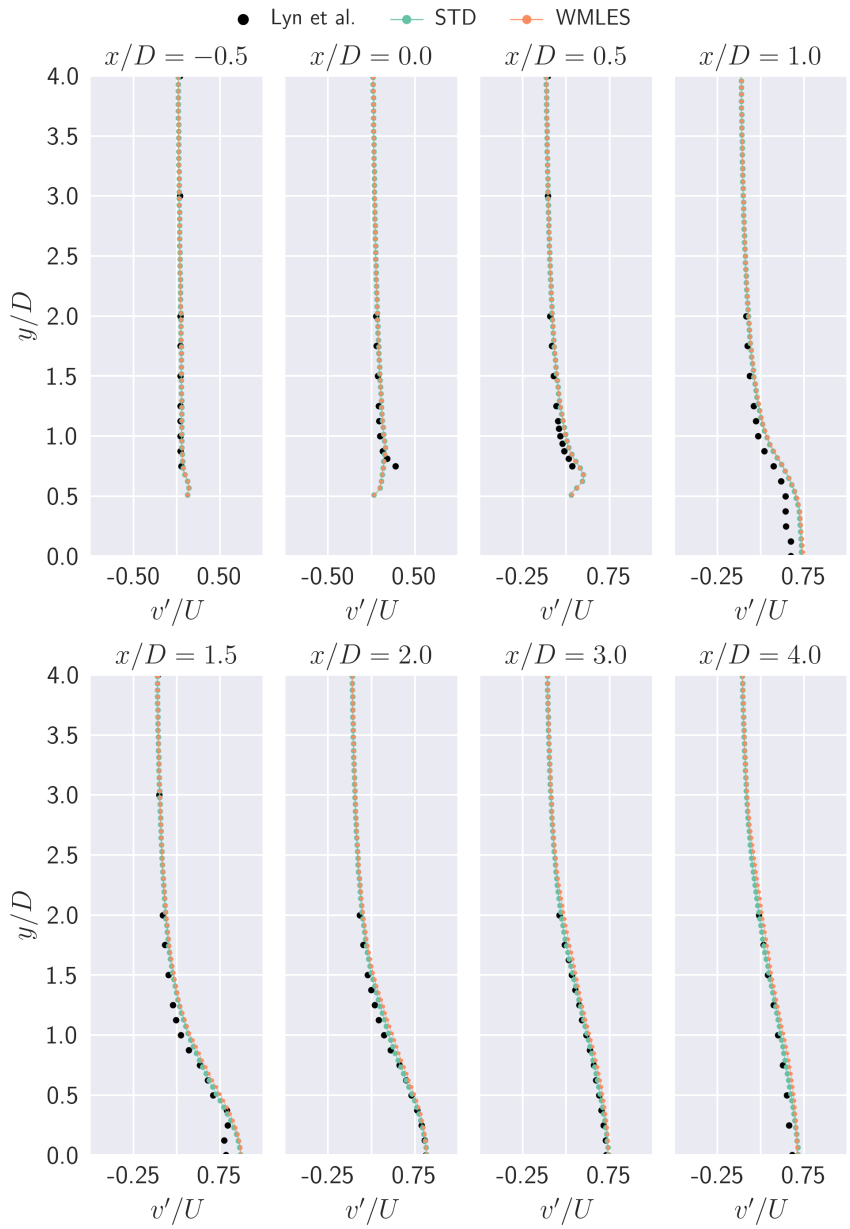


Figure 2.16: Profiles v'/U at various cross-sections for the square cylinder test case using a conventional (non-adaptive) grid.

2.6.3 Turbulent Mixing of Jet in Crossflow

The present test case investigates turbulent mixing induced by the discharge of a jet into a crossflow. The setup is based on the work of Galeazzo et al. [36, 37]. The computational domain is comprised of a circular jet inflow tube with a diameter $D = 0.008$ m and a length of 0.15 m, which is connected centrally to the main channel at a distance of 0.1 m downstream from the crossflow inlet. The main channel is rectangular, measuring 0.3 m in the streamwise direction and 0.108 m in both the vertical and spanwise directions. The domain includes two inlets - one

for the jet and one for the crossflow - and a single outlet. All remaining boundaries are treated as no-slip walls with appropriate wall-function modelling. A passive scalar, representing the effluent, is introduced through the jet inlet. The bulk velocity at the crossflow inlet is 9.08 m/s, while the jet inlet velocity is 37.72 m/s. These conditions correspond to Reynolds numbers of $6.24 \cdot 10^4$ and $1.92 \cdot 10^4$ for the crossflow and the jet, respectively, based on the fluid's kinematic viscosity $\nu = 1.57 \cdot 10^{-5}$ m^2/s. An overview of the computational domain is provided in Figure 2.17.

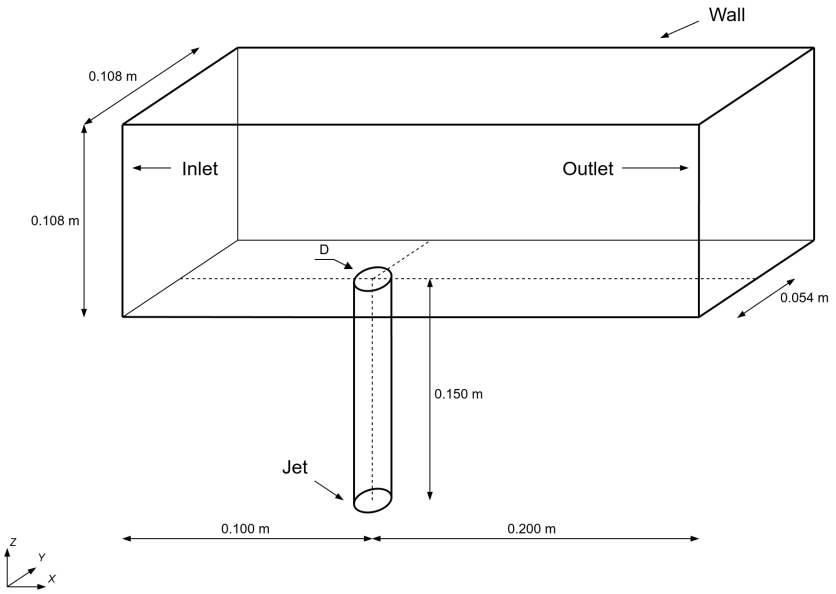


Figure 2.17: Computational domain for the turbulent mixing test case.

Following the methodology presented for the square cylinder test case, two LES cases were prepared, one using a conventional LES approach and the other employing a WMLES strategy. A precursor RANS simulation was performed using the SST k-ω turbulence model. All simulations were conducted using a custom solver, `scalarPimpleFoam`, based on `pimpleFoam` and extended to solve the scalar transport equation.

For the RANS simulation, a turbulent Schmidt number of 0.9 was employed. Adaptive time-stepping was used with $\mathrm{Co}_{\max} = 0.8$. A blended second-order/first-order accurate scheme was used for time. Convective terms for velocity and scalar transport were discretised using the `limitedLinear` scheme, while the remaining convective terms employed first-order schemes. Second-order accurate schemes were applied to all remaining terms.

In the LES simulations, the turbulent Schmidt number was set to 1.0. A mapping procedure was used to introduce turbulent fluctuations at both inlets. The LES computational mesh within the main channel consisted of 292 $\times$ 108 $\times$ 108 cells in the x-, y-, and z-directions, respectively.

The jet tube was discretised using a mesh with 24×150 cells in radial and axial directions. The WALE model was used, with the filter width defined as the cube root of the local cell volume. A fixed time step $\Delta t = 3 \cdot 10^{-6}$ s was employed. Second-order accurate schemes were used throughout. For velocity, `LUST` scheme was used, whereas for scalar transport `linearUpwind` scheme. Simulation execution time was $t = 1$ s, with averaging conducted over the final 0.5 s.

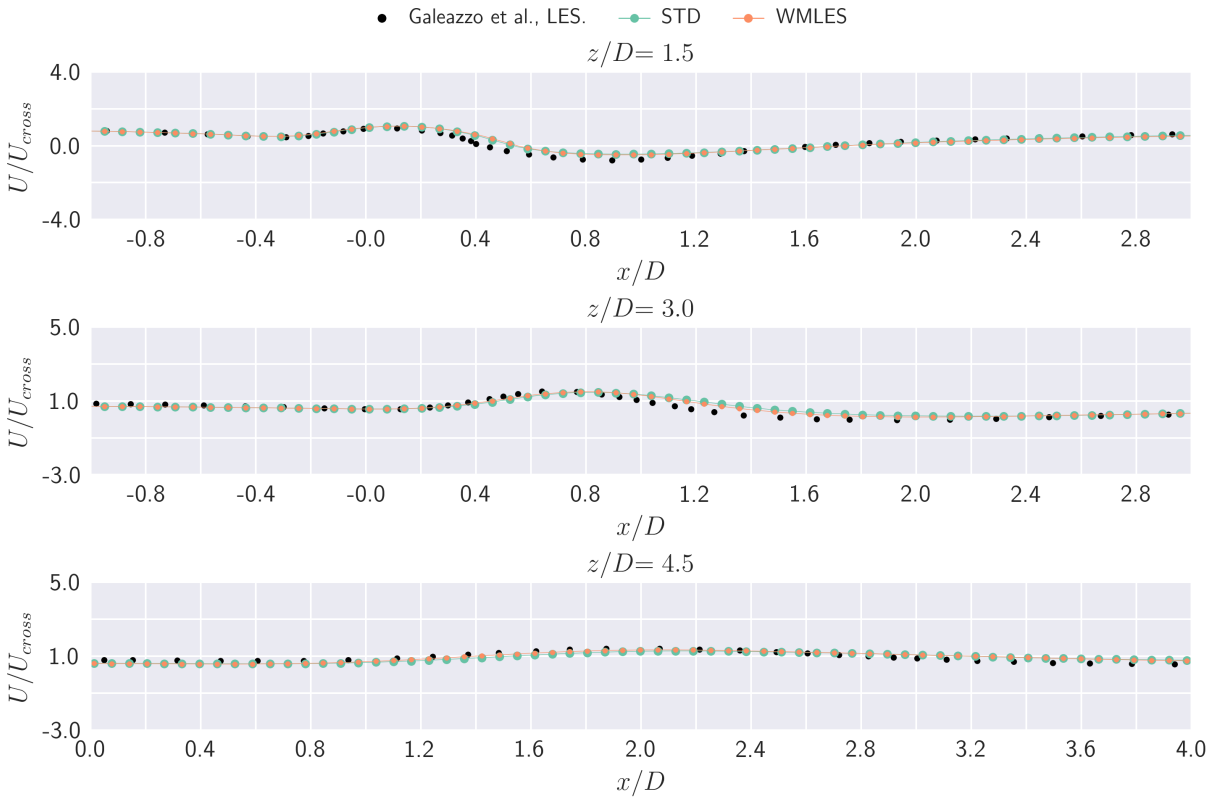


Figure 2.18: Results for the normalised velocity at different cross-sections for the turbulent mixing test case using a conventional (non-adaptive) grid.

Velocity results for the LES simulations are presented in Figure 2.18. Velocity data are normalised using the crossflow bulk velocity $U_{\text{cross}} = 9.43$ m/s. As shown, both the conventional LES and the WMLES approaches yield comparable results and show good agreement with the reference LES data reported by Galeazzo et al. [37]. With regards to the scalar concentration shown in Figure 2.19, several observations can be made. At lower z/D cross-sections, specifically $z/D = 1.5$ and $z/D = 3.0$, the results deviate from the reference numerical data. Such deviations are expected and have been documented in previous studies [97]. Interestingly, while the results at $z/D = 1.5$ diverge from the numerical reference data, they align better with the experimental measurements [36]. In the case of $z/D = 3.0$, the observed discrepancies are

consistent with findings in the literature and are strongly influenced by the numerical setup, particularly the discretisation schemes and turbulent Schmidt number. At $z/D = 4.5$, the computed scalar values align well with the reference numerical data.

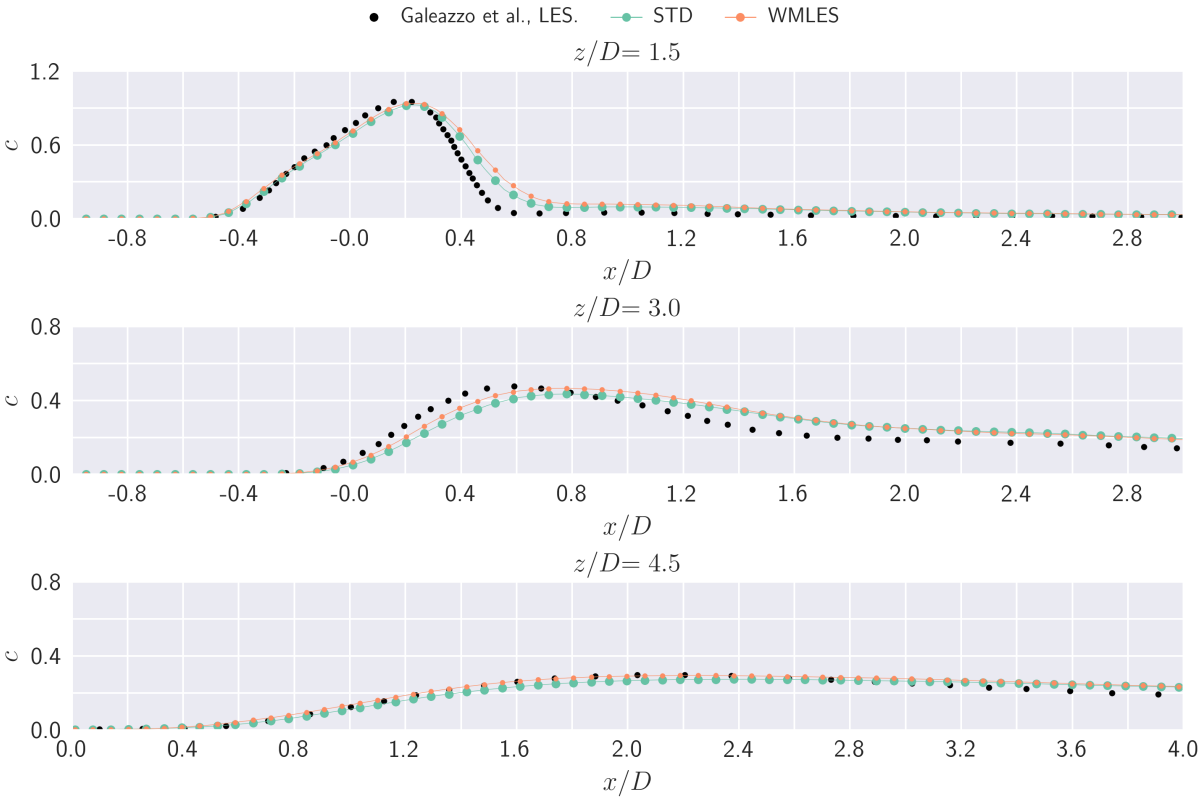


Figure 2.19: Results for the scalar concentration at different cross-sections for the turbulent mixing test case using a conventional (non-adaptive) grid.

3 ADAPTIVE MESH REFINEMENT IN OPENFOAM

Adaptive mesh refinement in OpenFOAM has evolved through community contributions and in-distribution developments, with core capabilities available via the `fvMeshTopoChangers` class. As of OpenFOAM 10, the framework supports isotropic refinement for hexahedral cells, leveraging `hexRef8` mesh cutter. While the OpenFOAM 10 implementation remains limited, various extensions and additions targeting different OpenFOAM variants and versions have been proposed. Karlsson [52] implemented anisotropic AMR in OpenFOAM by replacing the standard `hexRef8` mesh cutter with a directionally-aware alternative. Joshi [48] extended AMR to support arbitrary polyhedral cells through tetrahedral decomposition. A paper by Rettenmaier et al. [90] introduced several improvements to the AMR framework, including support for both 2D and 3D problems, extensions for refinement criteria, and a dynamic load balancing class. Lapointe et al. [58] extended the refinement mechanism to allow multiple criteria to govern mesh adaptation independently. Other studies, such as those by Kuo and Trujillo [57] and Sikirica et al. [97], focused on the overall performance of AMR, identifying computational bottlenecks and evaluating efficiency improvements.

3.1 Native Implementation

Historically, OpenFOAM managed dynamic mesh capabilities, including mesh motion and topological changes, through specialised subclasses derived from the `dynamicFvMesh` base class. One such subclass is `dynamicRefineFvMesh`, which was used in earlier versions of OpenFOAM and provided AMR functionality. However, as of OpenFOAM 10, this hierarchy has changed. Two distinct base classes were directly integrated into the standard `fvMesh` class:

- `fvMeshMover` which handles geometric changes, such as the movement of mesh points.
- `fvMeshTopoChanger` which manages topological changes, including adding or removing cells and faces.

This change in architecture eliminated the need for the `dynamicFvMesh` class, which was subsequently removed.

The AMR functionality is available through the `Foam::fvMeshTopoChangers::refiner` class. This class inherits from `fvMeshTopoChanger` and provides the logic for dynamic mesh refinement and unrefinement based on user-specified `volScalarField` values. To use AMR in a simulation, it is necessary to select this refiner within the `topoChanger` sub-dictionary of the `dynamicMeshDict` dictionary. The `refiner` class leverages the `hexRef8` class to perform uniform refinement of hexahedral mesh cells. A flowchart of the cell selection process using the `refiner` class is given in Figure 3.1.

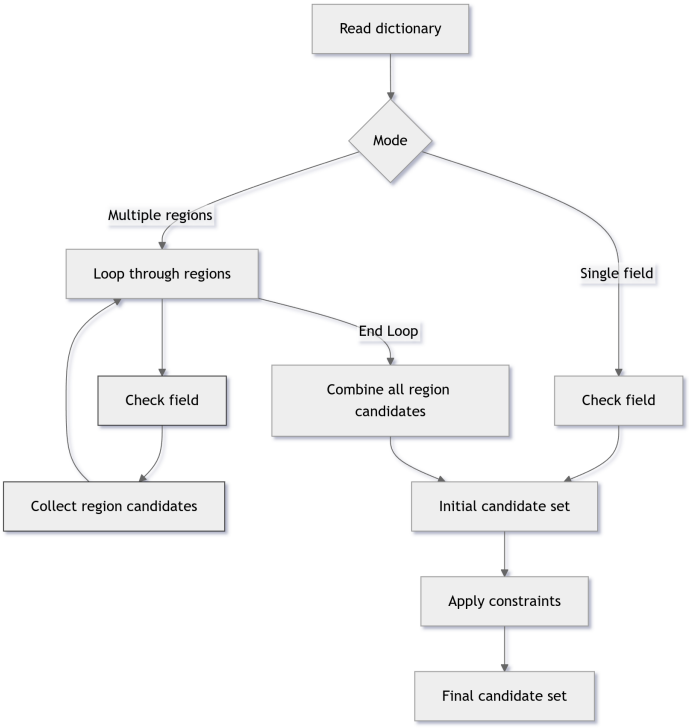


Figure 3.1: Refinement candidate selection flowchart for `refiner` class.

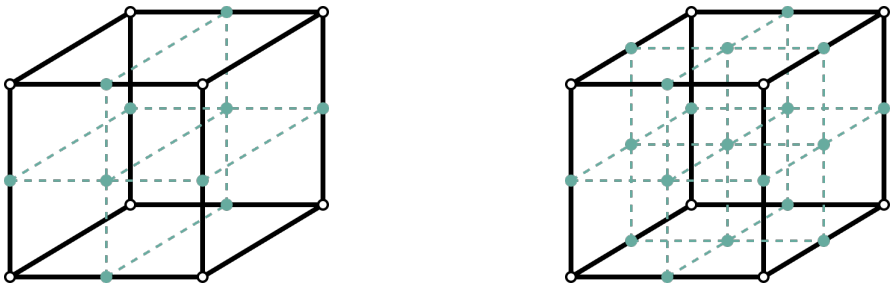
Since the native `refiner` class in OpenFOAM supports isotropic refinement of hexahedral (three-dimensional) grids, this chapter will focus on extending its capabilities to handle two-dimensional and three-dimensional problems. The applicability of AMR for these problems will be evaluated, along with its computational efficiency.

3.2 Extension for Two-Dimensional Problems

The two-dimensional AMR implementation is built upon the original three-dimensional AMR framework. Given that OpenFOAM computes 2D problems using a single-cell-thick 3D mesh, the code was modified to support the planar subdivision of hexahedral cells into four smaller hexahedra, thus preserving the geometrical and numerical characteristics of a two-dimensional configuration. These functional changes are implemented in the newly defined `hexRef4` and `refiner2D` classes.

3.2.1 Implementation Details

The newly defined `hexRef4` class is introduced alongside the original `hexRef8` mesh cutter. The primary distinction lies in the subdivision strategy: while `hexRef8` subdivides a cell into eight by introducing a central cell midpoint, `hexRef4` performs a four-way split using only edge and face midpoints (Figure 3.2). Consequently, the cell midpoint is omitted, and the refinement logic in `setRefinement` is simplified to reflect the reduced geometric complexity. The choice of mesh cutter is user-specified; for the purposes of this thesis, separate classes were implemented to apply the appropriate cutter based on the refinement algorithm.



(a) (b)

Figure 3.2: Cell subdivision resulting from: (a) `hexRef4`, (b) `hexRef8`.

Face and edge handling routines are adapted to enforce 2D-specific constraints. `hexRef4` introduces logic to prevent the refinement of non-quadrilateral faces and boundary edges using checks such as `isDivisibleFace` and `isDivisibleEdge`. Internal face creation is also restructured; rather than connecting a central point to surrounding geometry, `hexRef4` assembles four internal faces by connecting face midpoints to edge midpoints only. Unrefinement logic shifts from a point-based to an edge-based paradigm. Candidate edges are identified as

those shared by exactly four cells originating from the same refinement. Figure 3.3 provides a simplified summary of the logic implemented in the `hexRef4` class.

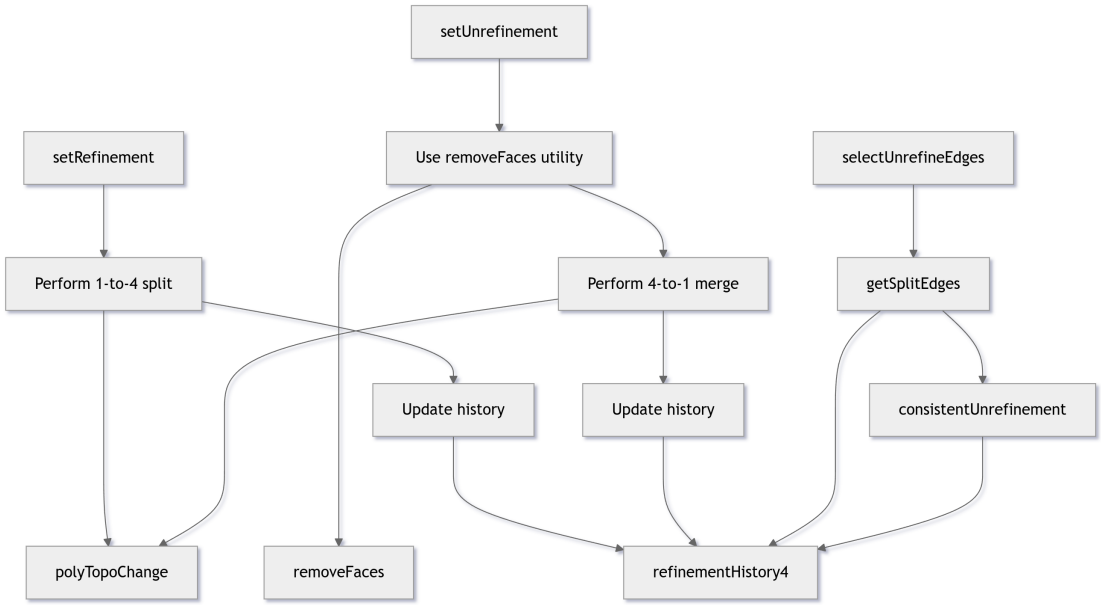


Figure 3.3: Simplified flowchart for the `hexRef4` class.

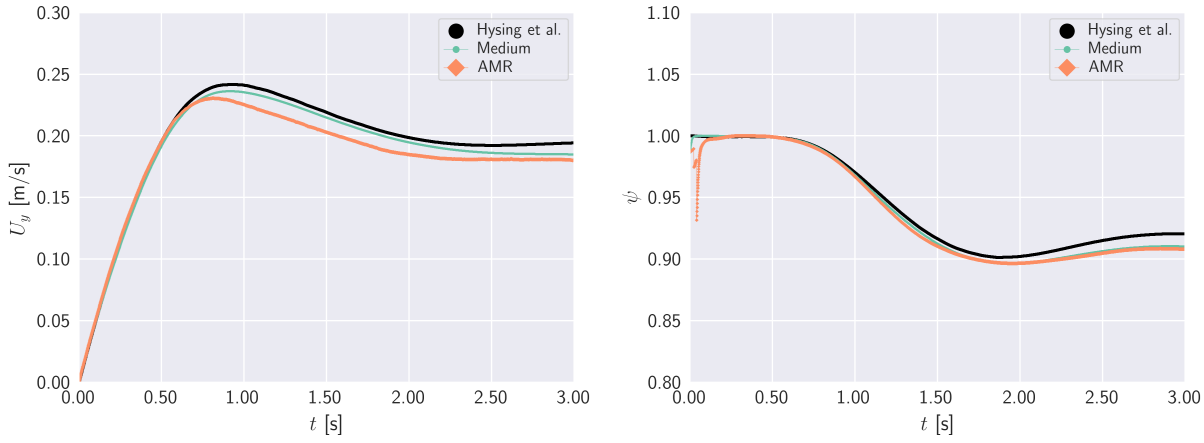
The `refiner2D` class governs refinement and unrefinement based on field-based criteria. It replaces the native `refiner` class, reusing much of the control logic, but adapted for the 2D refinement context. A simplified threshold-based scheme is employed for refinement. Candidate cells are identified through direct comparison with user-specified upper and lower limits. Unrefinement is handled by `selectUnrefineEdges`, which performs edge discovery, consistency checks, and filtering based on proximity to active refinement regions. This streamlines the overall code logic. The cell protection mechanism, used to prevent the refinement of degenerate or incompatible elements, retains its original structure.

In addition to the noted changes, the core infrastructure for tracking cell refinement history was redefined as `refinementHistory4`. This included modifying the underlying data structure to reference four child cells instead of eight. Similarly, `refinementHistoryConstraint4` and `hexRef4Data` were updated to operate correctly with the 1-to-4 splitting logic inherent to 2D refinement.

3.2.2 Validation for Two-Dimensional Problems

The implemented code is validated on a two-dimensional rising bubble test case. The overall setup, as described in Subsection 2.5.2, remains unchanged apart from the inclusion of the `dynamicMeshDict` dictionary and the definition of a refinement criterion to control mesh adaptation. A simple scalar-based criterion using the `alpha.water` field is applied, with refinement thresholds set to 0.01 and 0.99. Additionally, to preserve refinement history during parallel execution, `refinementHistory4` must be specified as a constraint in the `decomposeParDict` dictionary.

The initial cell size was set to be three times larger than that of the coarse mesh used in the conventional case. With two levels of refinement allowed, this theoretically enables AMR to reach the same resolution as the medium mesh and thus achieve comparable accuracy. However, this is not observed in practice. As shown in Figure 3.4, the AMR results exhibit noticeably lower accuracy. Minor improvements can be obtained by adjusting the refinement range or expanding the buffer zone. Several factors contribute to these results. First, the initially coarse mesh introduces significant numerical errors during the early time steps. These errors persist and are not corrected, even as the mesh refines dynamically. Second, the initial refinement criterion is too limiting, failing to capture a sufficiently large region to offset the effects of the coarse initial discretisation. As a result, the AMR solution does not reach the accuracy of the medium-resolution conventional mesh. Still, the results confirm that the AMR implementation is functionally correct and that the underlying two-dimensional refinement operates as intended.



(a) (b)

Figure 3.4: Results for the bubble dynamics case obtained using the `refiner2D` class: (a) rise velocity, (b) sphericity.

3.3 Validation for Three-Dimensional Problems

OpenFOAM 10 natively supports three-dimensional adaptive mesh refinement through `refiner` class. `refiner` allows refinement of the computational domain or specific regions based on scalar field values. The class relies on the `hexRef8` mesh cutter, which performs isotropic refinement by subdividing each hexahedral cell into eight smaller hexahedra.

Three-dimensional functionality was assessed on a rising bubble test case analogously to two-dimensional validation. The test setup mirrors that used for the conventional rising bubble case. Refinement is governed by the `alpha.water` field, with refinement thresholds set to 0.01 and 0.99. The initial cell size is three times larger than that of the coarse mesh, with two refinement levels allowed.

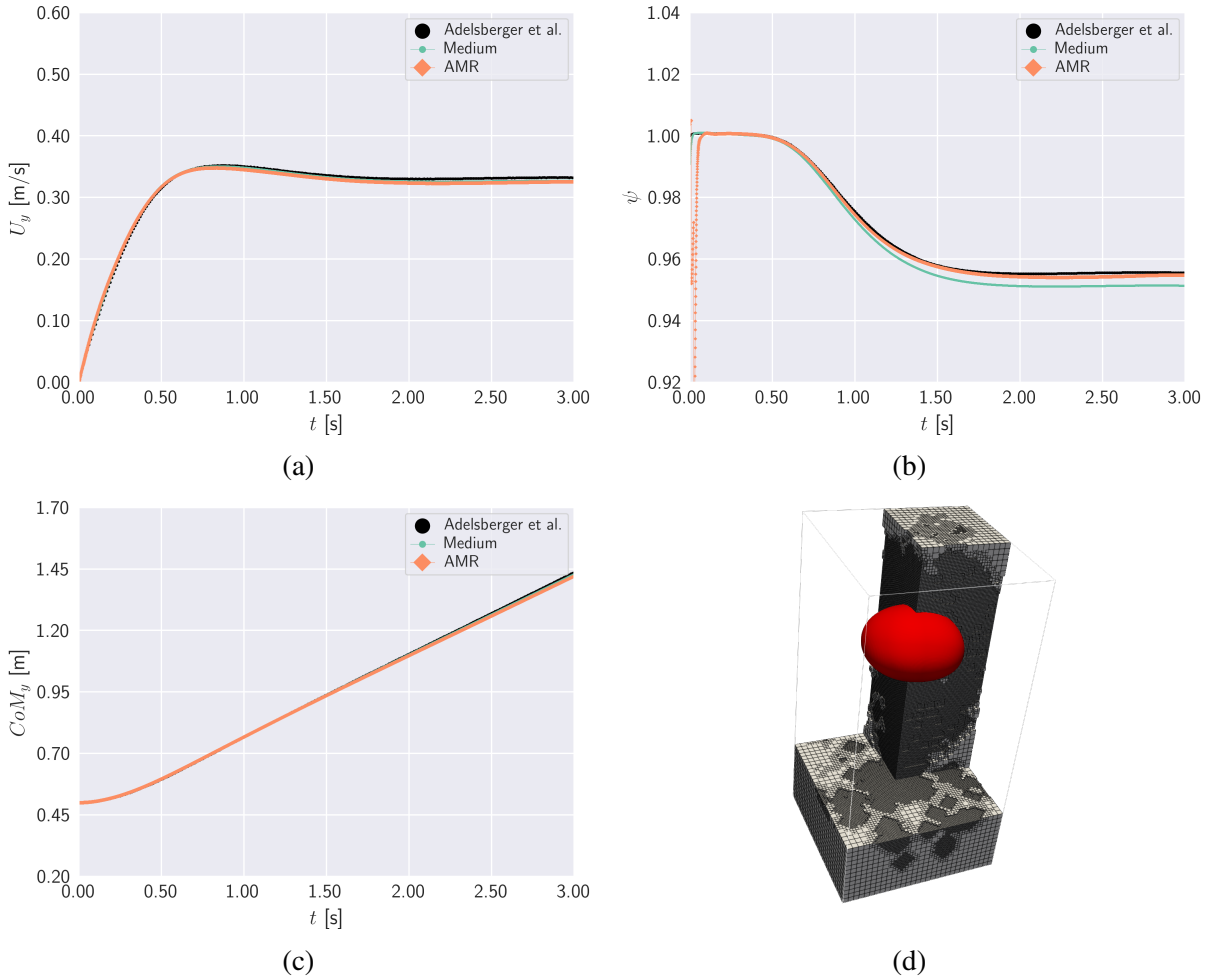


Figure 3.5: Results for the bubble dynamics case obtained using the `refiner` class: (a) rise velocity, (b) sphericity, (c) centre of mass, (d) computational mesh and the bubble.

The results for the three-dimensional case, shown in Figure 3.5, differ considerably from those observed in the two-dimensional case. The values of the rise velocity, U_y, are comparable to those obtained using the medium-resolution conventional grid. Concerning sphericity, the AMR solution shows better agreement with the reference data, although some oscillatory behaviour is observed. This can be attributed to the overly coarse initial mesh, which is subsequently sufficiently refined in the three-dimensional case, allowing the solution to align with the reference data after approximately $t \approx 0.1$ s. Overall, the results are satisfactory and achieved using a notably coarser grid.

4 MULTI-CRITERIA ADAPTIVE MESH REFINEMENT

The native adaptive mesh refinement implementation in OpenFOAM 10, specifically the `refiner` class, allows refinement based only on a single scalar field. This lack of flexibility limits its applicability in complex simulations. To address this limitation, a multi-criteria adaptive mesh refinement (mcAMR) algorithm has been proposed for both two-dimensional and three-dimensional problems. This algorithm extends the functionality of the previously introduced `refiner2D` and `refiner` classes.

4.1 Multi-Criteria Refinement Strategy

The newly introduced `multiFieldRefiner` classes rely on logical operators to combine multiple scalar fields and geometric constraints. In the original `refiner` implementation, refinement is driven by a single scalar field specified directly in the `topoChanger` dictionary using the `field` keyword. Alternatively, the `refinementRegions` subdictionary could be used to define criteria within spatially distinct regions. However, the logic applied to these regions is independent, and the framework lacks an explicit mechanism to logically combine refinement criteria across different fields or regions.

4.1.1 Mathematical Formulation

Let ϕ_i denote the i^{th} scalar field defined over the computational domain Ω, and associate with each defined field:

- a lower threshold δ_i,
- an upper threshold ε_i,
- and a logical operation $\mathscr{O}_i \in \{\cup, \cap, \setminus\}$.

Refinement is triggered for cells c where the field value lies within the specified threshold interval $(\delta_i,\ \varepsilon_i)$, while unrefinement occurs when the value lies outside this interval.

The refinement process starts on a default field ϕ_0, which is used to define the initial candidate set:

$$\mathscr{R}_0 = \{c \in \Omega \mid \delta_0 < \phi_0(c) < \varepsilon_0\} \ . \tag{4.1}$$

For each subsequent field ϕ_i, where $i = 1, \ldots, N$, a temporary set is constructed:

$$\mathscr{T}_i = \{c \in \Omega \mid \delta_i < \phi_i(c) < \varepsilon_i\} \ . \tag{4.2}$$

The refinement candidate state is updated according to the following:

$$\mathscr{R}_i = \mathscr{R}_{i-1} \ \mathscr{O}_i \ \mathscr{T}_i \ . \tag{4.3}$$

The final refinement set is $\mathscr{R}_N$ and contains all the cells to be refined.

Unrefinement follows a similar logic but identifies cells outside the refinement interval. The initial unrefinement set is:

$$\mathscr{U}_0 = \{c \in \Omega \mid \phi_0(c) \leq \delta_0 \ \cup \ \phi_0(c) \geq \varepsilon_0\} \tag{4.4}$$

with subsequent fields evaluated as:

$$\mathscr{V}_i = \{c \in \Omega \mid \phi_i(c) \leq \delta_i \ \cup \ \phi_i(c) \geq \varepsilon_i\} \ , \tag{4.5}$$

$$\mathscr{U}_i = \mathscr{U}_{i-1} \ \mathscr{O}_i \ \mathscr{V}_i \ . \tag{4.6}$$

The final unrefinement set is $\mathscr{U}_N$ and contains all the cells eligible for coarsening.

4.1.2 Implementation Details

The `multiFieldRefiner2D` and `multiFieldRefiner3D` classes introduce a new dictionary structure called `refinementFields`, where each entry corresponds to a refinement criterion associated with a scalar field or a geometric region. For scalar fields, each block contains the field name along with refinement thresholds `lowerRefineLevel` and `upperRefineLevel`. The class includes a dedicated function, `processFieldRefinement`, which evaluates each scalar field independently and identifies cells that meet the corresponding refinement conditions. This

subsequently allows multiple volScalarFields, such as alpha.water or p_rgh, to be considered jointly.

A key addition is the support for logical operators that combine refinement criteria from multiple blocks. Each block in the refinementFields dictionary includes an operation keyword. One block must be designated as the default, or one is assumed, establishing the initial set of candidate cells. The remaining blocks modify this set using logical operators. This logic is implemented in the selectRefineCandidates function, which processes each block in sequence and updates the candidate list accordingly.

In addition to scalar field-based refinement, the multiFieldRefiner classes support geometry-based criteria. Within the same refinementFields dictionary, a block of type geometry can be defined. This block enables the use of an external surface file, an STL file, and specifies whether refinement should occur inside or outside the surface. An optional buffer distance can also be applied to extend or shrink the effective geometry boundary. The function geometryRefineCandidates evaluates these criteria and determines which cell centres fall within the specified geometric region. The resulting cell set can then be logically combined with other criteria through the same operation mechanism. An example of the criteria definition block is included in Code 4.1.

Code 4.1: Segment of the dynamicMeshDict configuration file defining refinement criteria.

```
refinementFields
{
    alpha.water
    {
        lowerRefineLevel  : 0.001
        upperRefineLevel  : 0.999
        operation         : default
    }

    p_rgh
    {
        lowerRefineLevel  : 1e4
        upperRefineLevel  : 1e5
        operation         : AND
    }

    geometry
    {
        type              : geometry
        surface           : "surface.stl"
        mode              : inside
        buffer            : 0.1
        operation         : AND
    }
}
```

The decision-making process for identifying refinement candidates in the `multiFieldRefiner` classes is substantially more sophisticated than the original implementation. An overview of the cell selection process is provided in Figure 4.1. The implemented solution addresses the limitations of single-field or region-based approaches and supports complex refinement strategies. This makes it particularly well suited for cases where refinement must respond to spatial features not easily captured by single scalar field thresholds alone.

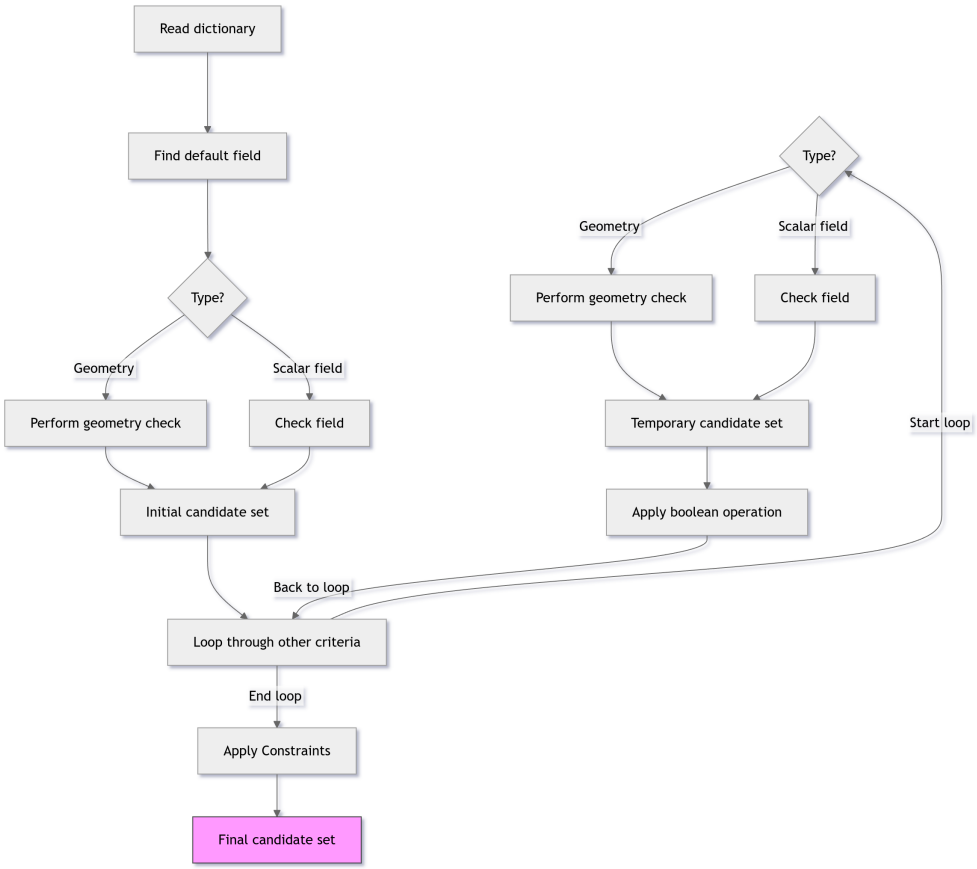


Figure 4.1: Flowchart of the mcAMR cell selection process.

4.2 Application of mcAMR to Two-Dimensional Problems

In a previous chapter, the rising bubble dynamics test case was used to validate the two-dimensional implementation of the `refiner2D` class. While the test case demonstrated the basic functionality, the numerical results were unsatisfactory. This chapter will re-evaluate the same case using the newly implemented `multiFieldRefiner2D` class. Furthermore, the flow around a cylinder test case will be used to assess the new implementation, providing a broader validation context of the `multiFieldRefiner2D` class.

4.2.1 Criteria and Validation for 2D Rising Bubble Dynamics

The refinement criteria are calculated during runtime using a coded function object in OpenFOAM. These calculations are performed just before the refinement process is executed. The `smoothField` function (S) is used to apply spatial smoothing to selected scalar fields by averaging the value in each cell with those of its neighbouring cells. The averaging is weighted based on the distance between the cells, with closer neighbours having a greater influence. The weight for each neighbour is determined using a Gaussian function:

$$w_i = \exp\left(-\frac{d_i^2}{2\sigma^2}\right) \tag{4.7}$$

where d_i is the distance between the centres of the neighbouring cells, and σ is a smoothing parameter calculated as:

$$\sigma = f \cdot \left(\frac{\sum V}{n_{\text{cells}}}\right)^{\frac{1}{3}} . \tag{4.8}$$

Here, $\sum V$ represents the sum of the volumes of all cells in the mesh, and n_{cells} is the total number of cells. The constant $f = 4.0$ is user-defined and controls the smoothing scale. Finally, the smoothing is computed as:

$$S(\phi) = \frac{\phi_i + \sum_{j \in \mathcal{N}_i} w_{ij}\phi_j}{1 + \sum_{j \in \mathcal{N}_i} w_{ij}} \tag{4.9}$$

where $\mathcal{N}_i$ denotes the set of neighbouring cells of cell i. The smoothing function is used to calculate the normalised smoothed gradient of the alpha field, which serves as a refinement criterion:

$$\phi_0 = \frac{S(|\nabla\alpha|)}{\max\left(S(|\nabla\alpha|)\right)} . \tag{4.10}$$

Similarly, the normalised rise velocity is used as a refinement criterion:

$$\phi_1 = \frac{S(|U_y|)}{\max\left(S(|U_y|)\right)} . \tag{4.11}$$

Refinement thresholds are set to 0.25 and 1.0 for both criteria, with refinement performed every 16 time steps. The overall computational setup is identical to the setup described in Subsection 3.2.2.

The results obtained using the mcAMR approach are presented in Figure 4.2. They are now

compared to the results on the fine grid despite theoretically only being able to achieve the accuracy of the medium grid. The mcAMR approach provides significant improvements over the initial results using the `refiner2D`. The rise velocity is well captured, surpassing the accuracy of the fine grid and closely matching the reference data. While sphericity initially exhibited some oscillations, it quickly stabilised. Similarly, the centre of mass behaves in accordance with the reference data, outperforming the fine grid results in terms of accuracy.

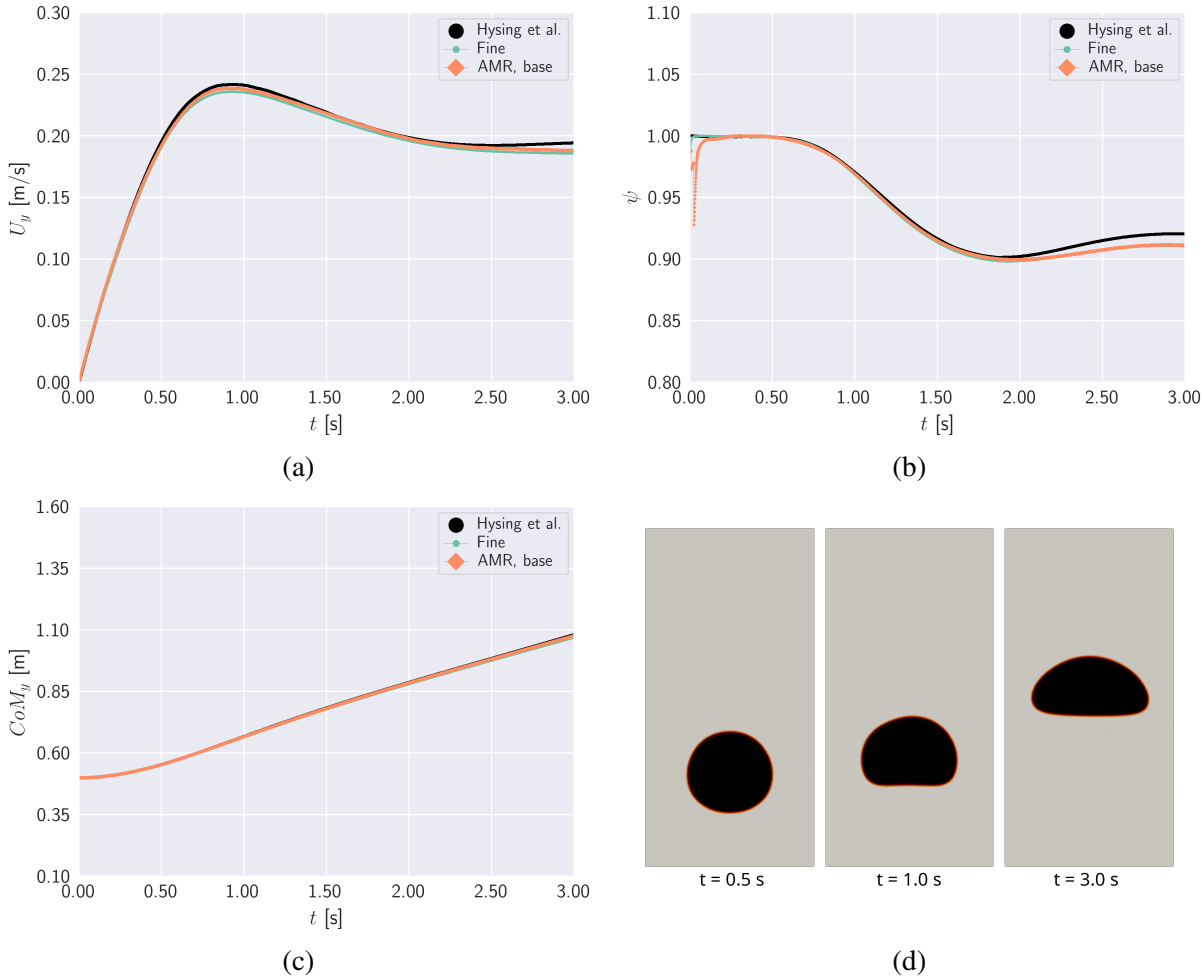


Figure 4.2: Results for the bubble dynamics case obtained using the `multiFieldRefiner2D` class: (a) rise velocity, (b) sphericity, (c) centre of mass, (d) bubble evolution.

4.2.2 Criteria and Validation for 2D Flow Around a Cylinder

The flow around a cylinder test case follows the same general pattern as the previously described case. The overall setup is identical to that outlined in Subsection 2.5.1, except for the inclusion of the `dynamicMeshDict` and coded functions that calculate the refinement criteria. The default

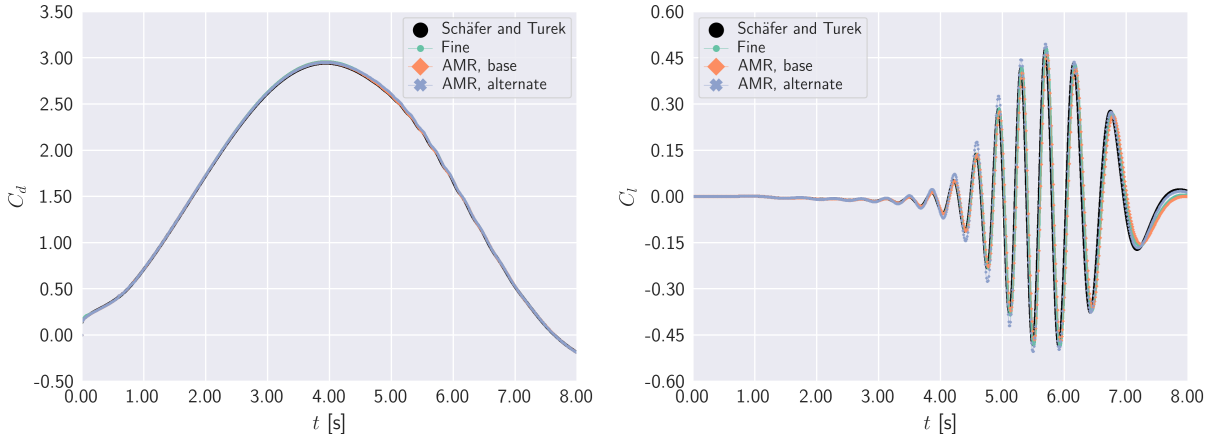
criterion is geometric, restricting refinement to a specified inner region of the domain. The smoothing factor is $f = 5.0$. Two additional criteria are considered - the normalised vorticity, defined as:

$$\phi_1 = \frac{S(\omega)}{\max(S(\omega))} \tag{4.12}$$

where $\omega = \|\nabla \times \mathbf{u}\|$, and the normalised velocity gradient, given by:

$$\phi_2 = \frac{S(\nabla \mathbf{u})}{\max(S(\nabla \mathbf{u}))} \,. \tag{4.13}$$

Two distinct scenarios are evaluated, differing only in their refinement thresholds. The base scenario (denoted base) uses thresholds 0.3 to 1.0 for ϕ_1 and ϕ_2. In the second scenario, denoted alternate, the minimum value is lowered to 0.2. Refinement is performed every 512 time steps. The initial mesh is slightly coarser than the conventional coarse mesh (refinement ratio $r = 1.2$). AMR is permitted to perform up to two levels of refinement.



(a) (b)

Figure 4.3: Drag and lift coefficients for the two-dimensional AMR cylinder case: (a) drag coefficient, (b) lift coefficient.

Based on the results shown in Figure 4.3, it can be concluded that, for the drag coefficient, both the conventional approach and the mcAMR method yield comparable results that align well with the available reference data. No significant deviations are observed. In contrast, several observations can be made regarding the lift coefficient. First, the base AMR case and the results obtained using a fine mesh are in good agreement. Although the AMR results show a slight delay, the difference is minimal. Compared to the base case, the alternate case better captures the overall trend of the reference data. However, between approximately $t = 3.5$ s and $t = 6$ s, the alternate case tends to overpredict the lift coefficient, as evident from the peak

values. Beyond this interval, it settles and aligns with the reference data. As the only practical difference between the cases lies in the mesh resolution, it is the primary reason for the observed differences.

4.3 Application of mcAMR to Three-Dimensional Problems

The application of mcAMR to three-dimensional problems follows the same principles as in the two-dimensional framework. Although the underlying refinement strategy remains the same, the increased complexity of three-dimensional flows introduces additional challenges, such as resolution requirements and computational cost. To assess the effectiveness and robustness of the mcAMR approach in this context, two test cases are considered: flow around a cylinder and breaking of a dam.

4.3.1 Criteria and Validation for 3D Flow Around a Cylinder

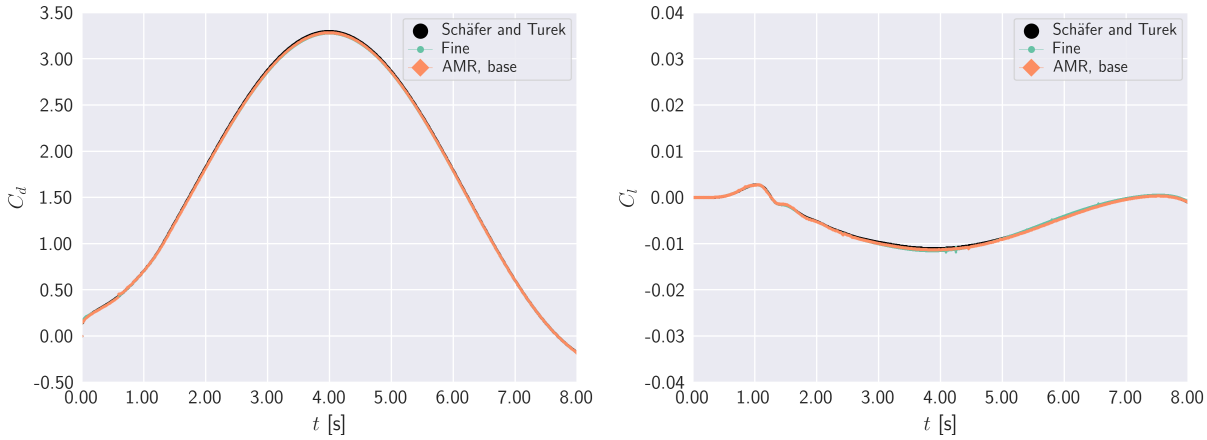
The three-dimensional flow around a cylinder case using mcAMR is largely analogous to the two-dimensional case. The thresholds for the refinement criteria are set to 0.3 for the lower and 1.0 for the upper limit. A default geometric constraint is applied and combined with the normalised vorticity and the normalised velocity gradient:

$$\phi_1 = \frac{S(\omega)}{\max(S(\omega))}, \quad \phi_2 = \frac{S(\nabla \mathbf{u})}{\max(S(\nabla \mathbf{u}))} \, . \tag{4.14}$$

A smoothing factor $f = 2.0$ is used. Refinement is performed every 512 time steps. As in the two-dimensional case, the initial mesh is slightly coarser than the conventional coarse mesh ($r = 1.2$). AMR is permitted to perform up to two levels of refinement.

Based on the results presented in Figure 4.4, the conclusion is straightforward. The conventional grid-based approach and the mcAMR-driven strategy provide comparable results that align well with the reference data, with no notable discrepancies. While one might argue that the conventional approach is preferable due to its simplicity, it is important to note that the mcAMR case completed the simulation in approximately 18 % less computational time. This efficiency gain can be further improved by adjusting the refinement frequency. Moreover, results with slightly reduced accuracy but requiring only around 50 % of the computational time can be achieved by relaxing the refinement criteria, effectively trading accuracy for efficiency.

Nonetheless, the core concept holds: it is possible to obtain similarly accurate results with a reduced computational cost.



(a) (b)

Figure 4.4: Drag and lift coefficients for the three-dimensional AMR cylinder case: (a) drag coefficient, (b) lift coefficient.

4.3.2 Criteria and Validation for 3D Breaking of a Dam

The conventional dam break test case employs subtle mesh grading, making direct grid comparison difficult. Nevertheless, on average, the initial cell size in the mcAMR case is approximately 2.7 times larger than that of the coarse mesh. Due to numerical instabilities, the setups across test cases were not identical. The mcAMR case used a first-order accurate temporal discretisation scheme and the MPLIC scheme for the volume fraction. The built-in utility `interfaceHeight`, used to measure water level, produced oscillatory or discontinuous results under AMR, which did not reflect the actual interface. To address this, a coded function was implemented to compute the interface height based on local and neighbouring cell values.

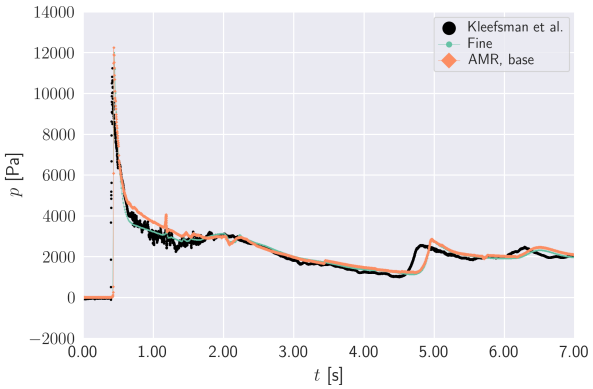
Mesh adaptation was performed every four time steps. Two refinement criteria were used: the value of the scalar field `alpha.water` and the normalised gradient of the `alpha.water` field:

$$\phi_0 = \alpha, \quad \phi_1 = \frac{S(|\nabla\alpha|)}{\max\left(S(|\nabla\alpha|)\right)} \tag{4.15}$$

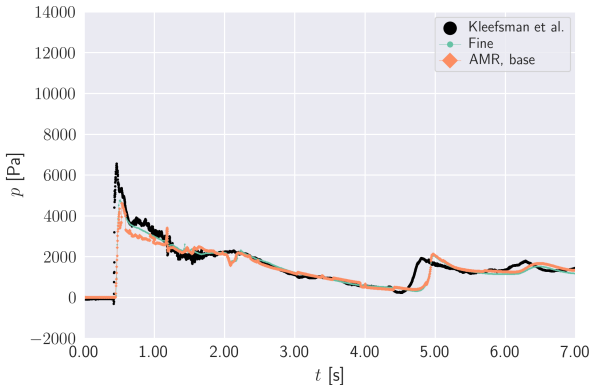
with smoothing factor $f = 5.0$. The cells were marked for refinement if ϕ_0 was within the range of 0.001 to 0.999 or if ϕ_1 exceeded 0.1. Two levels of refinement were allowed.

Since there are 12 measurement locations for the experimental data, this discussion will focus on two representative locations for pressure and two for interface height. Figure 4.5

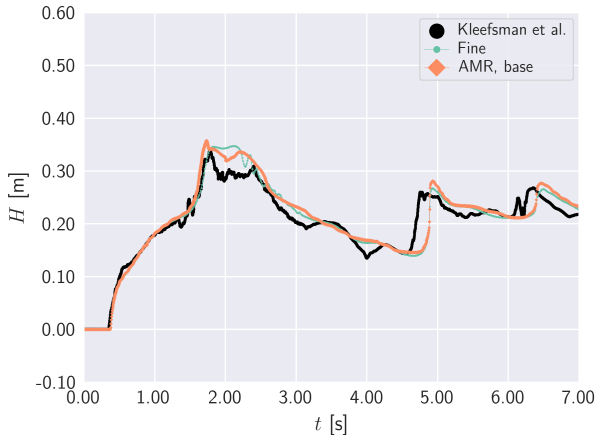
displays these results. At point P1, the mcAMR case shows a sharp pressure spike similar to the conventional approach. The values then decrease and, after approximately $t = 1.5$ s, closely match those from the fine grid. Around $t \approx 0.75$ s, a noticeable deviation appears between the two approaches. The mcAMR case slightly overpredicts the experimental data, while the conventional approach first underpredicts and then overpredicts. The results at point P2 follow a similar trend, with the main difference also occurring near $t \approx 0.75$ s, where the mcAMR case underpredicts. This deviation may be related to inaccuracies in probing on a dynamically changing grid. For interface height at point H2, the mcAMR case overpredicts the fine grid result from $t = 1.5$ s onwards, though the values remain close. At point H4, both approaches give unsatisfactory results, but the AMR case performs slightly better in capturing the transition around $t \approx 2.75$ s.



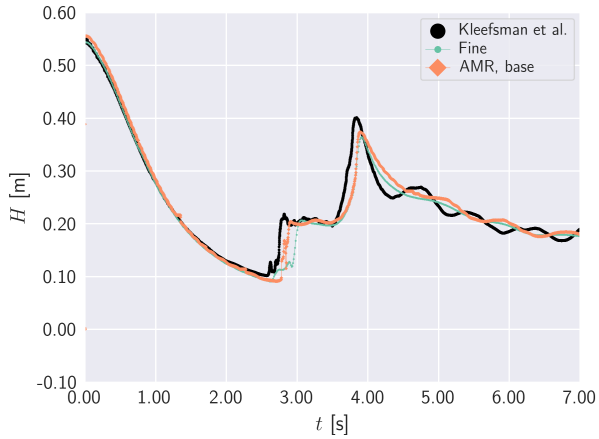
(a) Pressure at point P1.



(b) Pressure at point P3.



(c) Water height at point H2.



(d) Water height at point H4.

Figure 4.5: Selected results for the dam break test case using the AMR approach.

Importantly, the mcAMR case is 77 % faster than the simulation on the fine grid and 24 % faster than the medium grid case. Furthermore, a comparison using the dynamic time warping (DTW) algorithm across all 12 measurement points shows that the mcAMR approach is, on average, only 2.23 % less accurate. This includes both pressure and interface height data. In fact, the AMR solution performs slightly better for pressure, with a 0.5 % improvement in accuracy on average.

5 LOAD-AWARE DYNAMIC LOAD BALANCING

One of the major challenges associated with adaptive mesh refinement is maintaining efficiency on massively parallel systems. As AMR refines the mesh, certain subdomains of the decomposed problem inevitably experience a significant increase in computational load. This is primarily due to the growing number of cells, which increases the computational cost of solving the associated linear systems. As a result, dynamic load balancing has become essential for maintaining consistent performance across all MPI ranks. OpenFOAM 10 includes a basic load balancing mechanism through the `distribution` class, which calculates the number of cells per rank and governs redistribution accordingly. In this work, two new classes are introduced, `distributorMPI` and `distributorRollingMPI`, which assess MPI rank loads directly to determine imbalance and manage redistribution.

5.1 MPI-Based Load Redistribution

The native load balancing implementation, `distributor`, employs a simple geometric strategy. Its objective is to distribute the computational load evenly across processors by ensuring that each MPI rank handles approximately the same number of cells. At each interval, the total number of cells is divided by the number of processors to compute the ideal cell count per rank. The number of cells per rank is then compared to this ideal value, and the imbalance is quantified as the maximum relative deviation across ranks. If this imbalance exceeds a predefined threshold, `maxImbalance`, a redistribution is triggered using a decomposition method specified in the `decomposeParDict`. While straightforward and efficient, this method assumes a uniform computational cost per cell and homogeneity in compute performance.

The proposed implementation introduces a performance-aware load balancing strategy to address the limitations of the cell-based method. Instead of assuming uniform computational

loads for all cells and identical behaviour across ranks, this approach uses direct runtime measurements to assess the workload and memory footprint of each MPI rank. The new implementation, `distributorMPI`, relies on modifications to the communication layer in `UPstream.C`. Redistribution is triggered based on two criteria: CPU load imbalance and memory usage imbalance. A simplified flowchart of `distributorMPI` is shown in Figure 5.1.

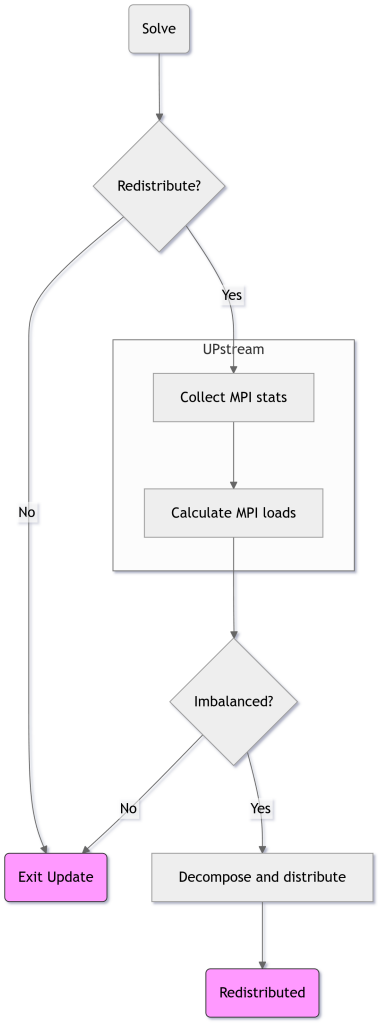


Figure 5.1: Simplified flowchart of the `distributorMPI` class logic.

The `UPstream.C` file was extended with a new function, `getMPIStats()`, to measure two runtime metrics for each MPI rank. The first metric, CPU load, is estimated by measuring the time spent waiting in blocking MPI communication calls, such as `MPI_Allreduce`, `MPI_Alltoall`, and `MPI_Waitall`. Each call is timed to compute the total waiting duration, and the CPU load is inferred as the proportion of time spent actively computing (i.e. not waiting) relative to the total elapsed time. The second metric is memory usage, which provides the

current memory consumption of the MPI rank. The `getMPIStats()` function collects these metrics locally and aggregates them across all ranks, returning a `scalarList` containing the CPU and memory loads for the entire domain.

The returned metrics are used within the `update` method of the `distributorMPI` class. After collecting the data, the average CPU and memory loads across all ranks are computed. The imbalance is then evaluated as the maximum relative deviation of each rank's CPU and memory load from the corresponding average, calculated as:

$$CPU = \left| \frac{T_i - \frac{1}{N}\sum_{i=1}^{N} T_i}{\frac{1}{N}\sum_{i=1}^{N} T_i} \right|, \quad MEM = \left| \frac{M_i - \frac{1}{N}\sum_{i=1}^{N} M_i}{\frac{1}{N}\sum_{i=1}^{N} M_i} \right| \tag{5.1}$$

where T_i is the effective compute time and M_i is the memory load for rank i. A redistribution is triggered if these imbalances exceed their respective thresholds, `maxImbalance` for CPU and `maxMemImbalance` for memory.

The distributor call and imbalance criteria are defined in the `dynamicMeshDict`. Furthermore, the distribution algorithm for the distributor must be specified in the `decomposeParDict`. An example of the configuration definition is shown in Code 5.1.

Code 5.1: Segment of the `dynamicMeshDict` configuration file defining distributor parameters.

```
distributor
{
    type                        distributorMPI;
    libs                        ("libfvMeshDistributorsMPI.so");
    redistributionInterval      1;
    maxImbalance                0.1;
    maxMemImbalance             0.1;
}
```

5.2 Archive-Based Load Redistribution

The `distributorRollingMPI` class extends `distributorMPI` by introducing temporal filtering to suppress reactions to transient imbalances. It maintains rolling histories of CPU and memory imbalance using circular buffers sized according to `redistributionInterval`. Instead of acting on a single measurement, it tracks imbalance values over multiple update steps and evaluates their persistence.

A new parameter, `maxImbalancedStates`, specifies the minimum number of historical entries that must exceed the defined thresholds to trigger redistribution. Redistribution is initiated

only when the buffer is full and this condition is met. This approach reduces the sensitivity to short-lived fluctuations and helps avoid unnecessary redistributions. Table 5.1 summarises the key implementation differences between `distributorMPI` and `distributorRollingMPI`.

Table 5.1: Implementation differences between `distributorMPI` and `distributorRollingMPI`.

Feature	`distributorMPI`	`distributorRollingMPI`
Decision basis	Instantaneous imbalance	Rolling history of imbalances
Reaction time	Immediate	Delayed for robustness
MPI sensitivity	High	Resistant to transient behaviour
Historical data	None	Circular buffer
Key parameters	`maxImbalance`, `maxMemImbalance`	`maxImbalance`, `maxMemImbalance`, `maxImbalancedStates`

5.3 Computational Efficiency

The computational efficiency of the implemented load balancing classes, along with the native `distributor`, is evaluated using the dam break test case (Subsection 2.5.3). The overall adaptive mesh refinement configuration follows the mcAMR setup outlined in Subsection 4.3.2. While the selected scenario does not address the full range of possible flow regimes or problem types, it provides a controlled and representative baseline for comparing the performance characteristics of the available distribution strategies.

The initial assessment was conducted using the Score-P 8.4 profiling infrastructure [55]. The simulations were run in a distributed environment across two computational nodes. Each node was equipped with two Intel Xeon E5-2690 v3 processors with hyperthreading disabled, resulting in 48 cores. One MPI rank was assigned per core, yielding 48 MPI ranks in total. The nodes were connected via an InfiniBand FDR interconnect with a maximum throughput of 6.31 GB/s. The Intel MPI Library 2021.5 was used as the underlying MPI runtime. Unless otherwise stated, the distributor parameters were configured as follows: `refineInterval = 1`, `redistributionInterval = 16`, `maxImbalance = 0.25`, and `maxImbalancedStates = 5`. An overview of the performance results is given in Table 5.2.

Based on the results, all three distributors provide comparable parallel efficiency, with `distributorMPI` showing a marginal advantage. Communication efficiency is highest for `distributorRollingMPI`, indicating improved resilience to communication overhead. Both

Table 5.2: Performance statistics obtained using Score-P 8.4 for different distribution strategies.

Metric	`distributor`	`distributorMPI`	`distributorRollingMPI`
Parallel efficiency	0.546	0.556	0.556
Load balance efficiency	0.795	0.792	0.776
Communication efficiency	0.687	0.702	0.716
Simulation time [ms]	41299	40738	41153
Total `distribute` [ms]	24932	59008	61584
Total time, $n \cdot CPU$ [ms]	1982453	1955717	1979775

MPI-based strategies incur a longer time in the `distribute` phase, especially `distributorRollingMPI`, due to their more complex decision logic and more frequent redistributions. Despite these differences, the overall runtime remains similar.

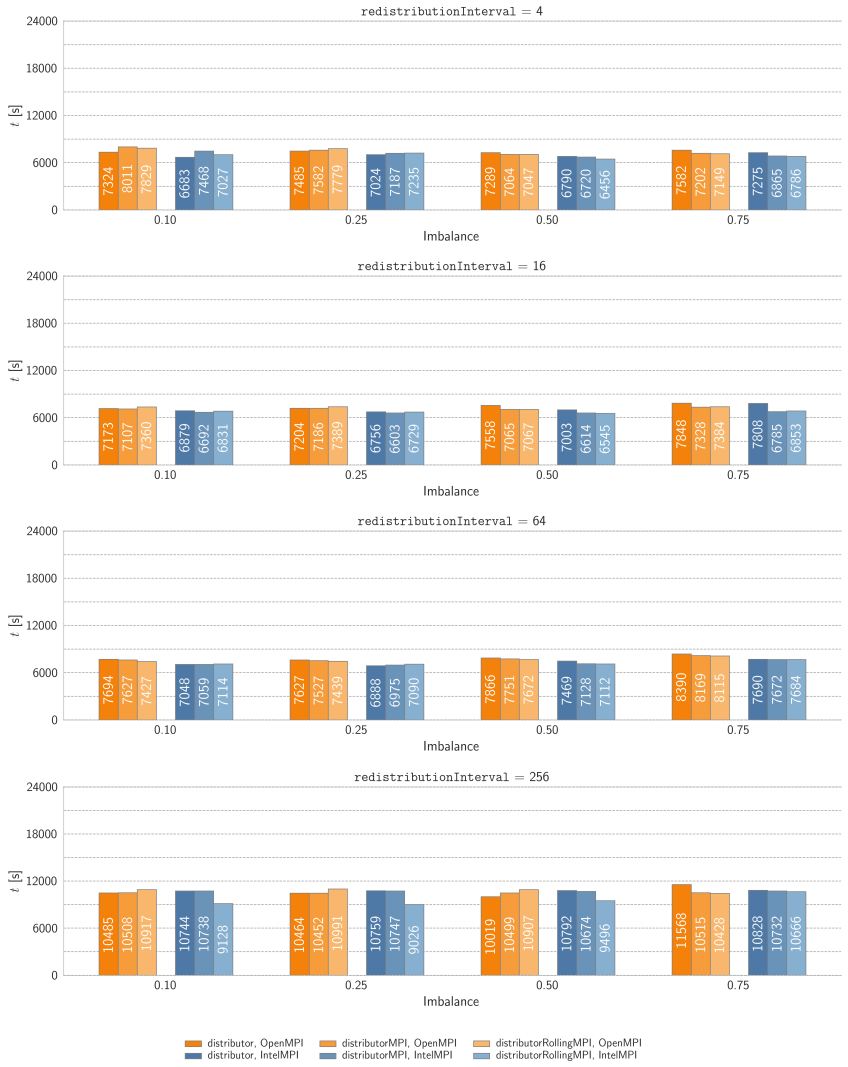


Figure 5.2: Impact of imbalance threshold and redistribution interval on simulation time for different distributor classes using Intel MPI and OpenMPI.

Further assessment was conducted on the same hardware platform, using Intel MPI Library 2021.5 and OpenMPI 4.1.1. All three distributor classes were tested across a range of parameter combinations, specifically imbalance thresholds $\{0.10,\ 0.25,\ 0.50,\ 0.75\}$ and redistribution intervals $\{4,\ 16,\ 64,\ 256\}$. The corresponding results are presented in Figure 5.2.

Several trends can be observed. First, except for the `redistributionInterval` = 256 case, simulations are typically completed faster with Intel MPI. Second, performance depends on the imbalance threshold. As `maxImbalance` increases, `distributorRollingMPI` outperforms the others. Conversely, for lower thresholds, its performance drops relative to the simpler strategies, suggesting it is less efficient when reacting to frequent, minor imbalances. The best result was observed for `distributorRollingMPI` with `redistributionInterval` = 4 and `maxImbalance` = 0.5, which led to the fastest overall simulation time.

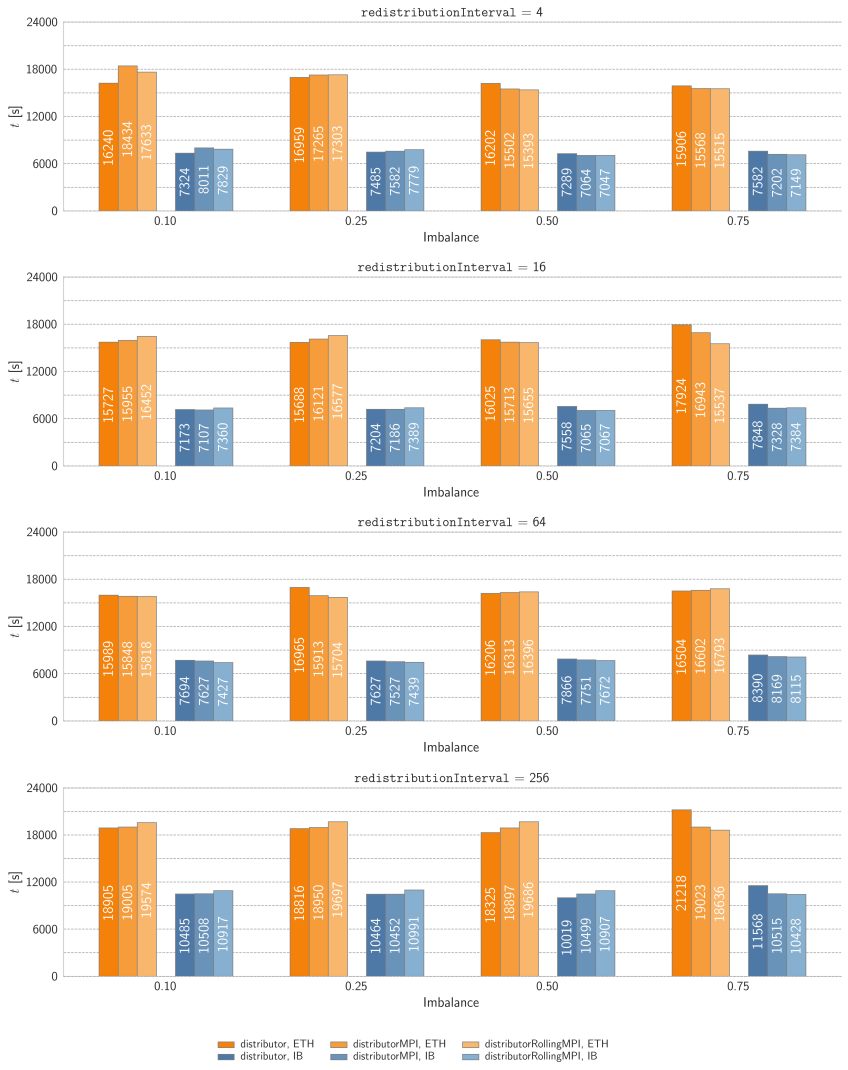


Figure 5.3: Impact of network interconnect on the performance of distributor classes across various imbalance thresholds and redistribution intervals.

To assess the impact of the network interconnect, additional tests were performed using the same hardware, parameters, and OpenMPI 4.1.1. In addition to the InfiniBand FDR baseline setup, a standard Ethernet connection with a throughput of approximately 0.12 GB/s was considered. The results are shown in Figure 5.3.

As expected, simulations using Ethernet were significantly slower than those using InfiniBand, though the difference was less pronounced than the bandwidth gap would suggest. This indicates that overall communication demands are relatively low. Despite the noticeable difference in computational times (computational times were approximately twice as long), general trends remained similar, with a few exceptions. The best result was observed for `distributorRollingMPI`, with `redistributionInterval` = 4 and `maxImbalance` = 0.5.

Behaviour across different hardware environments was also evaluated by comparing the results from the cluster against two unified, non-network-limited machines. The first, denoted Intel, is a high-capacity computational node comprised of 16 Intel E7-8867 v3 CPUs, each with 16 physical cores and 32 threads (hyperthreading disabled). The second, denoted AMD, features 2 AMD Epyc 7662 CPUs, each with 64 physical cores and 128 threads (hyperthreading disabled). All simulations were run using 48 physical cores mapped one-to-one to 48 MPI ranks to ensure consistency. Both systems were provisioned with sufficient memory to avoid memory-related bottlenecks.

The results from the cluster using Intel MPI, with `redistributionInterval` = 4 and `maxImbalance` = 0.5, are used as a reference. As in previous assessments, the following parameter combinations were considered: imbalance thresholds $\{0.10,\ 0.25,\ 0.50,\ 0.75\}$ and redistribution intervals $\{4,\ 16,\ 64,\ 256\}$. The results are presented in Figure 5.4.

Several observations can be made based on the results shown in Figure 5.4. As anticipated, newer hardware yields better performance, with the lowest computational times observed on the most modern platform. More importantly, on the newer system, the newly implemented classes, `distributorMPI` and `distributorRollingMPI`, consistently outperform the native `distributor`, except in configurations where `redistributionInterval` = 4 and `maxImbalance` < 0.5. A single outlier is detected at `redistributionInterval` = 64 and `maxImbalance` = 0.1.

These results lead to the following conclusions. First, newly implemented classes generally

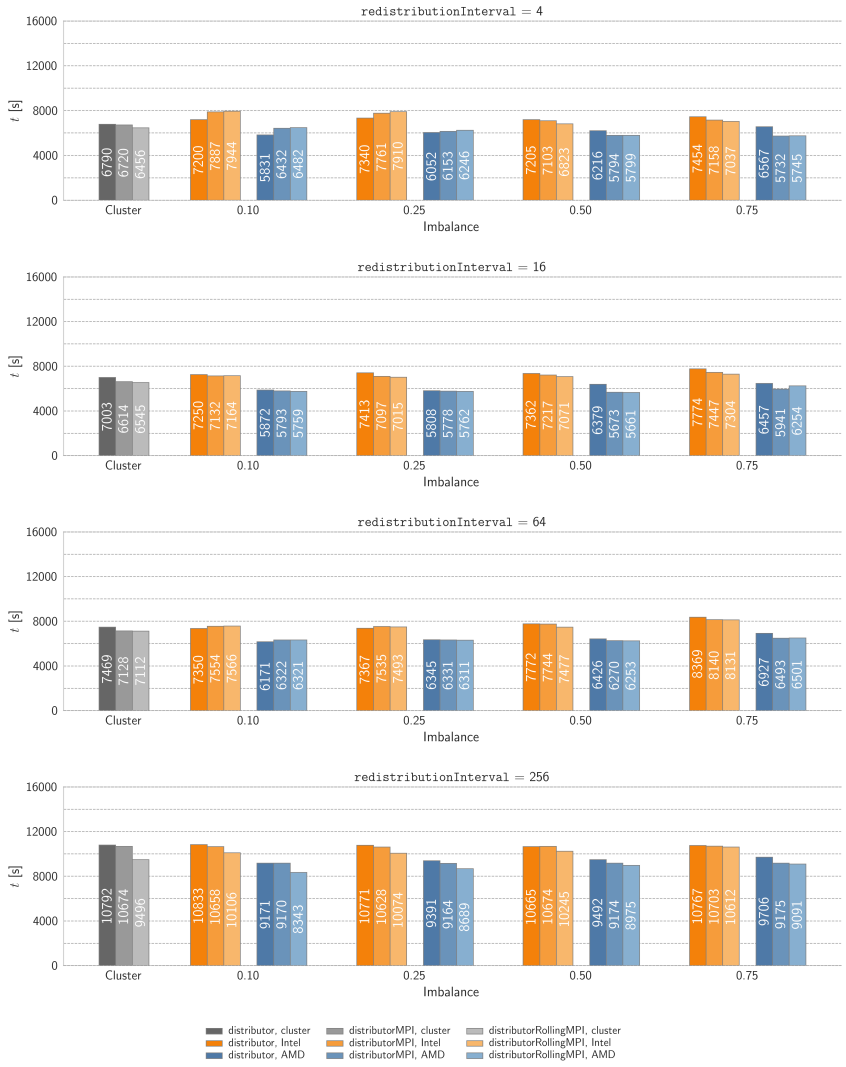


Figure 5.4: Performance of distributor classes across three hardware environments for varying imbalance thresholds and redistribution intervals.

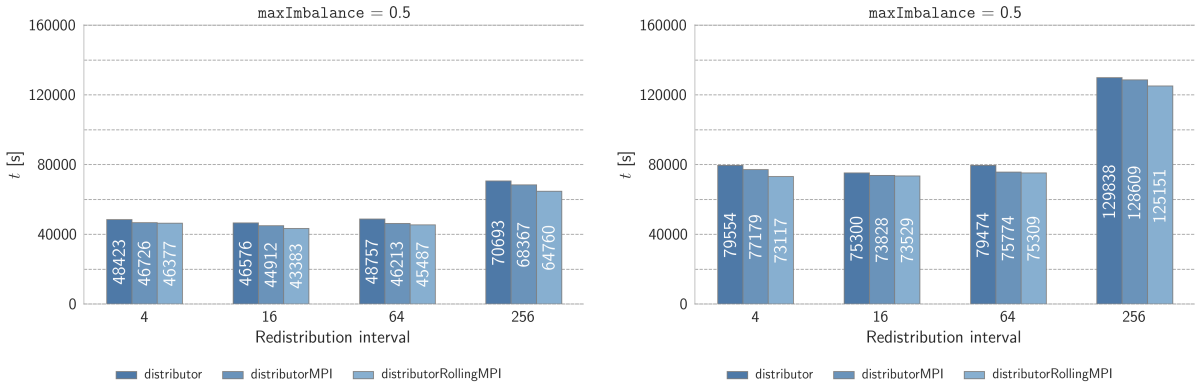
perform better when `maxImbalance` is set to higher values (e.g. 0.5). Second, their performance improves significantly when the redistribution interval is less frequent. This is particularly relevant for realistic three-dimensional problems, where frequent redistributions are typically avoided, as they tend to induce numerical instabilities and increase overhead, which can significantly hinder overall computational efficiency.

The dam break test considered so far is relatively simple, with a moderate mesh size. On average, the resulting mesh at $t = 7$ s contained approximately $3 \cdot 10^5$ cells. This number is relatively low and is not representative of more demanding scenarios. Therefore, a new set of test cases has been defined to address this, maintaining the same overall problem setup but utilising more refined initial meshes.

The first case, denoted medium, begins with a significantly finer grid than the one used

in the baseline case. The initial cell size is reduced by a factor $r_m = 1.8$. The second case, denoted fine, starts with an even more refined grid, with the cell size further reduced by a factor $r_f = 1.333$ relative to the medium case. These changes affect the intermediate and final sizes of the computational mesh and thus create more demanding scenarios for evaluation.

For both cases, the imbalance threshold was set to 0.5. Testing was performed for four different redistribution intervals $\{4,\ 16,\ 64,\ 256\}$. The tests were conducted on the previously described cluster, with five computational nodes and 24 cores (resulting in 120 MPI ranks) for the medium case and ten computational nodes and 24 cores (resulting in 240 MPI ranks) for the fine case. The resulting grid sizes were approximately $2.2 \cdot 10^6$ and $4.2 \cdot 10^6$ for the medium and fine cases, respectively. The simulation times for these cases are presented in Figure 5.5.



(a) (b)

Figure 5.5: Distributor performance for scaled dam break test cases: (a) medium case, (b) fine case.

The measured computational times for the more complex test cases reveal a distinct trend. For the medium case, the newly implemented classes, `distributorMPI` and `distributorRollingMPI`, show significant performance improvements despite the more frequent redistributions. The optimal performance is achieved for a redistribution interval of 16. Similarly, the behaviour follows the same pattern for the fine case, with the same optimal redistribution interval observed. Interestingly, distribution across multiple nodes and the resulting network traffic do not appear to affect the scaling behaviour significantly.

6 REFINEMENT CRITERION FOR LARGE EDDY SIMULATION

Large eddy simulation provides notably higher fidelity than conventional RANS approaches by resolving the large, energy-containing turbulent structures. However, the high spatial and temporal resolution requirements, combined with its sensitivity to numerical artefacts, make LES particularly challenging to implement on dynamically evolving computational grids.

Adaptive mesh refinement introduces frequent grid changes, which can affect the turbulence content in sensitive regions, compromising the accuracy and stability of the simulation. To effectively integrate LES with AMR, the refinement process must be minimally intrusive. Grid changes should occur gradually, with sufficient temporal and spatial buffering, to prevent artificial damping or elimination of turbulent structures. This implies a more conservative refinement strategy compared to typical AMR applications.

6.1 Composite Refinement Criterion

A critical component of any AMR strategy is the refinement criterion used to determine which regions of the domain require increased or reduced resolution. In the context of LES, these criteria are often based on physically meaningful flow features such as vorticity, strain rate, or turbulent kinetic energy.

A single refinement criterion can fail to capture the complexity of turbulent flows, particularly in dynamically evolving domains. Since different flow features describe distinct phenomena, focusing on one feature risks neglecting other aspects. Despite this, many individual criteria are commonly employed, and therefore, a brief overview of the most widely used ones is provided.

6.1.1 Established Refinement Criteria

The pressure gradient ∇p can be used to determine the direction and magnitude of the maximum rate of spatial pressure variation. It drives flow and is particularly significant in boundary layers,

flow separation regions, and near shock waves, where large pressure gradients often coincide with significant changes in flow behaviour.

Vorticity captures the local rotational motion of fluid and is commonly used to detect vortices and regions of high shear:

$$\omega = \nabla \times \mathbf{u} \tag{6.1}$$

where $\mathbf{u}$ is the velocity field. High vorticity typically implies coherent structures or instabilities.

The strain rate tensor describes the rate at which the fluid deforms due to velocity gradients. Regions with high strain are typically associated with shear layers and near-wall turbulence. The strain rate tensor is symmetric and defined as:

$$\mathbf{S} = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right) \ . \tag{6.2}$$

Turbulent kinetic energy quantifies the kinetic energy contained in velocity fluctuations and thus reflects the local turbulence intensity, that is, it can be used to determine regions of the flow where turbulence is most intense:

$$k = \frac{1}{2}\langle \mathbf{u}' \cdot \mathbf{u}' \rangle \tag{6.3}$$

where $\mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle$ is the fluctuating component of the velocity.

The dissipation rate indicates how quickly turbulent kinetic energy is converted into internal energy by viscous action and can thus be considered an indicator of fine-scale turbulence. It can be defined in terms of the strain rate tensor:

$$\varepsilon = 2\nu\langle \mathbf{S}' : \mathbf{S}' \rangle \approx 2\nu\langle \mathbf{S} : \mathbf{S} \rangle \tag{6.4}$$

where ν is the kinematic viscosity, $\mathbf{S}'$ is the strain rate tensor of the fluctuating velocity field, and $\mathbf{S} : \mathbf{S}$ is the second invariant of the resolved strain rate tensor.

Wall shear stress is important for near-wall turbulence, i.e. in boundary layers. It quantifies the tangential frictional force exerted by the fluid on the wall, which directly influences the momentum transfer and the intensity of the turbulence near the surface. The magnitude of wall shear stress is given by:

$$\tau_w = \mu \left(\frac{\partial u_\tau}{\partial y} \right)_{y=0} \tag{6.5}$$

where u_τ is the velocity component tangential to the wall, y is the wall-normal coordinate, and μ is the dynamic viscosity.

A derived quantity, Q criterion, identifies vortical regions by measuring the balance between rotational and strain components in the flow:

$$Q = \frac{1}{2}\left(|\Omega|^2 - |\mathbf{S}|^2\right) \tag{6.6}$$

where Ω is the rotation rate tensor, defined as:

$$\Omega = \frac{1}{2}\left(\nabla\mathbf{u} - (\nabla\mathbf{u})^T\right) \ . \tag{6.7}$$

Regions where $Q > 0$ are identified as vortex dominated. The Q criterion is widely used for vortex detection, but may detect false positives in regions with high shear strain.

The λ_2 criterion is a more selective indicator of vortical structures. It is defined based on the eigenvalues of the symmetric tensor $\mathbf{S}^2 + \Omega^2$. A region is considered part of a vortex core if the second-largest eigenvalue, λ_2, of this tensor satisfies:

$$\lambda_2 = \mathrm{eig}_2\left(\mathbf{S}^2 + \Omega^2\right) < 0 \ . \tag{6.8}$$

The λ_2 criterion is more robust in detecting true vortices, especially in shear-dominated flows.

6.1.2 Practical Considerations

While each noted criterion effectively identifies specific aspects of the flow, turbulent phenomena rarely occur in isolation. Given the complexity and variability of turbulent flows, a single metric cannot, therefore, capture the full range of relevant features. To ensure that adaptive mesh refinement modifies the grid appropriately while balancing computational cost, a composite approach is proposed.

The composite approach offers several advantages: it increases robustness by reducing dependence on any single flow metric, improves adaptability across different flow regimes, and allows for user control or automatic tuning based on problem-specific requirements.

An effective starting point is the λ_2 criterion, which identifies coherent vortical structures by isolating regions where rotation dominates over strain. A particularly effective pairing is the λ_2 criterion and the magnitude of the strain rate tensor $\sqrt{\mathbf{S} : \mathbf{S}}$. This enables the detection of both

rotational features and regions of intense shear, such as boundary layers and free shear layers. Alternatively, the turbulent kinetic energy k can be used. While λ_2 captures coherent vortices, the turbulent kinetic energy provides a broader measure of turbulence intensity. Combining λ_2 with the vorticity is also feasible, however, the strong correlation between these quantities can lead to redundancy.

A practical AMR refinement criterion should also consider additional factors. Using multiple refinement criteria can improve accuracy, but it also adds computational overhead, both from evaluating the criteria and handling the refined mesh. Furthermore, refinement based solely on flow features may be insufficient if the mesh resolution is inadequate i.e. does not meet the numerical requirements for the simulation. It is, therefore, essential to incorporate a mesh-based criterion. A mesh-based criterion could be used to evaluate the local cell size Δ against a target resolution, with refinement triggered when $\Delta > \Delta_{\text{target}}$.

Based on these considerations, a baseline multi-criteria refinement strategy can be formulated. It combines the λ_2 criterion with a mesh-based filter Δ. This pairing balances computational efficiency with effective refinement. Additional flow-based criteria, such as the strain rate magnitude, may be included if greater sensitivity is required, although this must be weighed against the increased computational cost. While mesh-based criterion alone may be sufficient in some cases, combining it with targeted flow criterion improves robustness and adaptability.

6.1.3 Formulation of the Criterion

The proposed formulation combines two scalar fields: a binary field $\phi_0 := \Phi_{\lambda_2}$, which indicates the presence of coherent vortical structures, and a normalised measure of the local mesh resolution relative to turbulent scales, $\phi_1 := \Phi_\Delta$. Each field is associated with a threshold and a logical operation that determines how its contribution is combined with the current set of candidate cells. Importantly, the logic is applied sequentially.

Fields are computed on the fly using runtime functions, from the instantaneous velocity field $\mathbf{u}$, viscosity fields, and mesh geometry. To limit computational overhead, the fields are updated periodically but infrequently, typically just before mesh refinement events.

In order to calculate the λ_2 criterion, velocity gradient tensor $\nabla\mathbf{u}$ must be decomposed into symmetric $\mathbf{S}$ and antisymmetric Ω parts:

$$\mathbf{S} = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right), \quad \Omega = \frac{1}{2}\left(\nabla\mathbf{u} - (\nabla\mathbf{u})^T\right) . \tag{6.9}$$

The tensor $\mathbf{M}$ is calculated by summing the matrix products $\mathbf{S}^2$ and Ω^2:

$$\mathbf{M} = \mathbf{S}^2 + \Omega^2 \ . \tag{6.10}$$

The eigenvalues of the symmetric tensor $\mathbf{M}$ are computed by solving the characteristic polynomial of the 3×3 matrix:

$$\det(\mathbf{M} - \lambda \mathbf{I}) = -\lambda^3 + I_1 \lambda^2 - I_2 \lambda + I_3 = 0 \tag{6.11}$$

where λ denotes an eigenvalue, $\mathbf{I}$ is the identity matrix, and I_1, I_2, and I_3 are the principal invariants of $\mathbf{M}$, defined as:

$$I_1 = \mathrm{tr}(\mathbf{M}), \tag{6.12}$$

$$I_2 = \frac{1}{2}\left[(\mathrm{tr}(\mathbf{M}))^2 - \mathrm{tr}(\mathbf{M}^2)\right], \tag{6.13}$$

$$I_3 = \det(\mathbf{M}) \ . \tag{6.14}$$

Solving this cubic polynomial yields the three eigenvalues of $\mathbf{M}$, from which the second-largest eigenvalue λ_2 is extracted. The smoothed version of $-\lambda_2$ is computed using the smoothing operator S:

$$\widetilde{\lambda_2} = S(-\lambda_2) \ . \tag{6.15}$$

Subsequently, an isosurface of $\widetilde{\lambda_2}$ is constructed for a user-specified threshold (e.g. $\widetilde{\lambda_2}_{\mathrm{iso}} = 1.0$). Let Ω_{iso} denote the set of all cells c in the computational domain Ω intersected by this isosurface. Cells in Ω_{iso} are marked with a binary flag, and the field Φ_{λ_2} is derived as follows:

$$\Phi_{\lambda_2}(c) = \begin{cases} 1, & \text{if } c \in \Omega_{\mathrm{iso}} \\ 0, & \text{if } c \notin \Omega_{\mathrm{iso}} \end{cases} . \tag{6.16}$$

The quantity Φ_Δ characterises how well the local grid resolves the relevant turbulent structures by comparing the local cell size Δ to the Taylor microscale λ_T, a characteristic turbulence length scale. High values of Φ_Δ indicate that the local mesh is too coarse relative to the turbulence length scales. The cell size is approximated by the cube root of the cell volume:

$$\Delta = V^{1/3} \ . \tag{6.17}$$

The Taylor microscale is estimated as:

$$\lambda_T = \sqrt{\frac{15(\nu + \nu_{\text{SGS}})k}{\varepsilon}} \tag{6.18}$$

where ν is the molecular viscosity, ν_{SGS} is the subgrid-scale viscosity, and k is turbulent kinetic energy. The dissipation rate ε is approximated by:

$$\varepsilon = 2(\nu + \nu_{\text{SGS}})(\mathbf{S} : \mathbf{S}) \tag{6.19}$$

with $\mathbf{S}$ denoting the rate-of-strain tensor. The normalised cell size field is obtained by computing the ratio Δ/λ_T and applying a spatial smoothing operator S to reduce numerical noise:

$$\Phi_\Delta = S\left(\frac{\Delta}{\lambda_T}\right) . \tag{6.20}$$

Finally, the derived fields Φ_{λ_2} and Φ_Δ can be combined to formulate a multi-criteria refinement condition:

$$\mathscr{R}_{\lambda_2} = \left\{ c \in \Omega \;\middle|\; \Phi_{\lambda_2}(c) = 1 \right\}, \tag{6.21}$$

$$\mathscr{R}_\Delta = \left\{ c \in \Omega \mid \Phi_\Delta(c) \geq \delta_\Delta \right\}, \tag{6.22}$$

$$\mathscr{U}_{\lambda_2} = \left\{ c \in \Omega \;\middle|\; \Phi_{\lambda_2}(c) < 1 \right\}, \tag{6.23}$$

$$\mathscr{U}_\Delta = \left\{ c \in \Omega \mid \Phi_\Delta(c) < \delta_\Delta \right\} . \tag{6.24}$$

The final refinement and unrefinement sets are obtained by taking the union of the respective field-based criteria:

$$\mathscr{R} = \mathscr{R}_{\lambda_2} \cup \mathscr{R}_\Delta, \quad \mathscr{U} = \mathscr{U}_{\lambda_2} \cup \mathscr{U}_\Delta . \tag{6.25}$$

6.2 Application of mcAMR to LES

The criterion presented in the previous section was applied to three LES-based test cases using `multiFieldRefiner3D`. The results of these test cases will be discussed in the following subsections. Each test case has distinct characteristics. The general setup, including results, using the conventional grid generation approach, has been presented in Section 2.6.

6.2.1 Assessment for Turbulent Channel Flow

The effectiveness of multi-criteria adaptive mesh refinement was evaluated for a turbulent channel flow at a friction Reynolds number of $Re_\tau \approx 395$. The simulation was initialised on a coarse hexahedral mesh of approximately $4 \cdot 10^5$ cells. The initial grid resolution in dimensionless cell units was $\Delta x^+ \approx 98$ in the streamwise direction, $\Delta y^+ \approx 3.6$ near the wall and increasing up to $\Delta y^+ \approx 75$ in the channel core, and $\Delta z^+ \approx 25$ in the spanwise direction. These spacings are coarser than those employed for conventional LES cases, particularly in the near-wall region.

Two subgrid-scale models were considered: the standard Smagorinsky model and the dynamic Smagorinsky model. Mesh refinement was driven by a composite criterion based on normalised Φ_{λ_2} and Φ_Δ. For Φ_{λ_2}, an isovalue threshold of $\widetilde{\lambda_2}_{\text{iso}} = 10.0$ was considered. The lower bound for Φ_Δ was set to 2.0. To avoid numerical artefacts, a smoothing function S with a smoothing factor $f = 2.0$ was applied.

The adaptation strategy allowed up to two levels of refinement, with a two-cell buffer around the refined regions to ensure smooth transitions. A global limit of $3 \cdot 10^6$ cells was imposed to maintain comparability with the LES cases on the conventional grid. Mesh refinement was triggered every 128 time steps, which corresponds to a physical interval of 0.128 s per refinement cycle, given the constant time step size of $\Delta t = 0.001$ s. Over the course of the simulation, this has led to approximately 1900 adaptation steps.

With the Smagorinsky model, the mesh evolved to the specified upper limit of $3 \cdot 10^6$ cells. The highest resolution was achieved in the near-wall regions. The lowest dimensionless cell spacings were $\Delta x^+ \approx 49$, $\Delta y^+ \approx 1.8$, and $\Delta z^+ \approx 12$. In the case of the dynamic Smagorinsky model, the refinement pattern exhibited slight variations due to local differences in the computed eddy viscosity, which enters the refinement criterion via the Taylor microscale. This resulted in a final mesh that contained approximately $3.2 \cdot 10^6$ cells, with similar minimum cell spacings.

A comparative assessment of the results obtained using the AMR approach and conventional LES is provided in Figure 6.1 and Figure 6.2. Results are reported for both Smagorinsky and dynamic Smagorinsky models using conventional and adaptive meshes.

The results for the non-dimensional mean streamwise velocity profile, $\overline{U}/U_b$, are consistent across all models and grid generation approaches. No notable differences are observed between the conventional and AMR-based simulations, and the results agree well with the reference data. This indicates that the coarse initial mesh, combined with the targeted refinement strategy, is sufficient to accurately capture the mean velocity field.

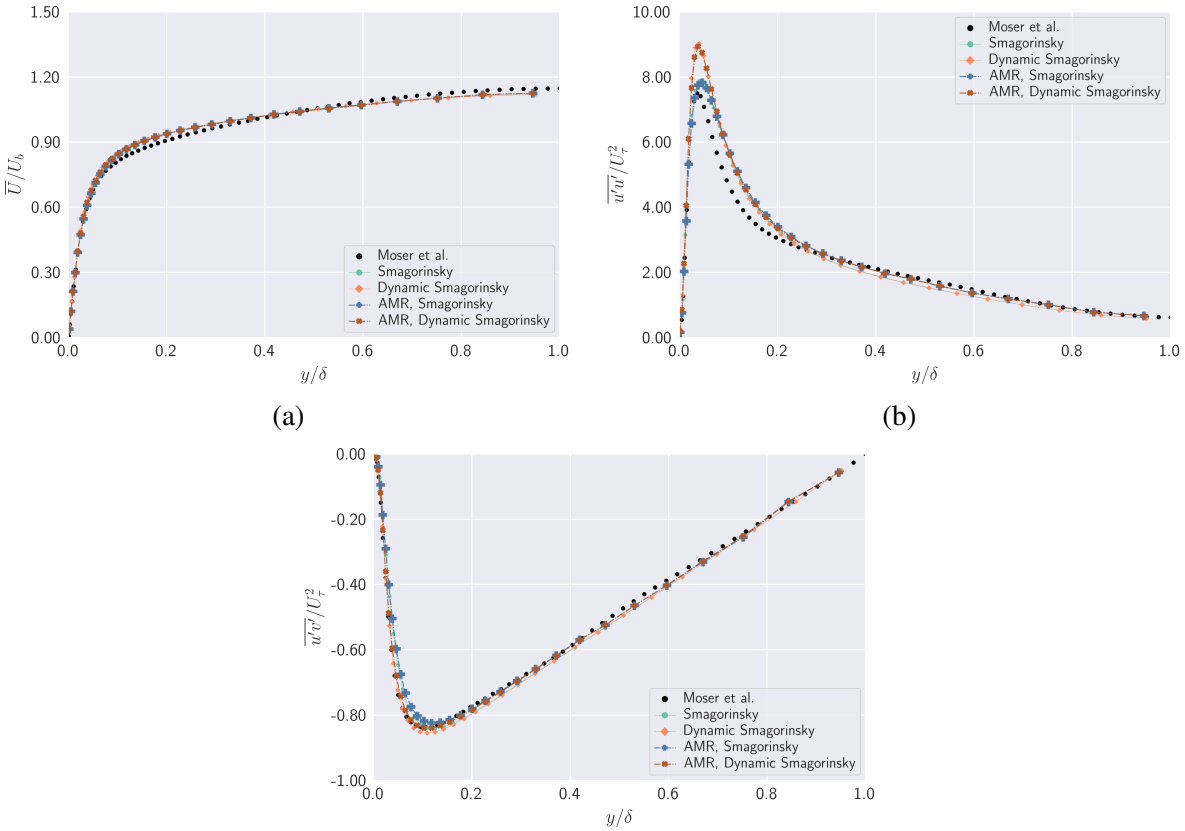
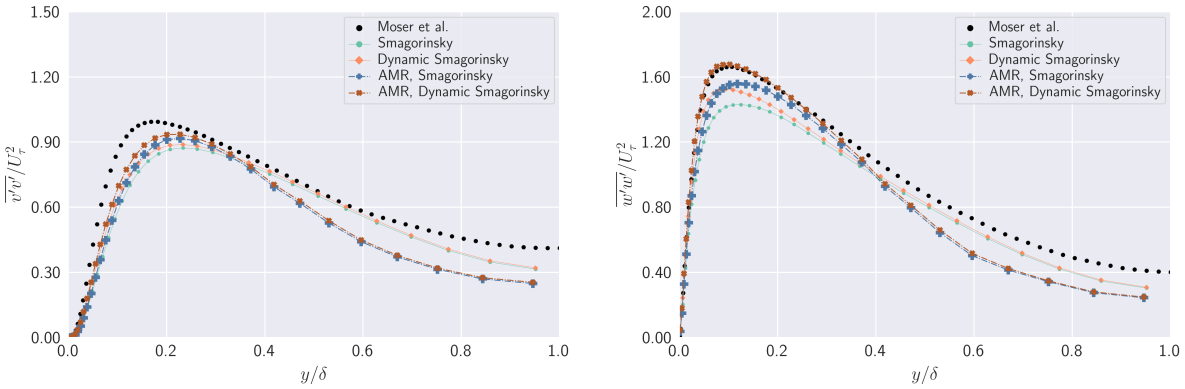


Figure 6.1: Results for the channel flow case using AMR: (a) non-dimensional mean streamwise velocity profile $\overline{U}/U_b$, (b) normalised streamwise Reynolds normal stress $\overline{u'u'}/U_\tau^2$, (c) normalised Reynolds shear stress $\overline{u'v'}/U_\tau^2$.

For the normalised streamwise Reynolds normal stress, $\overline{u'u'}/U_\tau^2$, the trends observed on the adaptive meshes closely follow those obtained on the conventional grids. The dynamic Smagorinsky model tends to overpredict the peak near $y/\delta \approx 0.05$, followed by a drop that aligns well with the reference data throughout the remainder of the profile. Notably, unlike the conventional grid results, which tend to underpredict the fluctuations beyond $y/\delta \approx 0.3$ slightly, the AMR-based results maintain good agreement in this region. The Smagorinsky model exhibits similar behaviour; the peak observed in the reference data is captured, although it appears slightly broader. After $y/\delta \approx 0.3$, the results align well with the dynamic model and the reference data.

Good agreement with reference data is observed for $\overline{u'v'}/U_\tau^2$. The results of the dynamic Smagorinsky model match the reference profile closely and outperform those from the conventional grid. For the Smagorinsky model, the peak is slightly delayed and underpredicted, but the overall trend remains consistent with the reference profile and shows better alignment than

the corresponding conventional results.



(a) (b)

Figure 6.2: Results for the channel flow case using AMR: (a) normalised wall-normal Reynolds normal stress $\overline{v'v'}/U_\tau^2$, (b) normalised spanwise Reynolds normal stress $\overline{w'w'}/U_\tau^2$.

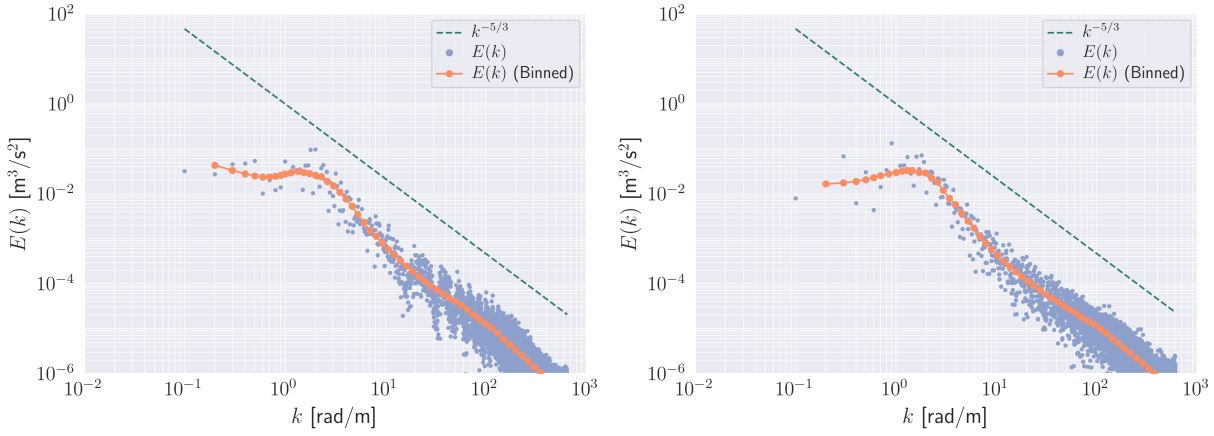
The results for the normalised wall-normal Reynolds normal stress and spanwise Reynolds normal stress shown in Figure 6.2 remain unsatisfactory. The conventional and AMR-based simulations show notable discrepancies when compared with reference data, suggesting that the error may stem from a more fundamental limitation, such as insufficient resolution and modelling constraints. For $\overline{v'v'}/U_\tau^2$, both AMR cases initially perform better than their conventional counterparts, approaching the peak, though delayed. However, beyond $y/\delta \approx 0.4$, both AMR cases show a sharp decline in accuracy and consistently underperform relative to the conventional grid results. A similar trend is observed for $\overline{w'w'}/U_\tau^2$, where the peak is well captured by the dynamic Smagorinsky model and slightly underpredicted by the Smagorinsky model, both outperforming the conventional approach in the near-wall region. However, beyond $y/\delta \approx 0.4$, the results deteriorate and fall below those obtained on the conventional grid.

The drop in accuracy in the outer region may be attributed to the reduced refinement in these areas, as the imposed cell count limit constrained the AMR process. This leads to a spatial imbalance in grid density, which likely contributed to the underprediction of wall-normal and spanwise stresses at larger wall distances.

The energy spectra presented in Figure 6.3 illustrate the distribution of turbulent kinetic energy as a function of wavenumber. Graphs show the characteristic three-region structure of turbulent flows: the energy-containing range at low wavenumbers, the inertial subrange at intermediate wavenumbers, and the dissipation range at high wavenumbers. In the energy-containing range, the energy levels increase with wavenumber from the lowest resolved modes,

reaching a peak around $k \approx 2$. This peak indicates the dominant eddy scale that carries most of the turbulent kinetic energy.

Beyond the peak, the spectra transition into the inertial subrange, where $E(k)$ declines gradually and approximately follows the Kolmogorov $k^{-5/3}$ scaling. At higher wavenumbers ($k > 100$), a steeper drop is observed, signalling the onset of the dissipation range, where viscous effects become significant. The raw simulation data extend well into this high-wavenumber region, demonstrating the fine-scale resolution captured by AMR. Both spectra display comparable shapes and magnitudes, capturing a physically consistent energy cascade.



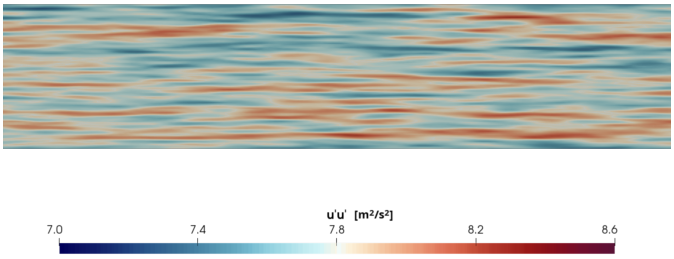
(a) (b)

Figure 6.3: Energy spectra for the channel flow case using AMR: (a) Smagorinsky model, (b) dynamic Smagorinsky model.

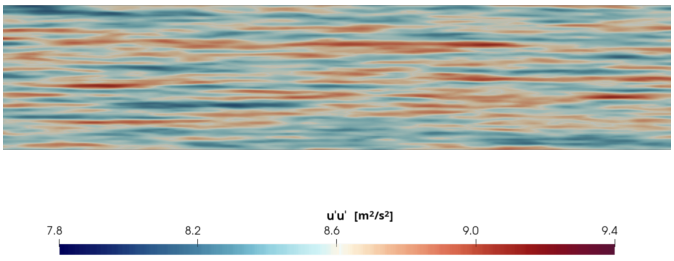
A sectional cut of the computational mesh, given in Figure 6.4, shows the refinements introduced during the simulation and corresponds to the mesh at the final time step. The mesh is visibly refined near the channel walls, where steep velocity gradients and intense vortical activity demand higher resolution.

Figure 6.4: Sample of the computational mesh for the channel flow case using AMR.

The cross-sectional views of the domain are given in Figure 6.5 and show elongated streaks at $y/\delta = 0.05$, indicative of alternating high-speed and low-speed fluid regions moving in the streamwise direction, which are characteristic of wall-bounded turbulent flows. Both figures show quasi-periodic streaks in the spanwise direction, perpendicular to the main flow, and the wall-normal direction. Upon closer inspection, the results for the dynamic Smagorinsky model appear more defined, exhibiting a wider range of velocity fluctuations compared to the standard Smagorinsky model. This suggests that the dynamic model resolves the finer details of the flow. Based on the lower and upper limits, it can be concluded that the dynamic Smagorinsky model predicts more intense and higher peak streamwise velocity fluctuations than the Smagorinsky model at this near-wall location and Reynolds number.



(a)



(b)

Figure 6.5: Fluctuating streamwise velocity $\overline{u'u'}$ at $y/\delta = 0.05$ for the channel flow case using AMR: (a) Smagorinsky model, (b) dynamic Smagorinsky model.

A three-dimensional visualisation of coherent vortical structures identified using an isosurface $Q = 50\ \mathrm{s}^{-2}$ is presented in Figure 6.6. The Q criterion highlights regions where rotational effects dominate over strain, thereby isolating the cores of vortical motion. The isosurface is coloured by the magnitude of local instantaneous velocity, revealing a strong correlation between high-speed regions and vortical activity. The vortices appear as elongated, tube-like,

and fragmented structures, densely concentrated near the channel walls, which is characteristic of wall-bounded turbulence. The colour variation along these structures indicates significant velocity gradients, reflecting dynamic interactions between slower and faster fluid regions. Notably, many high-velocity zones are embedded within or closely aligned with Q-identified vortices. In contrast, the central region of the channel contains relatively few such structures, consistent with the expected turbulence distribution in a fully developed channel flow.

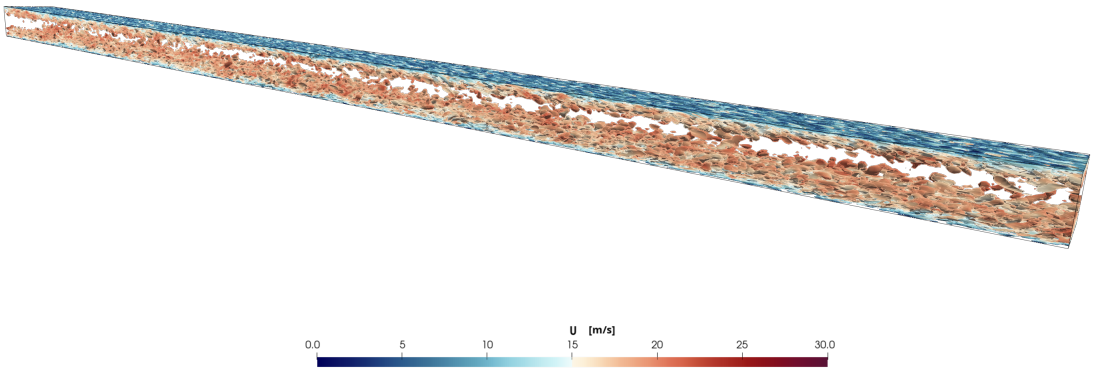


Figure 6.6: Isosurface $Q = 50$ coloured by velocity for the channel flow case using Smagorinsky model and AMR.

6.2.2 Assessment for Flow Around a Square Cylinder

A simulation was performed using adaptive mesh refinement for the flow around a square cylinder at $Re = 21400$. The simulation was initiated on a coarse base mesh of approximately $7.5 \cdot 10^4$ hexahedral cells. The initial grid featured a characteristic wall-normal resolution near the cylinder of $\Delta z/D \approx 0.01$ and a spanwise resolution of $\Delta y/D \approx 0.2$. These spacings are coarser than for the conventional LES, relying on AMR to achieve the targeted resolution.

Mesh refinement was driven by a composite criterion. The isovalue threshold for Φ_{λ_2} was set to 1.0, and the maximum filter width Φ_Δ was capped at 5.0 to avoid excessive refinement caused by transient or isolated flow features. Following the approach used in the channel flow test case, a smoothing filter S with a smoothing factor $f = 2.0$ was applied to suppress numerical noise in the criterion field.

Refinement was performed every 512 time steps. With a fixed time step of $\Delta t = 5 \cdot 10^{-5}$ s, this corresponds to a physical time interval of 0.0256 s between refinement cycles. Over the course of the simulation, approximately 1800 refinement steps were completed. The total cell count was limited to $3 \cdot 10^6$ to control the computational cost. A maximum of two refinement

levels beyond the initial mesh was allowed, with a single-cell buffer surrounding the refined regions to ensure a gradual transition in cell size.

Turbulence was modelled using the WALE subgrid-scale model, consistent with the conventional LES setup. As discussed in Subsection 2.6.2, the conventional simulations also included a WMLES case. However, the current WMLES implementation is incompatible with dynamically adapting meshes, as it depends on a fixed wall-layer sampling region. This constraint is a significant limitation. As a result, the WMLES case was excluded from the AMR validation. Other than the adaptive refinement mechanism, all other numerical settings, including the solver, working fluid, and temporal and spatial discretisation schemes, remained consistent with the conventional LES configuration.

As the simulation progressed, the mesh reached a final resolution $\Delta z/D = 0.002$ and $\Delta y/D = 0.03$ in the most refined regions. While the wall-normal resolution remained slightly coarser than in the conventional LES, the spanwise resolution was finer. Notably, the peak cell count was modest, approximately $1.7 \cdot 10^6$, which is roughly one-third of the total cell count used in conventional LES.

The results shown in Figure 6.7 are time-averaged streamwise velocity profiles at several cross-sections, normalised by the freestream velocity. Upstream of the cylinder ($x/D = -0.5$ and $x/D = 0.0$), all profiles align closely and match the reference data well. At $x/D = 0.5$, a slight deviation appears, with the AMR results showing a marginally higher velocity. However, the difference remains within an acceptable range and is comparable to the conventional case. Due to the absence of measurement data below $y/D \approx 0.75$, it is difficult to conclusively say which approach performs worse in that region. Further downstream, up to $x/D = 4.0$, the profiles from both cases largely overlap and remain consistent with the reference data. A modest deviation from the reference is observed at $x/D = 1.5$, though both numerical approaches behave similarly.

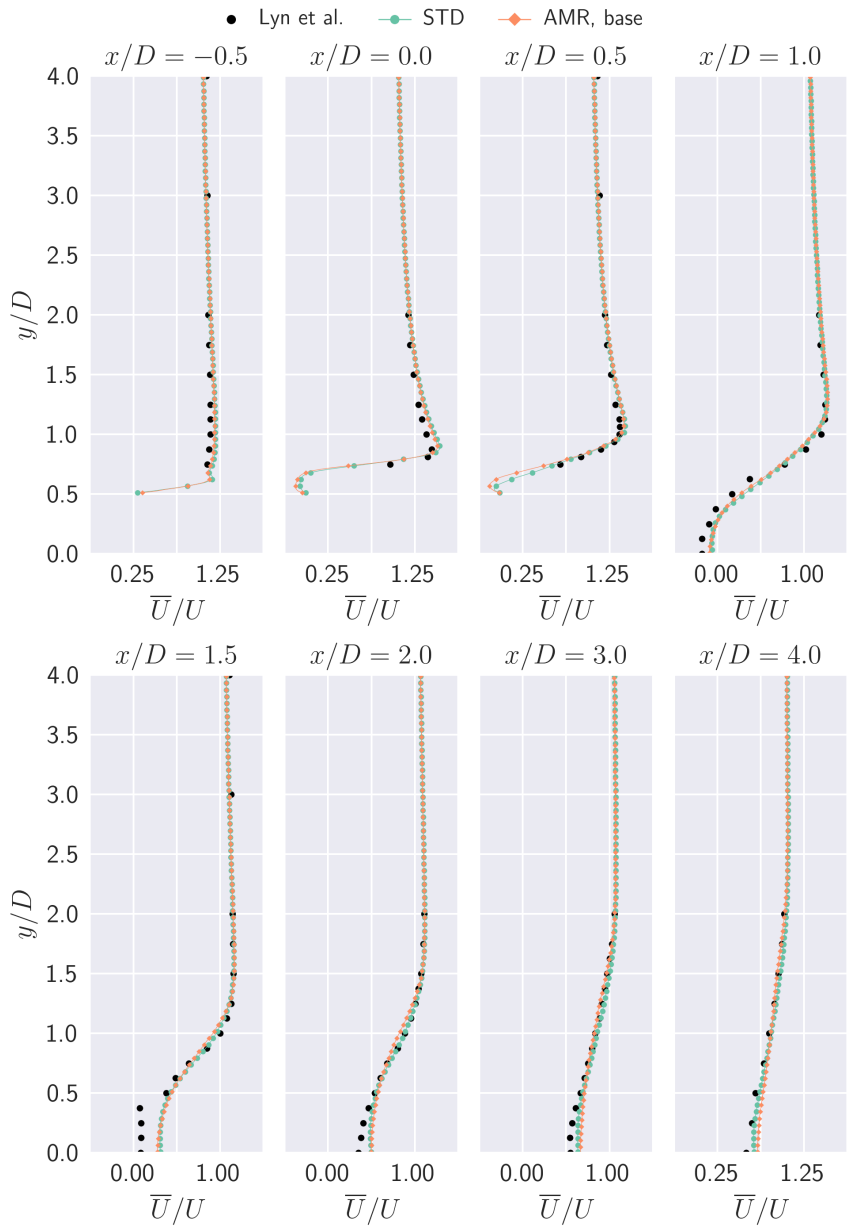


Figure 6.7: Results for the square cylinder test case using AMR showing $\overline{U}/U$ profiles at various cross-sections.

Profiles of normalised streamwise velocity fluctuations, u'/U, at various cross-sections are shown in Figure 6.8. At $x/D = 0.0$, a distinct peak emerges near $y/D \approx 0.75$, associated with turbulence in the separated shear layers. The AMR simulation captures both the magnitude and position of this peak with good accuracy, while the conventional approach yields a broader and slightly overestimated response. At $x/D = 0.5$ and $x/D = 1.0$, the peak structure persists and begins to widen, reflecting the development of the wake. The AMR results remain in agreement with the reference data. In contrast, the conventional simulation, although following the general

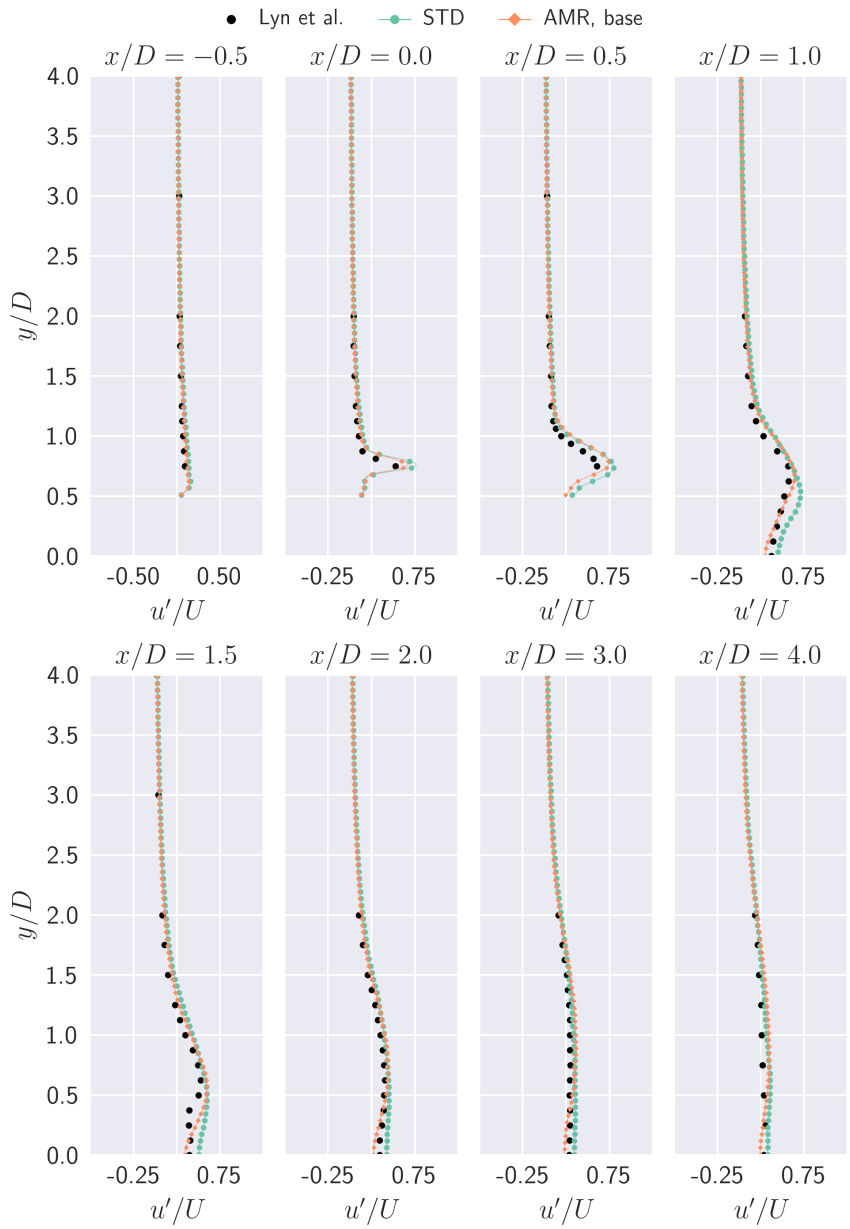


Figure 6.8: Results for the square cylinder test case using AMR showing u'/U profiles at various cross-sections.

trend, shows more pronounced deviations. Further downstream, turbulence levels gradually decrease, and the profiles broaden. The AMR solution continues to align well with reference data across all cross-sections, while the conventional approach consistently underperforms despite employing a larger mesh.

The profiles of the normalised time-averaged transverse velocity are presented in Figure 6.9. Overall, the trends for both cases align well with the reference data. In the near-wake region, inward-directed flow can be observed. At $x/D = 0.5$ and $x/D = 1.0$, the AMR simulation

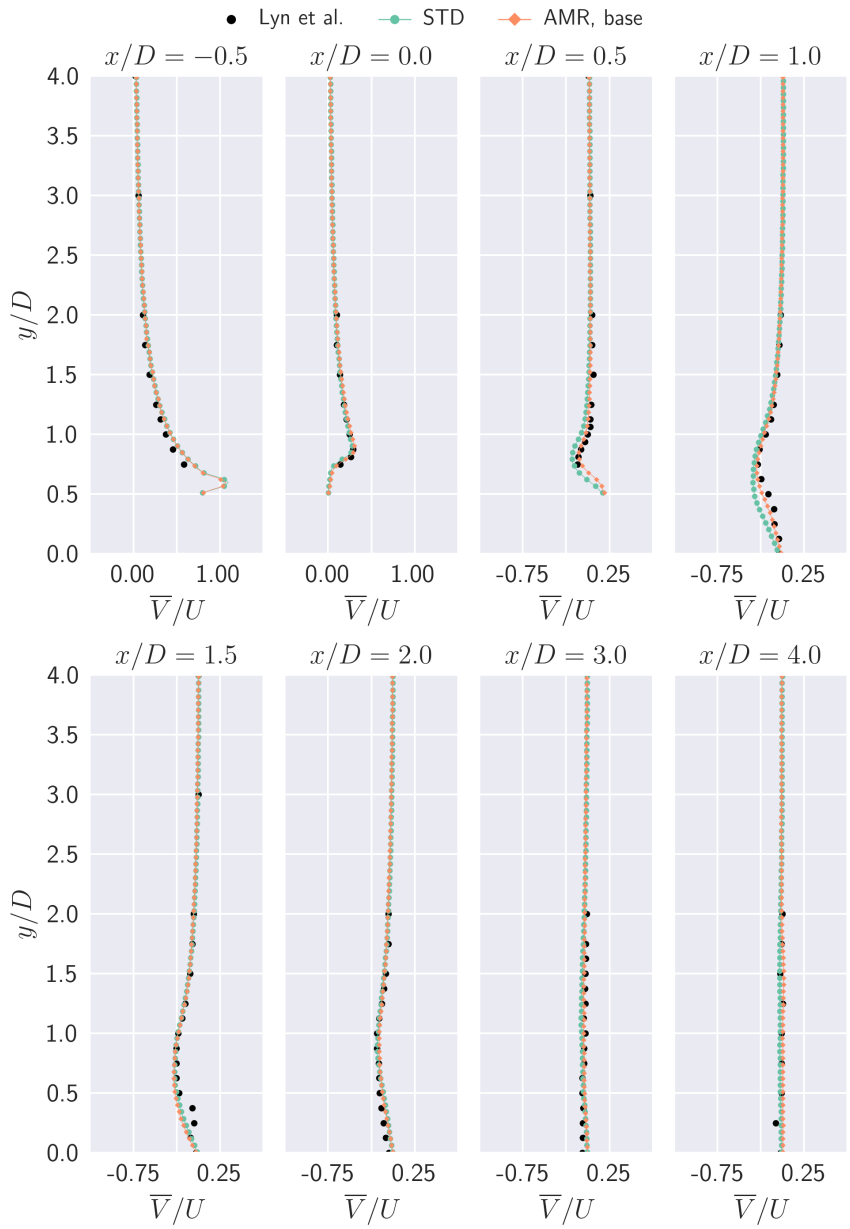


Figure 6.9: Results for the square cylinder test case using AMR showing $\overline{V}/U$ profiles at various cross-sections.

closely follows this behaviour, while the conventional simulation exhibits notable discrepancies. Further downstream, the transverse velocity profiles flatten, with both cases following the reference data.

Figure 6.10 shows the normalised transverse velocity fluctuations, v'/U, at various cross-sections. As with the mean transverse velocity, $\overline{V}/U$, the largest deviations are observed between $x/D = 0.5$ and $x/D = 1.5$. The AMR simulation closely matches the reference data, capturing the shape and magnitude of the v'/U peak. At $x/D = 1.0$, the peak remains prominent in the reference data, and the AMR results follow this trend. In contrast, conventional simulation

performs poorly across this range, with particularly pronounced discrepancies at $x/D = 1.0$. Further downstream, the profiles flatten and broaden. Beyond $x/D = 1.5$, the results from both simulations begin to converge and largely follow the reference data.

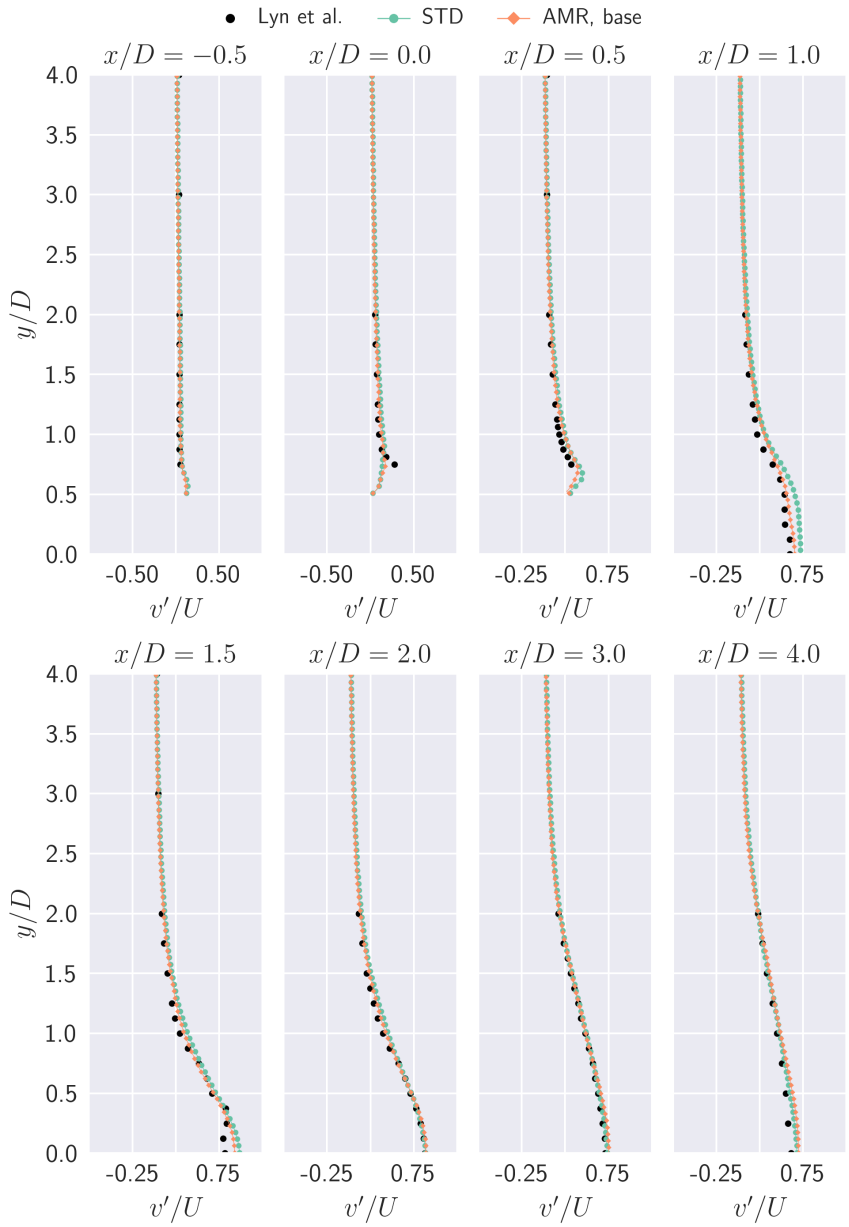


Figure 6.10: Results for the square cylinder test case using AMR showing v'/U profiles at various cross-sections.

A three-dimensional visualisation of the turbulent wake behind a square cylinder is given in Figure 6.11. The isosurface shown corresponds to $Q = 10\ \mathrm{s}^{-2}$ and is coloured by the magnitude of the instantaneous velocity. Immediately downstream of the cylinder, large coherent vortices can be observed shedding alternately from the top and bottom shear layers. These structures, initially aligned with the spanwise direction, exhibit strong three-dimensional instabilities as they

evolve downstream. Spanwise distortions, braids, and rib-like connections between vortices indicate the onset of turbulence and the breakdown of coherent rollers into smaller-scale eddies. Further into the wake, the vortices lose coherence, and the flow field becomes increasingly disordered, marking the transition to a fully developed turbulent regime. The spatial variation in colour across the isosurfaces reflects intense mixing and momentum transfer. High velocities are often seen on the peripheries of the vortices or in interstitial regions, while low-velocity zones cluster around vortex cores or near recirculation zones.

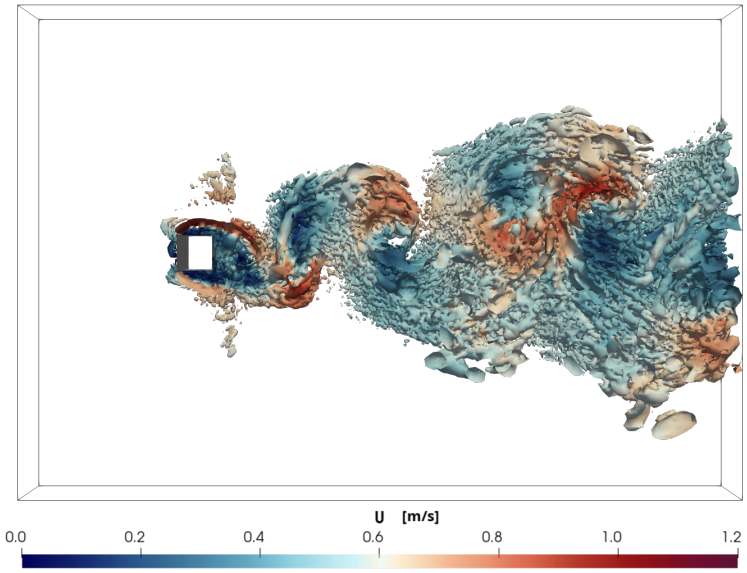


Figure 6.11: Isosurface $Q = 10$ coloured by velocity for the square cylinder test case using AMR.

Figure 6.12 illustrates the spatial distribution of mesh refinement criteria in the square cylinder test case at the final time step. The blue regions represent areas refined according to the Φ_{λ_2} criterion, while the red regions correspond to areas governed by the Φ_Δ criterion.

The blue regions dominate the wake and shear layers downstream of the cylinder, effectively capturing the unsteady flow structures associated with vortex shedding. In contrast, the red regions appear as more localised patches, typically near the cylinder. It is important to note that since the mesh has already been refined, the influence of the Φ_Δ criterion in the wake region has already been addressed. Consequently, no further refinement is necessary in this zone, which now also falls under the influence of the Φ_{λ_2} criterion.

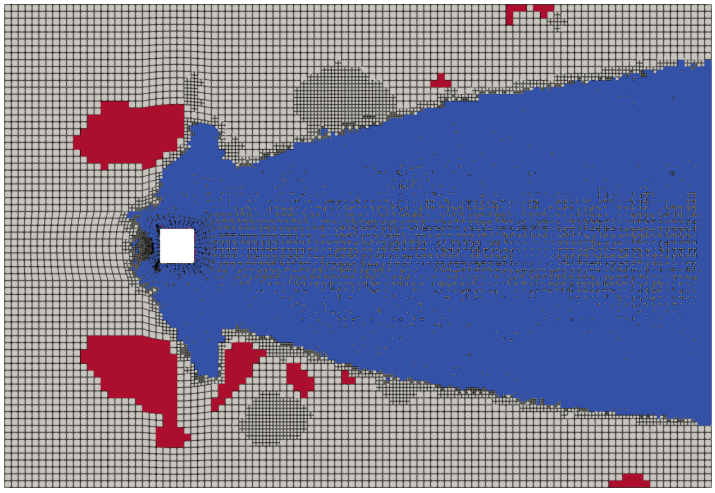


Figure 6.12: Regions of influence for refinement criteria in the square cylinder test case with AMR at the final time step. Blue indicates Φ_{λ_2}, red indicates Φ_Δ.

6.2.3 Assessment for Turbulent Mixing of Jet in Crossflow

Turbulent mixing of a jet in crossflow is investigated as a representative case involving scalar transport and complex turbulence dynamics. The overall setup follows the previously described configuration, with the exception of the WMLES case, which was omitted due to incompatibility between the WMLES library and the AMR framework. An additional test case using the Spalart-Allmaras improved delayed detached eddy simulation (IDDES) was introduced. The general setup is unchanged, except for the use of the `LUST` scheme for discretising the modelled turbulence viscosity. IDDES is an improvement over the traditional detached eddy simulation (DES) and combines features of RANS near the walls and LES in the free stream.

For the initial mesh, the number of cells in the cylindrical section of the jet inlet and its associated O-grid was halved. This coarser discretisation was uniformly applied across the entire domain. The main channel was meshed with $150 \times 54 \times 54$ cells in the streamwise x, vertical y, and spanwise z directions, respectively. The jet cylinder was discretised with 12 cells in the radial direction and 75 cells in the axial direction.

The refinement strategy followed a multi-criteria AMR approach. The Φ_{λ_2} criterion used an isovalue threshold of 3.0, with the mesh-based Φ_Δ criterion lower limit set to $\delta_{\Phi_\Delta} = 10.0$. A smoothing operator S with a smoothing factor of $f = 2.0$ was used to suppress numerical noise. Additionally, a scalar-based refinement criterion was introduced to capture the transport of the passive scalar c. Refinement was triggered in regions where the scalar concentration c was between the lower limit $\delta_c = 0.05$ and the upper limit $\varepsilon_c = 1.0$.

As in the square cylinder case, a maximum of two additional refinement levels was allowed, with a single-cell buffer layer around refined regions. The total cell count was limited to $3 \cdot 10^6$. Refinement was conducted every 256 time steps, which corresponds to a physical time interval of 0.000768 s, given the fixed time step size of $\Delta t = 3 \cdot 10^{-6}$ s. Consequently, 1300 refinement steps were performed over the course of the simulation.

Figure 6.13 compares results from three different simulations: conventional, AMR with WALE, and AMR with IDDES, against reference LES data. The resulting velocity profiles are similar across all cross-sectional slices ($z/D = 1.5$, 3.0, and 4.5). At $z/D = 1.5$, all simulations slightly overpredict the velocity between $x/D \approx 0.4$ and $x/D \approx 1.0$. At $z/D = 3.0$, the AMR case using the WALE model shows the best agreement with the reference data, while IDDES-based AMR performs slightly worse, but still outperforms the simulation on the conventional grid. All models show similar velocity values for $x/D \leq 0.4$ and $x/D \geq 2.0$. The differences become minimal for $z/D = 4.5$, with minor discrepancies around $1.0 \leq x/D \leq 1.8$. Elsewhere, the deviations from the reference data are negligible.

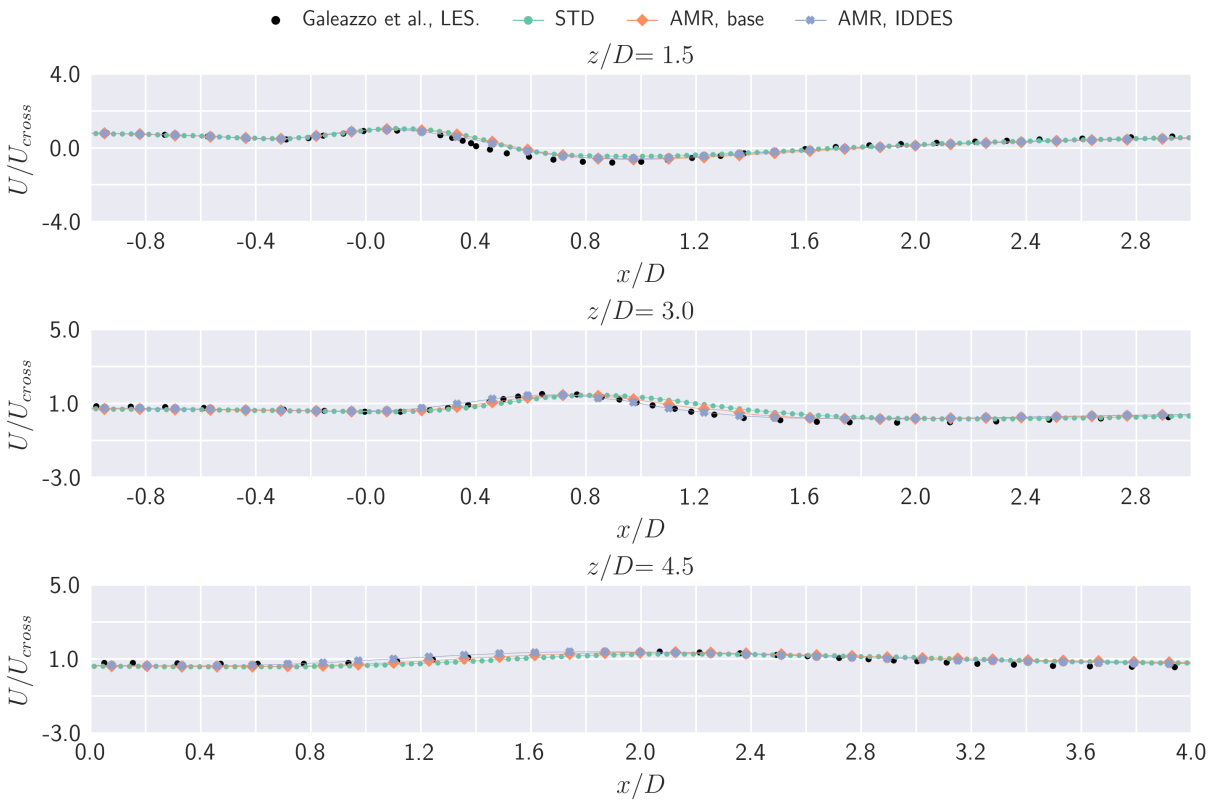


Figure 6.13: Normalised velocity at different cross-sections for the turbulent mixing test case using AMR.

For scalar transport, shown in Figure 6.14, greater variation in the data is observed across all three cases. At $z/D = 1.5$, conventional results underpredict the peak scalar concentration. Both AMR simulations match the reference data closely, although they slightly overpredict the concentration between $0.4 \leq x/D \leq 0.8$. For $z/D = 3.0$, the conventional results again underpredict the scalar concentration. The AMR cases are in reasonable agreement but slightly overpredict downstream of $x/D \approx 0.8$. By $z/D = 4.5$, all models align well with the reference, with the AMR simulations slightly overpredicting the scalar concentration and the conventional simulation underpredicting it.

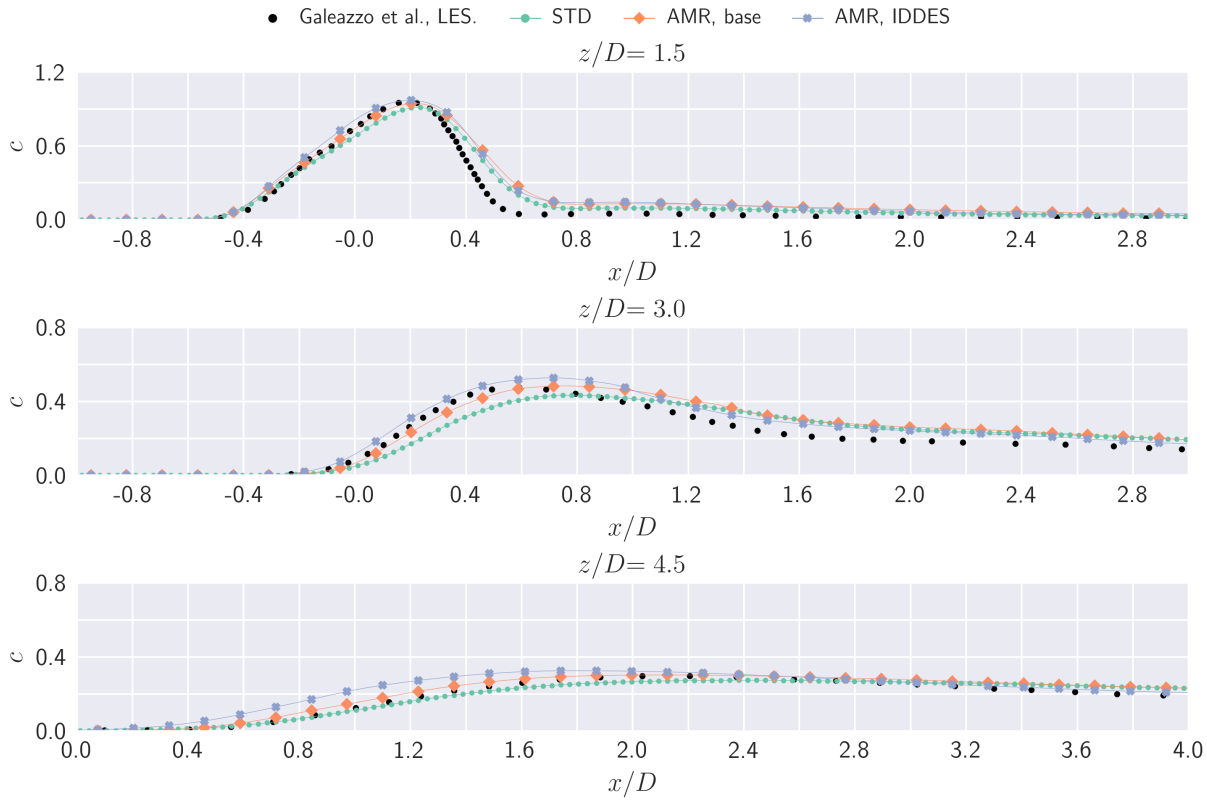


Figure 6.14: Scalar concentration at different cross-sections for the turbulent mixing test case using AMR.

The isosurface $c = 0.1$ given in Figure 6.15 shows the extent of the scalar plume. The plume originates from a cylindrical source at the bottom of the domain, with the scalar being continuously released and spreading outward (and downstream) as it rises. The irregular shape of the isosurface stems from the turbulent nature of the flow, with eddies causing the scalar to spread and mix as it flows through the domain. As the plume moves downstream, it widens and becomes more fragmented due to the entrainment of ambient fluid, which dilutes the scalar concentration.

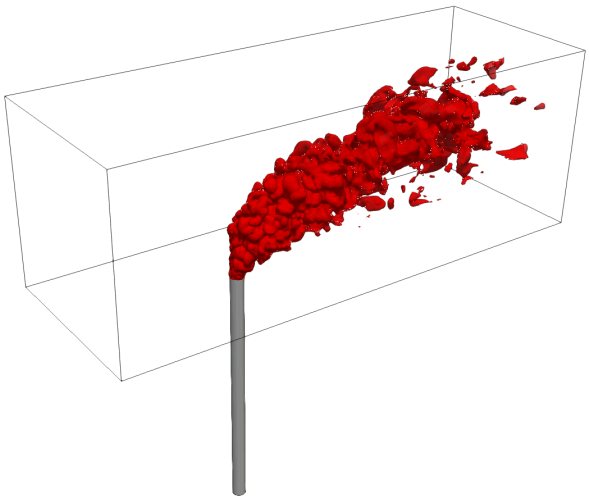


Figure 6.15: Isosurface $c = 0.1$ for the turbulent mixing test case using AMR.

Fine-scale vortical structures are visualised in Figure 6.16 using the isosurface $Q = 25000\,\mathrm{s}^{-2}$, coloured by the magnitude of the instantaneous velocity. Filament-like structures originate above the cylindrical source and follow the evolving plume. Their fragmented, thread-like appearance reflects the intermittent and anisotropic nature of turbulence at these scales. Spatial variations in velocity along the isosurface reveal regions of strong momentum transport embedded within the flow.

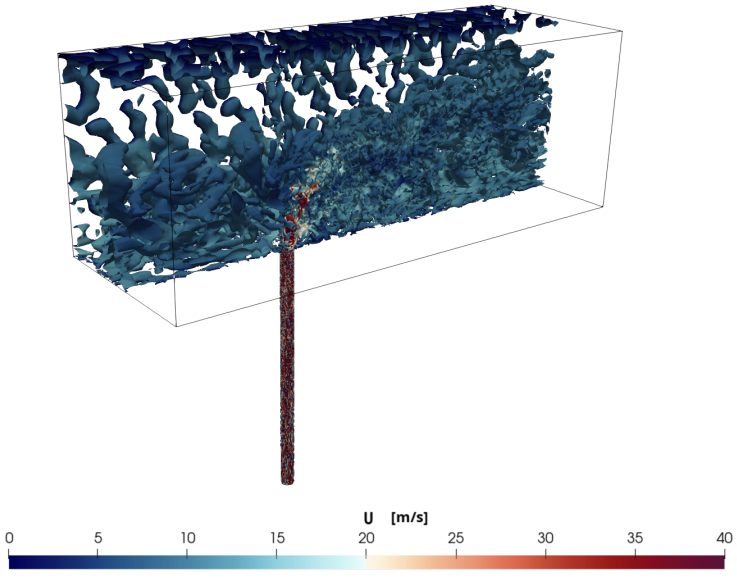


Figure 6.16: Isosurface $Q = 25000$ coloured by velocity for the turbulent mixing test case using AMR.

Results presented in Figure 6.17 show the instantaneous velocity field U and the time-averaged scalar concentration c at $t = 1$ s. In the vertical midplane, $y/D = 0$, a narrow, high-velocity jet emerges from the lower boundary. The jet core is enveloped by irregular structures, which is indicative of turbulent mixing. Peak velocities reach approximately 55 m/s. The development of shear layers between the jet and the ambient fluid is visible and is the primary mechanism for turbulence generation. In the horizontal cross-section, at $z/D = 1.5$, a crescent-shaped region of elevated velocity can be seen, which suggests the presence of large-scale vortical motion.

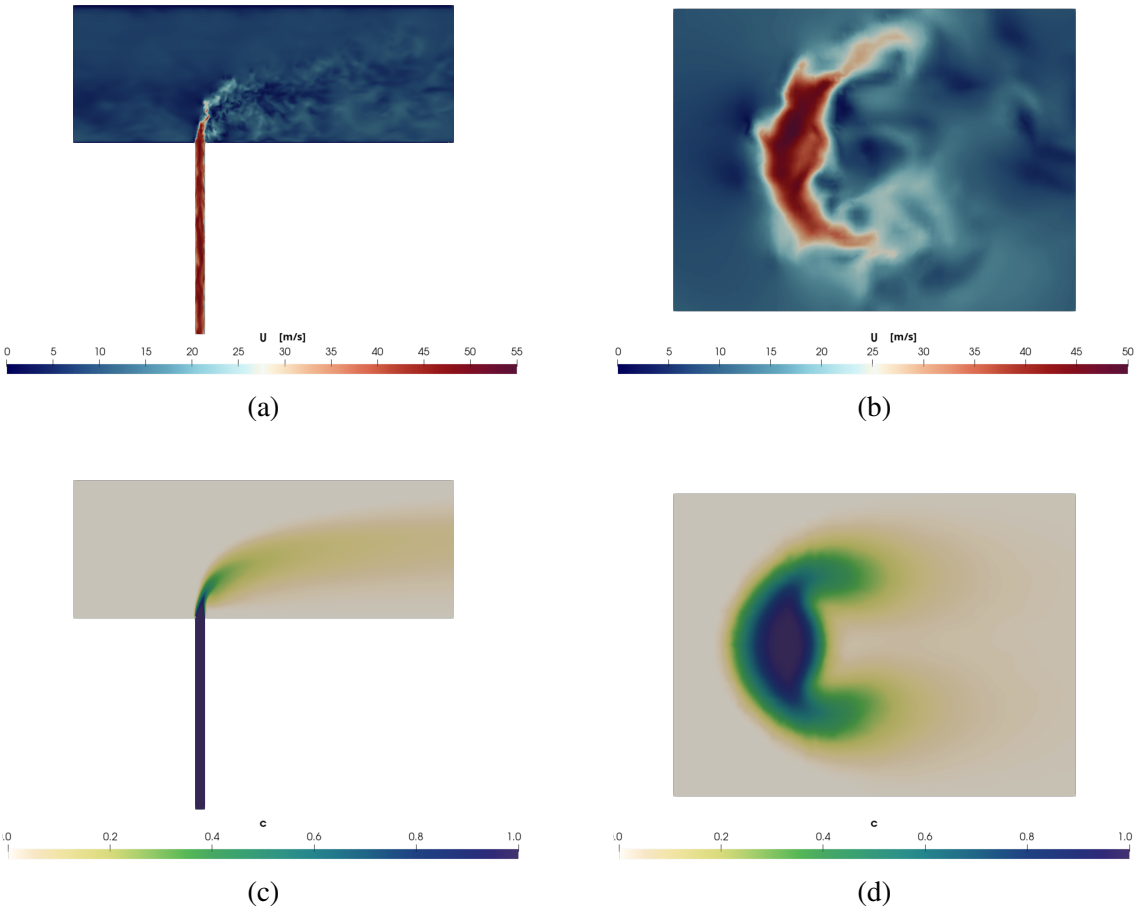


Figure 6.17: Velocity U and scalar c at $t = 1$ s, shown at different cross-sections: (a) U at $y/D = 0$, (b) U at $z/D = 1.5$, (c) c at $y/D = 0$, (d) c at $z/D = 1.5$.

The time-averaged scalar field in the vertical plane $y/D = 0$ shows a smoother distribution. The averaging process suppresses instantaneous fluctuations, revealing a statistically steady plume shape. The scalar field gradually widens with height. Concentration is highest within

the jet core and decreases outward and downstream, reflecting the combined effects of advection and turbulent diffusion. In the horizontal plane, at $z/D = 1.5$, the scalar concentration also shows a crescent-like shape, similar in form to the velocity field, but smoother and more regular. The distribution is largely symmetric, with the internal structure less detailed than the instantaneous velocity field.

6.3 Computational Cost and Accuracy

The results for all described LES cases, including conventional and AMR-based simulations, are compared with available reference data. Dynamic time warping was used to quantify the differences between the reference and obtained results. The resulting DTW values were then used to calculate average relative differences across all measurement locations for each case. These results are shown in Table 6.1.

Table 6.1: DTW-based difference between conventional and AMR cases.

Test case	**Comparison**	**Turbulence model**	**DTW difference [%]**
Channel flow	Conventional / AMR	Smagorinsky	-2.61
	Conventional / AMR	Dynamic Smagorinsky	-1.51
Square cylinder	Conventional / AMR	WALE	-6.11
	WMLES / AMR	WALE	-10.38
Turbulent mixing	Conventional / AMR	WALE	-6.41
	WMLES / AMR	WALE	0.74
	Conventional / AMR	WALE / IDDES	-3.78
	WMLES / AMR	WALE / IDDES	2.11

The results suggest that, in nearly all cases, AMR simulations outperform their conventional counterparts. The only exception is the turbulent mixing test case, where the WMLES simulation outperformed the baseline AMR case by less than 1 % on average. Similarly, the IDDES model performed about 2.11 % worse than the WMLES baseline. In all other instances, AMR-based simulations showed notable improvements.

In addition, computational times were recorded for all cases. The simulations were executed on a cluster (described in Chapter 5) using 120 CPUs. For AMR cases, measurements were obtained without and with load balancing, using the `distributorMPI` load balancer with a maximum imbalance of 0.2. The redistribution interval was set to half the refinement interval. These results are shown in Table 6.2.

Table 6.2: Computational times for conventional and AMR cases.

Test case	**Turbulence model**	**Computational time [s]**		
		Conventional	**AMR**	**AMR with LB**
Channel flow	Smagorinsky	66837	200491	80899
	Dynamic Smagorinsky	72378	260446	85080
Square cylinder	WALE	353386	304333	103148
	WALE (WMLES)	357756	-	-
Turbulent mixing	WALE	85058	368924	84030
	IDDES	91251	357242	89101
	WALE (WMLES)	89450	-	-

As can be seen, even though AMR simulations provide more accurate results, their computational cost is considerably higher. Without an appropriate load balancer, the AMR approach is inefficient for the problems presented. With load balancing, computational times are reduced significantly. However, the computational times for the channel flow case are still up to 21 % slower, which is expected, since approximately 1900 adaptation steps were performed. By increasing the refinement frequency (every 160 time steps), the computational time for the channel flow case becomes 3.71 % lower than that of the conventional approach. This highlights the importance of an effective load balancing strategy, as well as a well-designed refinement approach, to ensure both numerical accuracy and computational efficiency.

7 CONCLUSION

This thesis addresses the computational feasibility of adaptive mesh refinement for high-fidelity simulations within the OpenFOAM 10 framework. The proposed additions extend the OpenFOAM's functionality. They introduce refinement logic for two-dimensional problems, enable multi-criteria refinement, and load balancing for large-scale transient simulations.

The newly introduced class `refiner2D` is a 2D-specific variant of the native `refiner` class. It relies on edge data and enables consistent hexahedral refinement and unrefinement following a 1-to-4 pattern. This functionality is enabled by the newly implemented mesh cutter `hexRef4` and dedicated data hierarchy.

Building on this foundation, two new classes, `multiFieldRefiner2D` and `multiFieldRefiner3D`, were introduced. These classes extend the functionality of the native `refiner` class by allowing refinement based on multiple scalar fields combined using logical operators. Additionally, geometric constraints are supported, providing the ability to restrict refinement to specific regions.

To ensure parallel efficiency, two additional classes for dynamic load balancing were introduced: `distributorMPI` and `distributorRollingMPI`. These classes utilise the custom instrumentation of the MPI communication layer to track real-time processor loads and memory usage. The latter leverages a rolling history mechanism to smooth short-lived fluctuations caused by transient instabilities or hardware-related noise.

In addition to the AMR-specific developments, several complementary code modifications were made to streamline simulation workflows. These include:

- Port of the `perturbU` utility, which is used to initialise velocity field in channel flow simulations, thus eliminating the need for a precursor run.
- Port of the `dynamicSmagorinsky` turbulence model, which improves upon the classical Smagorinsky model by dynamically adjusting the model constant.
- Port of the `libWallModelledLES` library. This methodology allows for a cheaper LES by not resolving the inner region of the turbulent boundary layer.

All introduced code modifications were implemented to validate the effectiveness of adaptive mesh refinement. Test cases using laminar and RANS turbulence models were employed to assess the overall functionality. The impact of load balancing was evaluated on a selected benchmark case. Finally, a multi-criteria AMR approach was proposed and assessed for selected LES test cases. Based on the results, the following conclusions can be drawn:

- The proposed multi-criteria refinement strategy that combines the λ_2 criterion with a mesh-based criterion, Δ, is effective. This hybrid approach integrates a flow-detecting criterion and a mesh quality detector to determine whether refinement is necessary. The proposed criterion is straightforward to implement, easy to interpret, and can be extended with supplementary criteria.

- AMR can be effectively deployed with LES, as demonstrated by the results. However, frequent and extensive adaptations can impact accuracy. Therefore, the adaptation process should be gradual and controlled, to avoid instabilities and numerical issues. The WALE and dynamic Smagorinsky models perform adequately, while WMLES is incompatible.

- Load balancing is critical to ensure computational efficiency. It can reduce computational time by several orders of magnitude, making AMR more efficient than conventional LES approaches. Since refinement and load balancing frequencies are user-defined, they must be chosen carefully.

Due to the changing nature of the underlying grid, several obstacles were encountered while using AMR. In addition to the WMLES library, issues were observed with the `interfaceHeight` utility and postprocessing tools. For example, when decomposing or reconstructing a case, appropriate utilities are required to preserve the refinement history. This challenge is particularly relevant for the newly developed two-dimensional AMR classes. When using load balancing, the distribution process must include a decomposition constraint, and if periodic patches are involved, these patches must remain on the same processor to avoid distribution-induced crashes.

Based on the observations, general AMR behaviour, and results, several distinct avenues for future research are identified:

- Interplay between AMR parameters and flow physics.

- Numerical limitations and stability concerns.

- Implementation of anisotropic refinement.

- Extension of the framework to accommodate polyhedral mesh topologies.

The developments in multi-criteria AMR presented in this work represent a significant step toward more efficient and accurate simulations of complex, transient fluid dynamics problems. Observed computational savings are particularly promising for resource-intensive methods such as LES, supporting their broader adoption in research and industrial applications.

BIBLIOGRAPHY

[1] J. Adelsberger, P. Esser, M. Griebel, S. Groß, M. Klitz, and A. Rüttgers, "3d incompressible two-phase flow benchmark computations for rising droplets," Institut für Numerische Simulation (INS), Tech. Rep., 2014.

[2] P. Alberto, *Dynamic smagorinsky model*, `https://doi.org/10.5281/zenodo.4697995`, Accessed: 2025-05-01, 2021.

[3] J. D. Anderson and J. Wendt, *Computational Fluid Dynamics*. Springer, 1995, vol. 206.

[4] O. Antepara, O. Lehmkuhl, R. Borrell, J. Chiva, and A. Oliva, "Parallel adaptive mesh refinement for large-eddy simulations of turbulent flows," *Computers & Fluids*, vol. 110, pp. 48–61, 2015.

[5] D. Apte, M. Ge, and O. Coutier-Delgosha, "Investigation of cloud cavitating flow in a venturi using adaptive mesh refinement," *Journal of Hydrodynamics*, vol. 36, no. 5, pp. 898–913, 2024.

[6] D. Arney, "An adaptive method with mesh moving and mesh refinement for solving the euler equations," in *Proceedings of the 1st National Fluid Dynamics Conference*, AIAA, 1988, p. 3567.

[7] I. Babuška and W. C. Rheinboldt, "Adaptive approaches and reliability estimations in finite element analysis," *Computer Methods in Applied Mechanics and Engineering*, vol. 17, pp. 519–540, 1979.

[8] I. Babuška, "The selfadaptive approach in the finite element method," in *The Mathematics of Finite Elements and Applications II (MAFELAP)*, New York: Academic Press, 1975, pp. 125–142.

[9] I. Babuška and W. C. Rheinboldt, "Error estimates for adaptive finite element computations," *SIAM Journal on Numerical Analysis*, vol. 15, no. 4, pp. 736–754, 1978.

[10] J. Baiges and C. Bayona, "Refficientlib: An efficient load-rebalanced adaptive mesh refinement algorithm for high-performance computational physics meshes," *SIAM Journal on Scientific Computing*, vol. 39, no. 2, pp. C65–C95, 2017.

[11] R. E. Bank and R. K. Smith, "Mesh smoothing using a posteriori error estimates," *SIAM Journal on Numerical Analysis*, vol. 34, no. 3, pp. 979–997, 1997.

[12] J. Behrens and M. Bader, "Efficiency considerations in triangular adaptive mesh refinement," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1907, pp. 4577–4589, 2009.

[13] M. J. Berger and P. Colella, "Local adaptive mesh refinement for shock hydrodynamics," *Journal of Computational Physics*, vol. 82, no. 1, pp. 64–84, 1989.

[14] M. J. Berger and J. Oliger, "Adaptive mesh refinement for hyperbolic partial differential equations," *Journal of Computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[15] E. Blayo and L. Debreu, "Adaptive mesh refinement for finite-difference ocean models: First experiments," *Journal of Physical Oceanography*, vol. 29, no. 6, pp. 1239–1250, 1999.

[16] L. Botti, M. Piccinelli, B. Ene-Iordache, A. Remuzzi, and L. Antiga, "An adaptive mesh refinement solver for large-scale simulation of biological flows," *International Journal for Numerical Methods in Biomedical Engineering*, vol. 26, no. 1, pp. 86–100, 2010.

[17] P. Brambilla and A. Guardone, "Assessment of dynamic adaptive grids in volume-of-fluid simulations of oblique drop impacts onto liquid films," *Journal of Computational and Applied Mathematics*, vol. 281, pp. 277–283, 2015.

[18] R. S. Cant et al., "An unstructured adaptive mesh refinement approach for computational fluid dynamics of reacting flows," *Journal of Computational Physics*, vol. 468, p. 111 480, 2022.

[19] J. J. Cooke, L. M. Armstrong, K. H. Luo, and S. Gu, "Adaptive mesh refinement of gas–liquid flow on an inclined plane," *Computers & Chemical Engineering*, vol. 60, pp. 297–306, 2014.

[20] P. Dechaumphai, "Evaluation of an adaptive unstructured remeshing technique for integrated fluid-thermal-structural analysis," *Journal of Thermophysics and Heat Transfer*, vol. 5, no. 4, pp. 599–606, 1991.

[21] L. Demkowicz, J. T. Oden, W. Rachowicz, and O. Hardy, "Toward a universal hp adaptive finite element strategy, part 1. constrained approximation and data structure," *Computer Methods in Applied Mechanics and Engineering*, vol. 77, no. 1–2, pp. 79–112, 1989.

[22] L. F. Diachin, R. Hornung, P. Plassmann, and A. Wissink, "Parallel adaptive mesh refinement," in *Parallel Processing for Scientific Computing*, SIAM, 2006, pp. 143–162.

[23] V. Dobrev, P. Knupp, T. Kolev, K. Mittal, and V. Tomov, "Hr-adaptivity for nonconforming high-order meshes with the target matrix optimization paradigm," *Engineering with Computers*, vol. 38, no. 4, pp. 3721–3737, 2022.

[24] A. Dubey et al., "A survey of high level frameworks in block-structured adaptive mesh refinement packages," *Journal of Parallel and Distributed Computing*, vol. 74, no. 12, pp. 3217–3227, 2014.

[25] C. M. Ertl, "A decentral framework with dynamic partitioning for numerical computations on massively parallel systems," Ph.D. dissertation, Technische Universität München, 2022.

[26] ESI-OpenCFD, *Openfoam*, `https://www.openfoam.com`, Accessed: 2025-04-29, Apr. 2025.

[27] J. H. Ferziger and M. Perić, *Computational methods for fluid dynamics*. Springer, 2002.

[28] C. Foucart, A. Charous, and P. F. J. Lermusiaux, "Deep reinforcement learning for adaptive mesh refinement," *Journal of Computational Physics*, vol. 491, p. 112 381, 2023.

[29] T. O. Foundation, *Openfoam v10*, `https://www.openfoam.org/version/10`, Release date: 2022-07-12, Jul. 2022.

[30] A. Fournier, G. Beylkin, and V. Cheruvu, "Multiresolution adaptive space refinement in geophysical fluid dynamics simulation," in *Adaptive Mesh Refinement–Theory and Applications: Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3–5, 2003*, Springer, 2005, pp. 161–170.

[31] D. Franke, "Investigation of mechanical contact problems with high-order finite element methods," Ph.D. dissertation, Technische Universität München, 2011.

[32] E. Frederix, J. A. Hopman, T. Karageorgiou, and E. M. J. Komen, "Towards direct numerical simulation of turbulent co-current taylor bubble flow," *arXiv preprint arXiv:2010.03866*, 2020.

[33] N. Freymuth, P. Dahlinger, T. Würth, S. Reisch, L. Kärger, and G. Neumann, "Swarm reinforcement learning for adaptive mesh refinement," *Advances in Neural Information Processing Systems*, vol. 36, pp. 73 312–73 347, 2023.

[34] B. Fryxell et al., "Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes," *The Astrophysical Journal Supplement Series*, vol. 131, no. 1, p. 273, 2000.

[35] J. P. de Gago, D. W. Kelly, O. C. Zienkiewicz, and I. Babuška, "A posteriori error analysis and adaptive processes in the finite element method: Part ii—adaptive mesh refinement," *International Journal for Numerical Methods in Engineering*, vol. 19, no. 11, pp. 1621–1656, 1983.

[36] F. C. C. Galeazzo, G. Donnert, P. Habisreuther, N. Zarzalis, R. J. Valdes, and W. Krebs, "Measurement and simulation of turbulent mixing in a jet in crossflow," *Journal of Engineering for Gas Turbines and Power*, vol. 133, no. 6, 2011.

[37] F. C. C. Galeazzo et al., "Computational modeling of turbulent mixing in a jet in crossflow," *International Journal of Heat and Fluid Flow*, vol. 41, pp. 55–65, 2013.

[38] J. Geese, J. Kimmerl, M. Nadler, and M. Abdel-Maksoud, "Adaptive mesh refinement for trailing vortices generated by propellers in interaction with slipstream obstacles," *Journal of Marine Science and Engineering*, vol. 11, no. 11, p. 2148, 2023.

[39] J. Gou, X. Su, and X. Yuan, "Adaptive mesh refinement method-based large eddy simulation for the flow over circular cylinder at re d= 3900," *International Journal of Computational Fluid Dynamics*, vol. 32, no. 1, pp. 1–18, 2018.

[40] J. A. Gutiérrez Suárez, C. H. Galeano Urueña, and A. Gómez Mejía, "Adaptive mesh refinement strategies for cost-effective eddy-resolving transient simulations of spray dryers," *ChemEngineering*, vol. 7, no. 5, p. 100, 2023.

[41] J.-F. Hétu and D. H. Pelletier, "Fast, adaptive finite element scheme for viscous incompressible flows," *AIAA Journal*, vol. 30, no. 11, pp. 2677–2682, 1992.

[42] G. Hindi, E. E. Paladino, and A. A. M. de Oliviera Jr, "Effect of mesh refinement and model parameters on les simulation of diesel sprays," *International Journal of Heat and Fluid Flow*, vol. 71, pp. 246–259, 2018.

[43] C. W. Hirt and B. D. Nichols, "Volume of fluid (vof) method for the dynamics of free boundaries," *Journal of Computational Physics*, vol. 39, no. 1, pp. 201–225, 1981.

[44] J. Holke, "Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types," *arXiv preprint arXiv:1803.04970*, 2018.

[45] S. Hysing et al., "Quantitative benchmark computations of two-dimensional bubble dynamics," *International Journal for Numerical Methods in Fluids*, vol. 60, no. 11, pp. 1259–1288, 2009.

[46] H. Jasak and A. D. Gosman, "Automatic resolution control for the finite-volume method, part 2: Adaptive mesh refinement and coarsening," *Numerical Heat Transfer, Part B: Fundamentals*, vol. 38, no. 3, pp. 257–271, 2000.

[47] M. T. Jones and P. E. Plassmann, "Parallel algorithms for adaptive mesh refinement," *SIAM Journal on Scientific Computing*, vol. 18, no. 3, pp. 686–708, 1997.

[48] S. V. Joshi, "Adaptive mesh refinement in openfoam with quantified error bounds and support for arbitrary cell-types," Thesis, Technical University of Munich, Germany, 2016.

[49] Y. Kallinderis, "A 3-d finite-volume method for the navier-stokes equations with adaptive hybrid grids," *Applied Numerical Mathematics*, vol. 20, no. 4, pp. 387–406, 1996.

[50] Y. Kallinderis and A. Vidwans, "Generic parallel adaptive-grid navier–stokes algorithm," *AIAA Journal*, vol. 32, no. 1, pp. 54–61, 1994.

[51] Y. Kallinderis and P. Vijayan, "Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes," *AIAA Journal*, vol. 31, no. 8, pp. 1440–1447, 1993.

[52] J. Karlsson, "Implementing anisotropic adaptive mesh refinement in openfoam," Thesis, Chalmers University of Technology, Sweden, 2012.

[53] M. Khosravi and M. Javan, "Three-dimensional features of the lateral thermal plume discharge in the deep cross-flow using dynamic adaptive mesh refinement," *Theoretical and Computational Fluid Dynamics*, vol. 36, no. 3, pp. 405–422, 2022.

[54] K. M. T. Kleefsman, G. Fekken, A. E. P. Veldman, B. Iwanowski, and B. Buchner, "A volume-of-fluid based simulation method for wave impact problems," *Journal of Computational Physics*, vol. 206, no. 1, pp. 363–393, 2005.

[55] A. Knüpfer et al., "Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir," in *Tools for High Performance Computing 2011: Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing, September 2011, ZIH, Dresden*, Springer, 2012, pp. 79–91.

[56] G. Kumar and A. G. Nair, "Dominant balance-based adaptive mesh refinement for incompressible fluid flows," *arXiv preprint arXiv:2411.02677*, 2024.

[57] C.-W. Kuo and M. F. Trujillo, "An analysis of the performance enhancement with adaptive mesh refinement for spray problems," *International Journal of Multiphase Flow*, vol. 140, p. 103 615, 2021.

[58] C. Lapointe et al., "Efficient simulation of turbulent diffusion flames in openfoam using adaptive mesh refinement," *Fire Safety Journal*, vol. 111, p. 102 934, 2020.

[59] K. D. Lee, J. M. Loellbach, and M. S. Kim, "Adaptive control of grid quality for computational fluid dynamics," *Journal of Aircraft*, vol. 28, no. 10, pp. 664–669, 1991.

[60] L.-m. Li et al., "Large eddy simulation of cavitating flows with dynamic adaptive mesh refinement using openfoam," *Journal of Hydrodynamics*, vol. 32, pp. 398–409, 2020.

[61] S. Li, "Comparison of refinement criteria for structured adaptive mesh refinement," *Journal of Computational and Applied Mathematics*, vol. 233, no. 12, pp. 3139–3147, 2010.

[62] A. Liapi et al., "Adaptive grid refinement for high-order finite volume simulations of unsteady compressible and turbulent flows," *International Journal of Computational Fluid Dynamics*, vol. 38, no. 2-3, pp. 155–178, 2024.

[63] D. K. Lilly, "A proposed modification of the germano subgrid-scale closure method," *Physics of Fluids A: Fluid Dynamics*, vol. 4, pp. 633–635, 1992.

[64] R. Löhner, K. Morgan, and O. C. Zienkiewicz, "Adaptive grid refinement for the euler and compressible navier-stokes equations," *NASA STI/Recon Technical Report A*, vol. 85, p. 16 108, 1984.

[65] D. A. Lyn, S. Einav, W. Rodi, and J.-H. Park, "A laser-doppler velocimetry study of ensemble-averaged characteristics of the turbulent near wake of a square cylinder," *Journal of Fluid Mechanics*, vol. 304, pp. 285–319, 1995.

[66] G. Maragkos, E. Funk, and B. Merci, "Analysis of adaptive mesh refinement in a turbulent buoyant helium plume," *International Journal for Numerical Methods in Fluids*, vol. 94, no. 9, pp. 1398–1415, 2022.

[67] F. R. Menter, "Two-equation eddy-viscosity turbulence models for engineering applications," *AIAA Journal*, vol. 32, no. 8, pp. 1598–1605, 1994.

[68] T. Miller, P. Aref, M. Ghoreyshi, A. Jirasek, and R. Greenwood, "Adaptive mesh refinement for computing unsteady ship air wakes," in *AIAA Aviation 2019 Forum*, 2019, p. 3031.

[69] M. Minguez, C. Brun, R. Pasquetti, and E. Serre, "Experimental and high-order les analysis of the flow in the near-wall region of a square cylinder," *International Journal of Heat and Fluid Flow*, vol. 32, no. 3, pp. 558–566, 2011.

[70] R. D. Moser, J. Kim, and N. N. Mansour, "Direct numerical simulation of turbulent channel flow up to re = 590," *Physics of Fluids*, vol. 11, no. 4, pp. 943–945, 1999.

[71] T. Mukha, S. Rezaeiravesh, and M. Liefvendahl, "A library for wall-modelled large-eddy simulation based on openfoam technology," *Computer Physics Communications*, vol. 239, pp. 204–224, 2019.

[72] J.-D. Müller and M. Giles, "Solution adaptive mesh refinement using adjoint error analysis," in *15th AIAA Computational Fluid Dynamics Conference*, AIAA, 2001, p. 2550.

[73] S. Muzaferija and D. Gosman, "Finite-volume cfd procedure and adaptive error control strategy for grids of arbitrary topology," *Journal of Computational Physics*, vol. 138, no. 2, pp. 766–787, 1997.

[74] M. Nemec, M. Aftosmis, and M. Wintzer, "Adjoint-based adaptive mesh refinement for complex geometries," in *46th AIAA Aerospace Sciences Meeting and Exhibit*, 2008, p. 725.

[75] B. D. Nichols and C. W. Hirt, "Methods for calculating multidimensional, transient free surface flows past bodies," in *Proceedings of the 1st International Conference on Ship Hydrodynamics*, Naval Ship Research and Development Center, Bethesda, MD, 1975, pp. 253–277.

[76] F. Nicoud and F. Ducros, "Subgrid-scale stress modelling based on the square of the velocity gradient tensor," *Flow, Turbulence and Combustion*, vol. 62, no. 3, pp. 183–200, 1999.

[77] M. L. Norman, "The impact of AMR in numerical astrophysics and cosmology," in *Adaptive Mesh Refinement–Theory and Applications: Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3–5, 2003*, Springer, 2005, pp. 413–430.

[78] O. Obiols-Sales, A. Vishnu, N. Malaya, and A. Chandramowlishwaran, "Adarnet: Deep learning predicts adaptive mesh refinement," in *Proceedings of the 52nd International Conference on Parallel Processing*, 2023, pp. 524–534.

[79] S. S. Ochs and R. G. Rajagopalan, "An adaptively refined quadtree grid method for incompressible flows," *Numerical Heat Transfer, Part B: Fundamentals*, vol. 34, no. 4, pp. 379–400, 1998.

[80] A. A. Patel and M. Safdari, "Smart adaptive mesh refinement with nemosys," in *AIAA Scitech 2021 Forum*, American Institute of Aeronautics and Astronautics, 2021.

[81] U. Piomelli and E. Balaras, "Wall-layer models for large-eddy simulations," *Annual Review of Fluid Mechanics*, vol. 34, no. 1, pp. 349–374, 2002.

[82] T. Plewa, T. J. Linde, and V. G. Weirs, Eds., *Adaptive Mesh Refinement–Theory and Applications: Proceedings of the Chicago Workshop on Adaptive Mesh Refinement Methods, Sept. 3–5, 2003*. Springer, 2005.

[83] S. B. Pope, "Turbulent flows," *Measurement Science and Technology*, vol. 12, no. 11, pp. 2020–2021, 2001.

[84] S. Prakash, "Adaptive mesh refinement for finite element flow modeling in complex geometries," Ph.D. dissertation, University of Toronto, 1999.

[85] T. foam-extend Project, *Foam-extend*, `https://sourceforge.net/projects/foam-extend/`, Accessed: 2025-04-29, Apr. 2025.

[86] X. Qi, Y. Yang, L. Tian, Z. Wang, and N. Zhao, "A parallel methodology of adaptive cartesian grid for compressible flow simulations," *Advances in Aerodynamics*, vol. 4, no. 1, p. 21, 2022.

[87] J. J. Quirk, "An adaptive grid algorithm for computational shock hydrodynamics," Ph.D. dissertation, Cranfield Institute of Technology, United Kingdom, 1991.

[88] J. J. Quirk and U. R. Hanebutte, "A parallel adaptive mesh refinement algorithm," Los Alamos National Laboratory, Tech. Rep., 1993, Technical Report.

[89] C. A. Rendleman, V. E. Beckner, M. Lijewski, W. Crutchfield, and J. B. Bell, "Parallelization of structured, hierarchical adaptive mesh refinement algorithms," *Computing and Visualization in Science*, vol. 3, pp. 147–157, 2000.

[90] D. Rettenmaier et al., "Load-balanced 2d and 3d adaptive mesh refinement in openfoam," *SoftwareX*, vol. 10, p. 100 317, 2019.

[91] R. Rossi, J. Cotela, N. M. Lafontaine, P. Dadvand, and S. R. Idelsohn, "Parallel adaptive mesh refinement for incompressible flow problems," *Computers & Fluids*, vol. 80, pp. 342–355, 2013.

[92] P. Sagaut, *Large Eddy Simulation for Incompressible Flows: An Introduction*. Springer Science & Business Media, 2005.

[93] M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, *Benchmark Computations of Laminar Flow Around a Cylinder*. Springer, 1996.

[94] F. G. Schmitt, "About boussinesq's turbulent viscosity hypothesis: Historical remarks and a direct evaluation of its validity," *Comptes Rendus Mécanique*, vol. 335, no. 9–10, pp. 617–627, 2007.

[95] A. M. Schwing, "Parallel adaptive mesh refinement for high-order finite-volume schemes in computational fluid dynamics," Ph.D. dissertation, University of Minnesota, 2015.

[96] S. Sezen and M. Atlar, "An alternative vorticity based adaptive mesh refinement (v-amr) technique for tip vortex cavitation modelling of propellers using cfd methods," *Ship Technology Research*, vol. 69, no. 1, pp. 1–21, 2022.

[97] A. Sikirica, L. Grbčić, M. Alvir, and L. Kranjčević, "Computational efficiency assessment of adaptive mesh refinement for turbulent jets in crossflow," *Mathematics*, vol. 10, no. 4, p. 620, 2022.

[98] J. Smagorinsky, "General circulation experiments with the primitive equations: I. the basic experiment," *Monthly Weather Review*, vol. 91, no. 3, pp. 99–164, 1963.

[99] R. Teyssier, "Cosmological hydrodynamics with adaptive mesh refinement – a new high resolution code called ramses," *Astronomy & Astrophysics*, vol. 385, no. 1, pp. 337–364, 2002.

[100] C. T. Traxler, "An algorithm for adaptive mesh refinement in n dimensions," *Computing*, vol. 59, pp. 115–137, 1997.

[101] T.-H. Un and S. Navarro-Martinez, "Stochastic fields with adaptive mesh refinement for high-speed turbulent combustion," *Combustion and Flame*, vol. 272, p. 113 897, 2025.

[102] B. Vanbersel et al., "A systematic adaptive mesh refinement method for large eddy simulation of turbulent flame propagation," *Flow, Turbulence and Combustion*, vol. 112, no. 4, pp. 1127–1160, 2024.

[103] D. A. Venditti and D. L. Darmofal, "Adjoint error estimation and grid adaptation for functional outputs: Application to quasi-one-dimensional flow," *Journal of Computational Physics*, vol. 164, no. 1, pp. 204–227, 2000.

[104] E. D. Villiers, "The potential of large eddy simulation for the modeling of wall-bounded flows," Ph.D. dissertation, Imperial College of Science, Technology and Medicine, 2006.

[105] R. Vilsmeier and D. Hänel, "Adaptive methods on unstructured grids for euler and navier-stokes equations," *Computers & Fluids*, vol. 22, no. 4–5, pp. 485–499, 1993.

[106] J. Wackers, "Adaptivity for complex flows," Ph.D. dissertation, Université de Nantes, 2019.

[107] J. Wackers et al., "Adaptive grid refinement for ship resistance computations," *Ocean Engineering*, vol. 250, p. 110 969, 2022.

[108] F. Wang et al., "Cpu ray tracing of tree-based adaptive mesh refinement data," in *Computer Graphics Forum*, Wiley Online Library, vol. 39, 2020, pp. 1–12.

[109] Y. Wang and W. Ge, "Simulation of fluid-structure interaction using the boundary data immersion method with adaptive mesh refinement," *International Journal for Numerical Methods in Fluids*, vol. 96, no. 7, pp. 1156–1169, 2024.

[110] Z. Wang, L. Li, H. Cheng, and B. Ji, "Numerical investigation of unsteady cloud cavitating flow around the clark-y hydrofoil with adaptive mesh refinement using openfoam," *Ocean Engineering*, vol. 206, p. 107 349, 2020.

[111] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby, "A tensorial approach to computational continuum mechanics using object-oriented techniques," *Computers in Physics*, vol. 12, no. 6, pp. 620–631, 1998.

[112] W. Ying and C. S. Henriquez, "Adaptive mesh refinement and adaptive time integration for electrical wave propagation on the purkinje system," *BioMed Research International*, vol. 2015, no. 1, p. 137 482, 2015.

[113] N. Zander, T. Bog, M. Elhaddad, F. Frischmann, S. Kollmannsberger, and E. Rank, "The multi-level hp-method for three-dimensional problems: Dynamically changing high-order mesh refinement with arbitrary hanging nodes," *Computer Methods in Applied Mechanics and Engineering*, vol. 310, pp. 252–277, 2016.

[114] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, 2005.

[115] D. Zuzio and J. L. Estivalezes, "An efficient block parallel amr method for two phase interfacial flow simulations," *Computers & Fluids*, vol. 44, no. 1, pp. 339–357, 2011.

LIST OF FIGURES

LIST OF TABLES

CURRICULUM VITAE

Ante Sikirica was born on 20 October 1991 in Rijeka, Croatia. He obtained his Master's degree in Mechanical Engineering from the Faculty of Engineering, University of Rijeka, in 2018. Since 2019, he has been a PhD candidate in Fundamental Technical Sciences at the same institution. That same year, he briefly served as an assistant at the Faculty of Engineering before joining the Centre for Advanced Computing and Modelling at the University of Rijeka, where he has held the position of assistant since. He also contributes in his capacity as an external associate at the Faculty of Engineering. His research focuses on the application of high-performance computing, with an emphasis on computational fluid dynamics, machine learning, and optimisation methods. He has contributed to numerous national and European research projects and is the author or co-author of over 20 scientific publications, 16 of which were published in journals ranked in the top quartile (Q1) by Web of Science. Mr Sikirica has received several honours for his work, including the University of Rijeka Rector's Award for Excellence (2024), two awards from the University of Rijeka Foundation (2023) for scientific achievement and knowledge transfer, and four Dean's Awards from the Faculty of Engineering.

LIST OF PUBLICATIONS

Scientific papers in peer-reviewed journals:

1. Jakac, K., Lanča, L., Sikirica, A., and Ivić, S. 2024. Approximation of sea surface velocity field by fitting surrogate two-dimensional flow to scattered measurements. Applied ocean research, 153, 104246.
2. Sikirica, A., Lučin, I., Alvir, M., Kranjčević, L., and Čarija, Z. 2024. Computationally efficient optimisation of elbow-type draft tube using neural network surrogates. Alexandria Engineering Journal, 90, 129-152.
3. Alvir, M., Grbčić, L., Sikirica, A., and Kranjčević, L. 2023. Reconstruction and analysis of negatively buoyant jets with interpretable machine learning. Marine pollution bulletin, 190, 114881.
4. Rak, A., Grbčić, L., Sikirica, A., and Kranjčević, L. 2023. Experimental and LBM analysis of medium-Reynolds number fluid flow around NACA0012 airfoil. International journal of numerical methods for heat & fluid flow, 33(5), 1955-1980.
5. Sikirica, A., Grbčić, L., and Kranjčević, L. 2023. Machine learning based surrogate models for microchannel heat sink optimization. Applied thermal engineering, 222, 119917.
6. Alvir, M., Grbčić, L., Sikirica, A., and Kranjčević, L. 2022. OpenFOAM-ROMS nested model for coastal flow and outfall assessment. Ocean engineering, 264, 112535.
7. Grbčić, L., Družeta, S., Mauša, G., Lipić, T., Vukić Lušić, D., Alvir, M., Lučin, I., Sikirica, A., Davidović, D., Travaš, V., Kalafatović, D., Pikelj, K., Fajković, H., Holjević, T. and Kranjčević, L. 2022. Coastal water quality prediction based on machine learning with feature interpretation and spatio-temporal analysis. Environmental Modelling & Software, 155, 105458.
8. Ivić, S., Sikirica, A., and Crnković, B. 2022. Constrained multi-agent ergodic area surveying control based on finite element approximation of the potential field. Engineering applications of artificial intelligence, 116, 105441.

9. Lučin, I., Družeta, S., Mauša, G., Alvir, M., Grbčić, L., Vukić Lušić, D., Sikirica, A., and Kranjčević, L. 2022. Predictive modeling of microbiological seawater quality in karst region using cascade model. Science of the total environment, 851, 158009.
10. Lučin, I., Sikirica, A., Šiško Kuliš, M., and Čarija, Z. 2022. Investigation of efficient optimization approach to the modernization of Francis turbine draft tube geometry. Mathematics, 10(21), p.4050.
11. Sikirica, A., Grbčić, L., Alvir, M., and Kranjčević, L. 2022. Computational Efficiency Assessment of Adaptive Mesh Refinement for Turbulent Jets in Crossflow. Mathematics, 10(4), p.620.
12. Bukmir, R. P., Paljevic, E., Braut, A., Sikirica, A., Carija, Z., Brekalo Prso, I., and Anic, I. 2021. Influence of operator experience on vertical force during instrumentation using Neoniti rotary files. Giornale italiano di endodonzia, 35(1).
13. Grbčić, L., Kranjčević, L., Lučin, I., and Sikirica, A. 2021. Large Eddy Simulation of turbulent fluid mixing in double-tee junctions. Ain Shams Engineering Journal, 12(1), 789-797.
14. Lučin, I., Lučin, B., Čarija, Z., and Sikirica, A. 2021. Data-driven leak localization in urban water distribution networks using big data for random forest classifier. Mathematics, 9(6), p.672.
15. Travaš, V., Kranjčević, L., Družeta, S., Holjević, T., Lučin, I., Alvir, M., Grbčić, L., and Sikirica, A. 2021. Model gibanja čestica mikroplastike u nehomogenom i laminarnom polju brzine. Hrvatske vode, 29(117), 201-213.
16. Čarija, Z., Ledić, F., Sikirica, A., and Niceno, B. 2020. CFD study of the PTS experiment in ROCOM test facility. Nuclear Engineering and Technology, 52(12), 2803-2811.
17. Sikirica, A., Čarija, Z., Lučin, I., Grbčić, L., and Kranjčević, L. 2020. Cavitation model calibration using machine learning assisted workflow. Mathematics, 8(12), p.2107.
18. Sikirica, A., Čarija, Z., Kranjčević, L., and Lučin, I. 2019. Grid type and turbulence model influence on propeller characteristics prediction. Journal of marine science and engineering, 7(10), 374.

Conference papers:

1. Lučin, I., Alvir, M., Sikirica, A., Družeta, S., Travaš, V., and Kranjčević, L. 2023. Remote Sensing Localization of Submerged Groundwater Discharges in Bathing Areas. 40th IAHR World Congress, 1260-1267.

2. Alvir, M., Grbčić, L., Sikirica, A., and Kranjčević, L. 2022. Numerical Modeling of Inclined Buoyant Jets for Different Flow Conditions. Pomorski zbornik, (4), 77-86.
3. Janeš, G., Sikirica, A., Grbčić, L., and Kranjčević, L. 2022. MPI associated scalability of open-source CFD codes for oil spill assessment. Pomorski zbornik, (4), 67-75.
4. Sikirica, A., Lučin, I., Čarija, Z., and Lučin, B. 2020. CFD Analysis of Marine Propeller Configurations in Cavitating Conditions. Pomorski zbornik, (3), 251-264.