A Rapid Steady Solver for the Navier-Stokes Equations

Mark Andrew George

A thesis submitted in fulfilment
of the requirements of the degree of
Doctor of Philosophy

Faculty of Engineering
The University of Sydney

April 21, 2026

This research reported in this thesis was supported
by the award of a Research Training Program scholarship
to the Ph.D. Candidate.

Acknowledgements

Completing this journey would not have been possible without my family. They have consistently provided unwavering support, regardless of my successes or failures, and I owe them an immense debt of gratitude.

I would like to express my sincerest gratitude to my lead supervisor, Professor Steven Armfield, whose guidance and support throughout my PhD have led to many valuable opportunities for growth. I am deeply thankful for the many inspiring discussions we have shared over the past four years.

I would also like to extend my special thanks to my supervisor, Associate Professor Nicholas Williamson, for his guidance and advice beyond the technical aspects of my work, particularly in navigating the broader academic and university environment.

My time here was enriched by my fellow PhD students, who provided a harmonious and productive work environment. While our work-related discussions were valuable, it was often the least intellectual conversations that proved the most meaningful. The friendships formed along the way made the experience far more enjoyable, and for these I am especially grateful.

Finally I would also like to express appreciation to Alexander Hill and Omar Ilaya from the Defence Science and Technology Group for their fruitful discussion and useful inputs throughout this work.

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Mark George

April 21, 2026

Use of Generative AI Statement

No content produced by generative AI tools has been used in the preparation of this thesis.

Mark George
April 21, 2026

Authorship Attribution Statement

This thesis contains material from the previously published work:

Journal Articles

M.A. George, N. Williamson, S. W. Armfield. A coupled block implicit solver for the incompressible Navier–Stokes equations on collocated grids. Computers & Fluids 284 (2024) 106426.

M.A. George, N. Williamson, S. W. Armfield. Mass-conserving ghost cell immersed boundary method with multigrid for coupled Navier-Stokes solvers. Journal of Computational Physics (2025) 114276.

M.A. George, N. Williamson, S. W. Armfield. Coupled FAS Multigrid for the Incompressible Navier-Stokes Equations on Collocated Grids. Available at SSRN 5460400. (Under review in Computers & Fluids) (2025).

Conference Presentations and Papers

M.A. George, N. Williamson, S. W. Armfield. A fully coupled block implicit solver for the incompressible navier-stokes equations on collocated grids. In Twentieth International Conference on Flow Dynamics, pages 538 – 541. Institute of Fluid Science, Tohoku University, 2023.

M.A. George, N. Williamson, S. W. Armfield. A fast coupled solver for the steady navier-stokes equations on rectilinear grids. In Twelfth International Conference on Computational Fluid Dynamics (ICCFD12), 2024.

M.A. George, N. Williamson, S. W. Armfield. Mass-conserving immersed boundary method for coupled incompressible solvers on collocated finite volume grids. The 22nd Biennial Computational Techniques and Applications Conference, 2024.

M.A. George, N. Williamson, S. W. Armfield. Mass-conserving immersed boundary method for coupled incompressible solvers on collocated finite volume grids. In 24th Australasian Fluid Mechanics Conference (AFMC2024), 2024.

M.A. George, N. Williamson, S. W. Armfield. A rapid RANS solver for wind field simulations in urban environments. In Division of Fluid Dynamics Annual Meeting 2025.

Mark George
April 21, 2026

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Professor Steven Armfield
April 21, 2026

A/Professor Nicholas Williamson
April 21, 2026

Abstract

Despite the significant advances in algorithms and computer hardware over the years, Computational Fluid Dynamics (CFD) models remain to be very time consuming to compute. For applications where fast turnaround time is required such as in hazard prediction models for the dispersion of airborne contaminants in urban environments, the calculation of the flow field can be a significant bottleneck, and current CFD methods are still too computationally expensive. As a result, state-of-the-art approaches for this application have resorted to diagnostic methods, which although fast, do not contain the complete flow physics. This work aims to bridge the gap between CFD methods and diagnostic approaches through the development of a highly efficient steady-state CFD method. This is through the development of three fundamental elements: an efficient coupled solver, a mass conservative immersed boundary method, and the use of a simple zero-equation eddy viscosity turbulence model.

In this work, a fully coupled matrix-free method for solving the incompressible steady-state Navier–Stokes equations on a collocated finite volume grid is introduced and used within an Full Approximation Scheme (FAS) multigrid method. To the authors knowledge, this is the first of such a scheme. The solver is implemented on a rectilinear mesh, meaning simple, fast data structures can be used. Verification and validation was performed for a number of laminar steady-state cases, and performance was compared to the coupled solver in ANSYS Fluent and the SIMPLE scheme in OpenFOAM. Up to two orders of magnitude speedup was observed on the same hardware for these cases, and linear scaling with problem size. Strategies for parallelising the method are also tested and discussed.

The immersed boundary method was used to obtain the flow field over complex geometries. Although simple to implement, ghost cell immersed boundary methods in their basic form do not conserve mass globally, even in a mass-conservative finite volume framework where local mass conservation is satisfied in the fluid domain. Reconstruction near solid boundaries with corners is also difficult. Furthermore, when used with coupled solvers on collocated grids, correct implementation of momentum weighted interpolation at the boundaries is not straightforward. These issues were solved here by combining a directional ghost cell method with a weighted face flux correction based on the global mass continuity error. The method is simple to implement and only requires the addition of source terms to the discrete equations. The method combined with the coupled FAS solver was

verified and validated for a number of canonical steady incompressible flows, and excellent performance and efficiency is demonstrated with linear scaling with problem size.

Finally, a zero-equation eddy-viscosity turbulence model, designed specifically for urban geometries was implemented into the code to solve the Reynolds-Averaged Navier-Stokes (RANS) equations. Despite their simplicity of zero-equation models, the significant savings in cost make them a useful tool for applications where fast runtime is a priority. Using the zero-equation model and the present solver method, validation tests using three cases from the University of Hamburg CEDVAL experimental reference data set were carried out, and comparisons for performance and accuracy where made against the $k - \varepsilon$ model with the ANSYS Fluent coupled solver, and the OpenFOAM SIMPLE scheme, both with the same unstructured body fitted mesh. For cases where the geometry was made up of simple surface mounted prisms, the accuracy of the zero equation model was comparable to the $k - \varepsilon$ models, however for more complicated building structures, the $k - \varepsilon$ models where approximately 25% better in terms of the average error from experimental data. However, on the same hardware, the present method solved all problems in less than 1 min, while the $k - \varepsilon$ models took between 1 and 3 hrs for a mesh with the same number of cells. This demonstrates that the present method outperforms standard CFD methods using the $k - \varepsilon$ by over 2 orders of magnitude in terms of runtime, with losses of accuracy on the order of 25% according to the cases tested here.

Contents

Acknowledgements *iii*

Abstract *ix*

Contents *xi*

List of Figures *xiv*

List of Tables *xxi*

1 Introduction **1**

1.1 Background and motivation 1

1.2 Objectives 6

1.3 Outline and Approach 8

2 Discretisation of the Navier-Stokes Equations **11**

2.1 Introduction 11

2.2 Finite Volume Method 11

2.3 Linearisation 13

2.4 Discretisation of Terms 14

2.4.1 Diffusion 14

2.4.2 Advection 15

2.4.3 Pressure Gradient and Velocity Divergence 17

2.5 Momentum Weighted Interpolation 17

2.6 Boundary Condition Treatment 18

2.6.1 Dirichlet Boundary Conditions 19

2.6.2 Neumann Boundary Conditions 19

2.6.3 Extrapolated Boundary Conditions 20

2.6.4 Periodic Boundary Conditions 20

2.6.5 General Remarks 21

2.7 Finite Volume Stencil on A Rectilinear Mesh 22

2.8 Convergence Residuals 24

2.9 Summary 26

3 The Coupled Solver **27**

3.1 Introduction 27
3.2 The Coupled Smoother Technique 30
3.3 Boundary Condition Treatment and Sweeping 34
3.4 Parallelism 36
3.5 FAS Multigrid 40
3.5.1 The FAS Algorithm 40
3.5.2 Multigrid Cycles 41
3.5.3 Calculation of Discrete Operators 42
3.5.4 Smoothing 43
3.5.5 Grid Hierarchy and Transfer Operators 44
3.5.6 Under-relaxation 45
3.6 Implementation Details 46
3.7 Numerical Tests 47
3.7.1 Lid Driven Cavity 48
3.7.2 Backwards Facing Step 54
3.7.3 Discussion 56
3.7.4 Parallel Performance 62
3.8 Summary 70

4 Immersed Boundary Method **73**

4.1 Introduction 73
4.2 Challenges for Coupled Solvers 74
4.3 The Immersed Boundary Technique 78
4.3.1 Reconstruction 78
4.3.2 Mass Flux Correction 80
4.3.3 Ghost Cell Extrapolation 82
4.3.4 MWI Ghost Cell Determination 82
4.3.5 Source Term Construction 83
4.3.6 Boundary Conditions 84
4.3.7 Multigrid Transfer Operators 85
4.3.8 General Remarks 86

4.4 Implementation Details . . . 87
4.5 Numerical Tests . . . 87
4.5.1 Sphere in Lid Driven Cavity . . . 88
4.5.2 Sphere in Free Stream . . . 90
4.5.3 45° Angled Surface Mounted Cube in Channel . . . 98
4.6 Summary . . . 102

5 Turbulence Modelling **103**

5.1 Introduction . . . 103
5.2 The Eddy Viscosity Model . . . 105
5.3 Discretisation of The Eddy Viscosity Model . . . 106
5.4 The Zero-Equation Model . . . 108
5.5 Implementation Details . . . 109
5.6 Numerical Tests . . . 110
5.6.1 40° Angled cube (CEDVAL A1-7) . . . 113
5.6.2 Array of Rectangular Blocks (CEDVAL B1-1) . . . 119
5.6.3 Flow Across an Intersection (CEDVAL B1-5) . . . 124
5.6.4 Discussion . . . 130
5.7 Summary . . . 133

6 Conclusions and Further Research **135**

6.1 Summary of current research . . . 135
6.2 Recommendations for future research . . . 138

References **141**

List of Figures

Chapter 1

1.1 Spectrum of main methods that can be used for the calculation of the windfield, ranging from higher accuracy and high cost, to lower accuracy and lower cost. . . . 3

Chapter 2

2.1 Staggered grid where velocities are stored at cell faces and pressure at cell centre (a), and collocated grid where both velocities and pressures are stored at cell centres (b). 12

2.2 Finite volume cell at location P with its neighbouring node labels. Lowercase letters are used to label cell faces. 13

2.3 2D rectilinear mesh with ghost cells (shown with grey dashed lines) on domain boundaries. The ghost cells have the same dimensions as the interior cell adjacent to them. 19

2.4 Periodic boundary conditions on a solution domain (a) imply that the e and w faces are shared (b), or that the field value on that face are identical. 20

3.1 Offset triad showing momentum equations and cells coupled together implicitly. Shown is the case where all momentum equations are offset in the positive coordinate direction, however the offset direction will depend on the sweeping direction. . . . 31

Chapter 3

3.2 An offset molecule at the boundary with the v-momentum equation "hanging" out of the domain boundary, allowing for the solution of the u-momentum and continuity at the boundary cells. The same concept can be extended to 3D. 34

3.3 The forward (a) and backward (b) update sweeps used to ensure that all boundary cells are solved. 36

3.4 An instance where a line is updated by sweeping a triad in the x-direction. These lines are swept in the y-direction to update the plane, and the plane is swept in the z direction to update the domain. This is then done in reverse with the molecule flipped in all directions to enforce the boundary conditions. 36

3.5 A single offset molecule (black solid lines) being solved with all with the variables required but not updated (grey dashed lines). The stencil dependence in 3D is the same. 37

3.6 Three colour ordering required to parallelise the smoother. Parallel regions can be points (a), contiguous lines (b), or contiguous planes (c). 38

3.7 FMG initialisation (a), V-cycle (b), F-cycle (c), and W-cycle (d) for a 4 level multigrid. Filled in circles indicate that the solution is solved, while hollow circles indicate that a fixed number of smoothing iterations are performed. 42

3.8 Fine grid (a) and coarsened grid (b) through agglomeration. Since there is an odd number of cells in each dimension, the cells on the right and bottom boundaries are not agglomerated. 44

3.9 Schematic diagram of 3D lid driven cubical cavity (a) and nonuniform rectilinear finite volume mesh used for 46^3 cells (b). 48

3.10 Velocity magnitude (a) and pressure (b) error norms relative to solution computed on a 320^3 uniform grid ($\Delta x = 0.003125$) for the 3D lid driven cavity on a uniform mesh with $Re = 200$. 49

3.11 Velocity magnitude contours at centre plane for 3D lid driven cavity with $Re = 200$ (a) and $Re = 1000$ (b) obtained with the present method on a 216^3 grid. 50

3.12 L_1-norm of velocity magnitude difference vs grid size between present method and other standard solvers for $Re = 200$ (a) and $Re = 1000$ (b) for 3D lid driven cavity. 51

3.13 Local continuity residual convergence history for 3D lid driven cavity on a 312^3 nonuniform grid for each multigrid cycle with $Re = 200$ (a) and $Re = 1000$ (b). . . 51

3.14 Comparison of solver times with other standard steady solvers at $Re = 200$ (a) and $Re = 1000$ (b) for the 3D lid driven cavity problem. "SG" refers to the coupled smoother on a single grid using the optimal plane and line sweeping direction. All solvers were set to solve the problem to a residual of 10^{-10}. 53

3.15 Lid driven cavity continuity residual convergence history for $Re = 200$ (a) and $Re = 1000$ (b) for different line and plane sweeping directions when solved without FAS multigrid, on a single grid. 53

3.16 Lid driven cavity continuity residual convergence history for $Re = 200$ (a) and $Re = 1000$ (b) for different line and plane sweeping directions when solved with FAS multigrid using W cycles. 54

3.17 Schematic diagram of 3D backwards facing step (a) and nonuniform rectilinear finite volume mesh used for $108 \times 54 \times 18$ cells (b). 55

3.18 Velocity magnitude contours at centre plane of backwards facing step with $Re = 100$ (a) and $Re = 200$ (b) obtained with the present method on a $492 \times 246 \times 82$ grid. . 56

3.19 L_1-norm of velocity magnitude difference vs grid size between present method and other standard solvers for $Re = 100$ (a) and $Re = 200$ (b) for 3D backwards facing step. 57

3.20 Local continuity residual convergence history for 3D backwards facing step on a $492 \times 246 \times 82$ nonuniform grid for each multigrid cycle with $Re = 100$ (a) and $Re = 200$ (b). 57

3.21 Comparison of solver times with other standard steady solvers at $Re = 100$ (a) and $Re = 200$ (b) for the 3D backwards facing step problem. "SG" refers to the coupled smoother on a single grid using the optimal plane and line sweeping direction. All solvers were set to solve the problem to a residual of 10^{-10}. 58

3.22 Backwards facing step continuity residual convergence history for $Re = 100$ (a) and $Re = 200$ (b) for different line and plane sweeping directions when solved without FAS multigrid, on a single grid. 59

3.23 Backwards facing step continuity residual convergence history for $Re = 100$ (a) and $Re = 200$ (b) for different line and plane sweeping directions when solved with FAS multigrid using W cycles. 59

3.24 Strong scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 1 (Table 3.4). 65

3.25 Strong scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=close`. 65

3.26 Strong scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=spread`. 65

3.27 Weak scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 1 (Table 3.4). 66

3.28 Weak scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=close`. 66

3.29 Weak scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=spread`. 66

3.30 Raw solver times for strong scaling test with 256^3 (a) and weak scaling tests with 128^3 cells/processor (b) for System 1 (Table 3.4). 68

3.31 Raw solver times for strong scaling test with 256^3 (a) and weak scaling tests with 128^3 cells/processor (b) for System 2 (Table 3.4) with `PROC_BIND=close`. 68

3.32 Raw solver times for strong scaling test with 256^3 (a) and weak scaling tests with 128^3 cells/processor (b) for System 2 (Table 3.4) with `PROC_BIND=spread`. 69

Chapter 4

4.1 Setting of ghost cells within the immersed boundary applies a mass flux on the cells which border the ghost cells (a), this is generally done by extrapolating in the direction normal to the immersed boundary for each ghost cell (b). 75

4.2 Reconstruction for a ghost cell is done in each coordinate direction independently, using the intercept with the immersed boundary in each direction for extrapolation. 79

4.3 Two cases need to be considered for the directional method depending on relative location of the immersed boundary to the face that borders the ghost cell and fluid cell. 79

4.4 Neumann boundary conditions can be applied by constructing the field variable at immersed boundary intersection ϕ_X in the normal direction by first interpolating onto an image point I, then extrapolating along the normal line. 85

4.5 Restriction from fine grid (grey) to coarse grid (black) near the immersed boundary. The green arrows indicate the nodes which contribute to the restriction from the fine to the coarse grid. 86

4.6 Sphere centred in lid driven cavity case used to verify order of accuracy and multigrid scaling. 88

4.7 Velocity magnitude (a) and pressure (b) error norms relative to solution computed on a $320 \times 320 \times 320$ grid ($\Delta x = 0.003125$) for the sphere inside a lid driven cavity with $Re = 200$. White cells are ghost cells. 89

4.8 Velocity magnitude contours for 3D sphere in lid driven cavity at centre plane for $Re = 1000$ of the complete flow field (a) and a close up of the sphere with cells shown for 160^3 mesh (b). 89

4.9 Solver CPU time with grid size for different Reynolds numbers for the sphere inside the 3D lid driven cavity. 90

4.10 Flow over a sphere case used for method validation. All boundary conditions except the inflow and outflow were symmetry conditions, and the sphere has diameter D. . 91

4.11 Cross-section of complete $343 \times 171 \times 171$ non-uniform mesh used to calculate drag coefficient. View is such that the inflow is on the left. The maximum cell growth ratios in each direction are 1.11301, 1.03324, and 1.03324 in the streamwise and both cross-stream directions respectively. The mesh used for the surface pressure coefficients has the same growth ratios, with the number of cells increased in each dimension. 92

4.12 Computational mesh used to calculate sphere drag coefficient (a) and surface pressure coefficient (b). Note the full computational domain is not show here. 93

4.13 Computed drag coefficient compared with experimental data of Roos and Willmarth [1]. 93

4.14 Velocity magnitude contours for flow over sphere for $Re = 100$ of the complete flow field using the reconstruction from Eq. 4.3 (a) and a close up of the sphere with cells shown (b). The white cells immediately around the sphere are ghost cells. 94

4.15 Pressure distribution around sphere for $Re = 100$ using the reconstruction from Eq. 4.3 (a) and an alternative reconstruction where $\phi_e = 0$ if the AB face lies within the solid (b) with a close up of pressure spikes near the boundary for both. The white cells immediately around the sphere are ghost cells. 95

4.16 Extrapolated sphere surface pressure coefficient for a range of Reynolds numbers for two different reconstructions. Validation data is from Johnson [2]. Reconstruction 1 uses Eq. 4.3 from Section 4.3.1, while reconstruction 2 is the case where $\phi_e = 0$ if the AB face lies within the solid. 96

4.17 Flow over 45° angled cube in channel. All boundary conditions except ones marked are walls. 98

4.18 Cross-sectional view from the side (a) and top (b) of complete $125 \times 50 \times 53$ coarse coarse mesh used for the 45° angled cube case. The maximum cell growth ratios are 1.09167, 1.16819, and 1.04023 in the streamwise, vertical, and cross-stream directions respectively. The fine mesh has the same growth ratios, with the number of cells increased in each dimension. 99

4.19 Top view of coarse mesh with $\sim 3.3 \times 10^5$ cells (a) fine mesh with $\sim 3.5 \times 10^6$ cells (b) near the solid cube. Note the full computational domain is not show here. . . . 99

4.20 Velocity profile comparison of present method for a fine and coarse grid with a fine unstructured grid in OpenFOAM along the centreline of the domain (a) and off-centre, halfway between the block and wall boundary (b). 100

4.21 Local and global mass continuity convergence history for angled block case with $Re = 50$ on the fine mesh without (a) and with (b) mass flux correction. 101

Chapter 5

5.1 Problem setup for wind tunnel 40° angled cube case (CEDVAL A1-7). The cube has side length 0.125 m, and all domain boundaries except for the inflow, symmetry, and outflow as marked are walls. 114

5.2 Side (a) and top (b) view of computational mesh used for 40° angled cube case (CEDVAL A1-7) in present method with $\sim 1.06 \times 10^6$ cells. 114

5.3 Comparison of velocity profiles with experiment for the 40° angled cube case (CEDVAL A1-7) in the $x - z$ plane with $y = 0$ m (a) and $x - y$ plane with $z = 0.05$ m (b). 115

5.4 Velocity magnitude in the $x - z$ plane for the 40° angled cube case (CEDVAL A1-7) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c). 116

5.5 Pressure in the $x - z$ plane for the 40° angled cube case (CEDVAL A1-7) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c). 117

5.6 Comparison of all experimental data points with methods tested for the 40° angled cube case (CEDVAL A1-7). Points with perfect agreement with experiment lie on the diagonal line. Also indicated lines which show the boundary of 10% and 30% error. 118

5.7 Problem setup for wind tunnel array of rectangular blocks case (CEDVAL B1-1). Each block has dimensions 0.1 m $\times$ 0.15 m $\times$ 0.125 m, and all domain boundaries except for the inflow, symmetry, and outflow as marked are walls. 120

5.8 Side (a) and top (b) view of computational mesh used for the array of rectangular blocks case (CEDVAL B1-1) in present method with $\sim 2.06 \times 10^6$ cells. 120

5.9 Comparison of velocity profiles with experiment for the array of rectangular blocks case (CEDVAL B1-1) in the $x - z$ plane with $y = -0.05$ m (a) and $x - y$ plane with $z = 0.0625$ m (b). 121

5.10 Velocity magnitude in the $x - z$ plane for the array of rectangular blocks case (CEDVAL B1-1) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c). 122

5.11 Pressure in the $x - z$ plane for the array of rectangular blocks case (CEDVAL B1-1) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c). 123

5.12 Comparison of all experimental data points with methods tested for the array of rectangular blocks case (CEDVAL B1-1). Points with perfect agreement with experiment lie on the diagonal line. Also indicated lines which show the boundary of 10% and 30% error. 124

5.13 Problem setup for wind tunnel flow over an intersection case (CEDVAL B1-7). The buildings have a height of 0.06 m without the pitched roof, and 0.09 m with the pitched roof, and all domain boundaries except for the inflow, symmetry, and outflow as marked are walls. The buildings are rotated 29° relative to the inflow direction. . 126

5.14 Side (a) and top (b) view of computational mesh used for the flow over an intersection case (CEDVAL B1-5) in present method with $\sim 3.15 \times 10^6$ cells. The (x, y) axis is used in the simulation and is aligned with the inflow direction, while the (x', y') axis is aligned with the geometry. 126

5.15 Comparison of velocity profiles with experiment for the flow over an intersection case (CEDVAL B1-5)in the $x' - z$ plane with $y' = -0.06$ m (a) $y' = -0.155$ m (b). u' is the velocity component in the x' direction. 127

5.16 Comparison of velocity profiles with experiment for the flow over an intersection case (CEDVAL B1-5) in the $x' - y'$ plane with $z = 0.03$ m (a) $z = 0.05$ m (b). u' is the velocity components in the x' direction. 128

5.17 Velocity magnitude in the $x - z$ plane for the flow over an intersection case (CEDVAL B1-5) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c). 129

5.18 Pressure in the $x - z$ plane for the flow over an intersection case (CEDVAL B1-5) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c). 130

5.19 Comparison of all experimental data points with methods tested for the flow over an intersection case (CEDVAL B1-5). Points with perfect agreement with experiment lie on the diagonal line. Also indicated lines which show the boundary of 10% and 30% error. 131

List of Tables

Chapter 3

3.1 Solver time vs. grid size for each multigrid cycle for the 3D lid driven cavity at $Re = 200$ (a) and $Re = 1000$ (b). All cases were solved to a residual of 10^{-14}. . . . 52

3.2 Solver time vs. grid size for each multigrid cycle for the 3D backwards facing step at $Re = 100$ (a) and $Re = 200$ (b). All cases were solved to a residual of 10^{-14}. . . 58

3.3 Percentage of total solver time spent in different parts of the solver for the 3D lid driven cavity with $Re = 200$ on a 216^3 nonuniform grid. "Other" refers to all other operations that occur during a solver such as convergence checks, and writing data to file. "Multigrid Operations" accounts for all operations that arise due to the use of an FAS scheme including restriction, prolongation, calculation of residuals, and application of the coarse grid correction. 61

3.4 Specifications of the two computer systems used in parallel scaling tests. 62

3.5 Timing results for solution update sweeps performed to solve the 3D lid driven cavity problem on a 256^3 grid with System 1 (Table 3.4) using a single processor for each colour ordering in Figure 3.6. 69

Chapter 4

4.1 Raw solver CPU times with grid size for different Reynolds numbers for the sphere inside the 3D lid driven cavity as shown in Figure 4.9. 91

4.2 L_1-norm of C_P error for present method against validation data of Johnson [2] for reconstruction 1 (a) and reconstruction 2 (b), with and without the mass flux correction. A positive percentage improvement indicates that the solution is more accurate with the mass flux correction. 97

4.3 L_1-norm of velocity magnitude error for present method on the coarse and fine grid against fine unstructured grid in OpenFOAM both with and without the mass flux correction. A positive percentage improvement indicates that the solution is more accurate with the mass flux correction. 101

Chapter 5

5.1 Model parameters used for zero-equation model (Eq. 5.17) for each test case. . . . 111
5.2 Average error from all experimental data points of each method tested and solver time to reach a residual of 10^{-6} with $\sim 1.06 \times 10^6$ cells for the 40° angled cube case (CEDVAL A1-7). . . . 118
5.3 Number of multigrid cycles and solver time for present method to reach a residual of 10^{-10} for the 40° angled cube case (CEDVAL A1-7) with each plane and line update direction. . . . 119
5.4 Average error from all experimental data points of each method tested and solver time to reach a residual of 10^{-6} with $\sim 2.06 \times 10^6$ cells for the array of rectangular blocks case (CEDVAL B1-1). . . . 124
5.5 Number of multigrid cycles and solver time for present method to reach a residual of 10^{-10} for the array of rectangular blocks case (CEDVAL B1-1) with each plane and line update direction. . . . 125
5.6 Average error from all experimental data points of each method tested and solver time to reach a residual of 10^{-6} with $\sim 3.15 \times 10^6$ cells for the flow over an intersection case (CEDVAL B1-7). . . . 129
5.7 Number of multigrid cycles and solver time for present method to reach a residual of 10^{-10} for the flow over an intersection case (CEDVAL B1-7) with each plane and line update direction. . . . 130

CHAPTER 1

Introduction

1.1 Background and motivation

The field of Computational Fluid Dynamics (CFD) has developed greatly since its emergence during the mid 20th century and has been a truly invaluable tool in many areas from fundamental research to engineering application. Unfortunately CFD models are still computationally expensive and can be very time consuming to compute despite the significant advances in algorithms and computer hardware over the years. One application in particular where fast turnaround time is critical is in hazard prediction models for the dispersion of airborne contaminants in urban environments. Given appropriate boundary conditions and geometry, these models aim to predict where the contaminant will be and its concentration after a release. It is essential that these models are easy to set up and and fast running so that an appropriate response to emergency scenarios can be made quickly. In these models, the calculation of the wind field is the most computationally time consuming part.

What makes wind field calculations in urban environments difficult in particular is the presence of complex building geometries and the wide variety of turbulent flow features that can occur as a result. Brown [3] highlights many of these flow features and their modelling challenges such as the intermittent channelling of flow in street canyons, vortex formation around buildings and in street canyons, and the impact of small scale building features such as roofs and air vents among others. Furthermore, flows in urban environments are highly unsteady, and have unpredictable time varying behaviour. The variability in geometry alone between cities and within cities adds an additional layer of complexity. Of course, it is easy to see that this is more than just a purely CFD problem. Other challenges include obtaining and preparing an accurate geometry of the city [4–6], specifying appropriate boundary conditions from measurements and other simulations [7–10], the inclusion of other physical processes within the model [11–14],

and post processing the obtained data [15, 16]. While these are all equally as important, this work focuses on the fundamental method used to obtain the wind field rapidly.

The operating requirements of the model will dictate what type of physical assumptions will be made and ultimately the implementation of the algorithm. In particular, these requirements include the type of computer hardware the model will be run on, the time frame in which the results are required, and the problem size that is required to be solved. While in the present case these requirements are not strictly defined, the methods developed in this thesis aim to create a model that can obtain a flow field solution around a built environment with an area on the order of 1 km^2 within a time frame that is on the order of minutes using high end consumer grade computer hardware at the time of writing. Such a model is not only useful for emergency response situations, but also for applications where many solutions to a large space of parameters is needed, such as in parameter studies and optimisation routines.

A wide array of techniques have been developed for CFD, ranging from finite volume methods, finite element methods, Lattice Boltzmann Methods [17, 18], and mesh free methods such as Smoothed-Particle Hydrodynamics [19, 20] to name a few. Of these, finite volume methods are the most widely adopted due to their intuitiveness, inherit ability to satisfy the conservation equations of fluid dynamics, and ability to handle complex geometries naturally. For these reasons, they are also the predominant approach used for codes specialising in fast urban response scenarios. Consequently, they will be the focus here.

The principle difficulty of simulating turbulent flows comes from the wide range of time and length scales present which must be accounted for in some way. With varying degrees of accuracy and cost, there is a spectrum of approaches available for dealing with this problem in CFD. This is depicted in Figure 1.1. On one end of the spectrum, there is Direct Numerical Simulation (DNS), where all time and length scales of turbulence are resolved in the simulation. For city scale flows, DNS is simply far too computationally expensive since extremely fine grids and small time steps are required to resolve all turbulent scales. Additionally, the detailed information that one gets from a DNS simulation may be much more than what is required for this application (or any other engineering application). As a result, DNS is generally only used as a tool for fundamental research.

Now, if only the smallest of turbulent scales are accounted for through modelling instead of being resolved exactly, while the larger, more energetic scales are resolved, we get Large Eddy Simulation (LES). LES is much less costly than DNS, but still quite expensive, especially for 3D flows. This is particularly true for urban response scenarios where a solution is required in a time frame that is on the order of minutes, and access to large High Performance Computing

(HPC) resources may not be possible [21].

If an ensemble average of the equations are considered, then large scale turbulent eddies are also accounted for through modelling, and we get the Reynolds-Averaged Navier-Stokes (RANS) equations. It is common to also assume that the flow is statistically stationary. That is, the statistical properties of the flow such as the mean velocity do not vary with time. In this case solutions to the RANS equations provide a time averaged, steady-state solution of the flow field, and all the physics of large and small scale turbulence is accounted for with a turbulence model. For many problems, the assumption that the flow is statistically stationary may not be a good one, and hence accuracy may suffer. Despite this, given their cost, compared to DNS and LES, their accuracy is sufficient to obtain useful results for many engineering applications.

One can relax the assumption that the flow is statistically stationary to solve the Unsteady RANS (URANS) equations. In this case, unsteady mean flow structures are resolved. This is unlike LES where the eddies of the turbulence itself are resolved. As a result, URANS does not require the same level of spatial and temporal detail in the mesh as LES, but is still much more computationally expensive than a steady RANS simulations. In terms of accuracy, this places URANS between RANS and LES in Fig. 1.1, and has the advantage over steady RANS that it can capture unsteady flow features such as shedding [22].

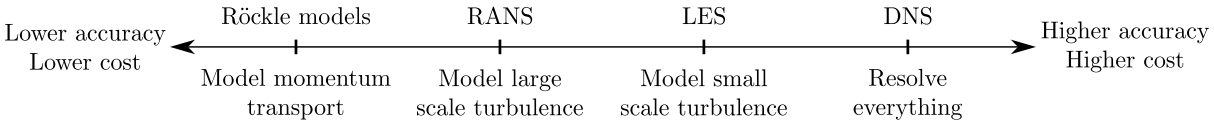


Figure 1.1: Spectrum of main methods that can be used for the calculation of the windfield, ranging from higher accuracy and high cost, to lower accuracy and lower cost.

For urban dispersion calculations where rapid turnaround time of solutions is required, even current steady RANS models fail to provide solutions in a short enough time frame. As a result, authors have taken to further modelling of the basic physics. Röckle [23] proposed a model where an initial windfield is first specified based on available meteorological data or another atmospheric model. This initial windfield does not take into account the presence of buildings, so it is modified in regions around buildings based on empirically derived parametrisations for each building. These parametrisations come from wind tunnel experiments for the flow over simple building-like objects such a surface mounted rectangular prism. This windfield is then projected onto a divergence free subspace to satisfy the incompressible continuity equation using a least squares variational approach [24]. With respect to the CFD models discussed above, this approach essentially accounts for the effect of momentum conservation through empirical, algebraic models, but still maintains mass conservation (Figure 1.1). Since Röckle's initial

introduction of this technique, numerous implementations have appeared, including the Quick Urban and Industrial Complex (QUIC) by Los Alamos National Lab (LANL) [25, 26], Micro SWIFT/SPRAY (MSS) by Aria Technologies Science Applications International Corporation (SAIC) [27], the 3-D Wind Field (3DWF) model by the U.S Army Research Laboratory (ARL) [28, 29], and the Israel Institute of Biological Research (IIBR) model [30] to name a few. Each of these models differ in their implementation from the original Röckle method with various modifications and improvements to the wind field parametrisations [31–35], but the general idea is the same.

Sing et al. [25] performed comparison studies of a modified Röckle scheme implemented in QUIC-URB with wind tunnel data for an array of cube shaped buildings. Good agreement is shown for simple isolated building cases however the model was unable to correctly predict the centres of circulation zones in the upwind cavity and the vortex between buildings. The more recent comparison by Hayati et al. [36] compares the QUIC-URB model, a RANS solver using the $k - \varepsilon$ turbulence model, and LES with wind tunnel experiments for a street canyon. The main shortcoming of the QUIC-URB model was it's failure to account for transitions in the flow regime, as well as for vertical flow in the lower half of the street canyons. Importantly, they report that QUIC-URB took approximately 5 s to run on a single core, while the RANS model they used took 5 h using four cores on the same machine. So, despite its shortcomings in accuracy, its utility becomes obvious when the time to compute the solution is considered.

These mass consistent models have a number of fundamental drawbacks however. Firstly, only the mean flow field is calculated and since the momentum equations are not solved, turbulence quantities such as the Reynolds stresses and the turbulence kinetic energy are not directly available. These quantities are needed for calculating the dispersion of a passive scalar through a turbulent velocity field due to the additional turbulent mixing that occurs. Most implementations of Röckle's scheme obtain turbulence parameters by using well-known relationships that relate them to the friction velocity and Monin-Obukhov length scale which come from dimensional analysis and experiments [37]. These relationships are true in the surface (constant stress) layer only, and do not exactly hold in the complex flow between buildings. The friction velocity is computed based on a local gradient turbulence closure that uses simple mixing length arguments [38, 39]. This approach is highly simplified but they are utilized out of necessity and justified on the grounds that the additional fidelity gained by using a more sophisticated approach is offset by the lack of accurate knowledge of other relevant flow parameters [39].

Additionally, in these methods, there is no natural way to deal with complex shaped buildings that do not correspond to the buildings that the parametrisations were created from, or multi-

building problems where recirculation zones interact. The parametrisations used come from empirical fits to experimental data, and cannot be immediately generalised. Brown et al. [33] describes the discrete logic that is used in QUIC-URB that introduces recirculation zones in an order based on relative building sizes and position, but the approach may vary between models [40]. Because of this, the models lack most of the physics required to generalise to a wider range of problems naturally and make reliable predictions on arbitrary geometries in urban environments.

Given these shortcomings, fast RANS-based models have been explored by authors with the aim of reintroducing the physics of the momentum equations into the model. As part of the QUIC software suite, a CFD based method called QUIC-CFD was developed to act as an intermediate model which produces reasonably fast solutions, without relying on empirical algorithms based on idealized building configurations [41, 42]. An artificial compressibility approach was used to solve the steady equations on a structured staggered grid. Buildings were accounted for by using a "staircase" approximation, where cells that fall within the building are simply marked as solid cells, and no attempt is made to accurately resolve the surface. Along with this, a simple mixing length eddy viscosity turbulence model was used with log law wall functions at solid boundaries. Validation studies of QUIC-CFD with wind tunnel data for both an isolated cube, an array of cubes [42], and with field experiments [21, 43] were carried out, and results were compared to both the Röckle based QUIC-URB model and an LES model. In comparison to QUIC-URB, most flow features were much better predicted. Qualitatively, it was noted that there were regions where the velocity vectors produced by QUIC-URB appeared "noisy" or unphysical, which was not the case with the CFD based methods. However, for all models, including LES, there were still regions that were very poorly predicted, which indicates that sometimes a more complete model may not lead to more accurate results. A large portion of these errors can be attributed to the many other uncertainties that exist in the simulation pipeline, and not just the fidelity of the model. Importantly, for the field experiment, QUIC-URB took on the order of 1 min compared to 30 min for QUIC-CFD using a "standard PC". The LES simulation took 30 h on an 8-node parallel cluster. Although QUIC-CFD is a step in the right direction of bridging the gap between complete CFD models and the diagnostic approach introduced by Röckle, there are a few areas where improvements can be made. The staircase approximation for buildings is quite crude, especially when coarser grids are used, and the applicability of the mixing length turbulence model used is questionable for this application. Additionally, the artificial compressibility approach may not be the most efficient option among currently available methods for achieving steady-state solutions in this case [44–46]. This is

cemented by the fact that the simulation time of QUIC-CFD and QUIC-URB differ by an order of magnitude, with the increase in accuracy not necessarily making up for this time discrepancy, indicating that there is still a need for development.

Because of this, there is ongoing interest in developing techniques which can more efficiently obtain windfield solutions using CFD or other more physics based models. Some of these approaches include leveraging Graphics Processing Units (GPUs) with existing standard algorithms [47–49], approximating geometries as porous media [50], using precomputed and stored solutions for different flow conditions to extrapolate to other conditions [51], and machine learning based data driven approaches [52–54]. It is worthy to note that for unsteady solutions there is Fast Fluid Dynamics (FFD) [55]. FFD uses a semi-Lagrangian approach which circumvents the time step restrictions associated with Eulerian time stepping of advection schemes, meaning large time steps and coarse grids can be used. Various authors have applied this technique to urban flows with good results for obtaining relatively accurate unsteady solutions with a fraction of the computational cost of other unsteady methods [56–59]. However it is unclear if using FFD to obtain steady solutions is competitive with steady methods for achieving fast solutions. Nonetheless, it appears that the literature has not seen much in the development of fundamental CFD methods for steady problems recently, especially for the present application. All the previous approaches mentioned depend on CFD at some level, so it is clear that the general development of fundamental CFD techniques is of interest and great value to the wider community.

1.2 Objectives

The aim of this work is to create a method which allows urban flows to be calculated to higher accuracy than a best-informed guess or any other modern method which can calculate the flow in a similar time. Such a method, despite not being able to perfectly replicate experimental measurements, will still be genuinely useful. In Section 1.1, it was highlighted that mass conservative Röckle based models lack the physics of the momentum conservation equations and so generalising them to arbitrary geometries is difficult, and there is no natural way to obtain turbulence quantities. On the other hand, current CFD based methods for this application are significantly more time consuming by comparison, by roughly an order of magnitude. Hence there is a gap in current techniques for this application: there is a need for fast running, but approximate CFD methods.

The CFD method that is developed throughout this thesis is optimised for the calculation of

steady-state turbulent flows. The reason for focusing on steady-state flows is as follows: if an unsteady solution is desired, it needs to be calculated at least up until the time where dispersion is required. This itself can already be very expensive if many time steps are to be computed. During this, the dispersion calculation will need to be performed, which adds an additional cost. At the end of all this, the data will generally be averaged in order to obtain a useful result for the user. So, naturally in this case it makes more sense to obtain a steady-state solution immediately. Of course, this cannot be expected to produce results to the same level of accuracy, but it comes with significant savings in computational cost.

Roughly speaking, the development of the solver can be broken down into three fundamental elements: the basic solver method, the treatment of complex geometries, and turbulence modelling. In each of these elements, careful, conscious decisions have been made to produce a solver which is optimised for rapid turnaround time of steady solutions with complex urban-like geometries. Furthermore, the techniques developed and the implementation of the solver is done with the intention of being highly efficient on high end consumer grade desktop hardware. One of the principle aims is to allow solutions to useful problems on commodity hardware. In many situations, access to HPC resources may not be possible, and using consumer desktop hardware is the only option. Furthermore, in any real usage scenario, the user will not have a perfect understanding of the wind inflow conditions. In this case, parametric tests are used to determine the sensitivity of the predictions on these unknown inputs, which means that dozens of jobs may be run for a single response scenario. A fast running code can do all these jobs in a smaller amount of time on a desktop computer. That being said, everything developed here can naturally be generalised to other applications and architectures. Above all, the greatest priority is to produce a method that is simple both conceptually and in implementation.

While the focus here is on methods for urban flows, many of the ideas that have been developed throughout this thesis are general and applicable to essentially any other fluid dynamics application – the governing equations are the same after all. Very little assumptions have been made that are specific to the application of urban flows, and when they are, it will be obvious how the method can be extended to other flows. In fact, many of the lines of reasoning used to create a performant algorithm are completely general, and come from properties of the Navier-Stokes equations and finite volume methods.

1.3 Outline and Approach

As mentioned in Section 1.2, the development of the solver is broken up into three elements: the basic solver method, the treatment of complex geometries, and turbulence modelling. All these elements are developed independently of each other, but progressively combined within the solver framework throughout this thesis. The basic solver method introduced forms the foundation of this work, and is what everything else is built upon. The finite volume method is adopted on collocated grids, which are chosen for their simplicity, particularly in the treatment of complex geometries and multigrid methods. The basic discretisation of the Navier-Stokes equations on a collocated finite volume grid are given in Chapter 2. In this chapter, the basic notation that will be used is also introduced, and some comments are made on the implementation of collocated grids.

In Chapter 3 the basic solver is introduced in detail where the method for solving the incompressible steady equations is developed. Here only the laminar equations are considered, with turbulence modelling being introduced later in the thesis. While the standard approach in incompressible CFD is to use segregated methods to solve the coupled momentum and continuity equations, a coupled approach is taken here. As will be shown, these methods are superior in terms of convergence rate and robustness when compared to segregated methods. Furthermore, the method is completely matrix free which results in a much simpler implementation. This solver is used within an Full Approximation Scheme (FAS) multigrid method. Results of numerical tests for validating and verifying the basic method on some simple canonical steady-state laminar test cases are given in this chapter. Performance against other standard steady-state solvers is also done in this section. Parallel techniques for the method and efficiency testing of a parallel implementation on shared memory CPU machines is given. The work in this chapter was the subject of a publication in Computers & Fluids [60] which covers the basic solver method and one currently under review [61] which extends the method to the FAS multigrid method.

Chapter 4 introduces the immersed boundary method that is used within the solver to allow for the flow to be calculated over complex geometries. The coupled nature of the method introduced in Chapter 3, as well the fact that collocated grids are used introduces a number of difficulties within the immersed boundary method that have not been addressed in previous literature. These issues are addressed here through the development of a mass conservative immersed boundary method that is simple to implement. Also, the immersed boundary method is coupled in to the FAS multigrid method. A range of tests are performed to verify and validate the method and demonstrate its efficiency when combined with the FAS scheme. The work in this chapter was the subject of a publication in the Journal of Computational Physics [62].

Finally, the turbulence modelling approach is given in Chapter 5. Along with Chapters 3 and 4, this chapter forms a complete solver which is capable of solving steady-state turbulent flows over complex urban-like geometries. In line with the goal of producing a fast, approximate method, a zero-equation turbulence model was used for its simplicity and efficiency. Its implementation is described in this chapter, and three urban flow cases which are part of the University of Hamburg CEDVAL experimental reference data set [63] are carried out. The accuracy and speed of the present method are compared with two standard implementations of the $k - \varepsilon$ model.

Conclusions are presented in Chapter 6. While the overall solver developed above is complete in that it can very efficiently solve for steady-state turbulent flows in complex built environments, there is still many areas of improvement and future research. This is also discussed in this chapter in detail.

CHAPTER 2

Discretisation of the Navier-Stokes Equations

2.1 Introduction

In this chapter, the approach for discretising the incompressible steady-state Navier-Stokes equations on a collocated finite volume grid is detailed. Since a coupled solver is used, the continuity equation is discretised in its primitive form, that is, a Poisson equation for pressure is not formed. The discretisation of each individual term will be described, and the stencil notation used throughout this thesis is introduced. Here, only the laminar equations are shown. In Chapter 5 when turbulence models are introduced, the discretisation of these models and the Reynolds stresses will be given there. The discretisation that will be described here are standard approaches in CFD, and a relatively brief overview will be given. For further details, readers are referred to the textbooks by J. H. Ferziger, M. Perić, and R. L. Street [44] and F. Moukalled, L. Mangani, and M. Darwish [64].

2.2 Finite Volume Method

The incompressible steady-state Navier-Stokes equations in vector notation are given by

$$\nabla \cdot (\boldsymbol{u} \otimes \boldsymbol{u}) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \boldsymbol{u} \tag{2.1}$$

$$\nabla \cdot \boldsymbol{u} = 0, \tag{2.2}$$

where $\boldsymbol{u} = (u, v, w)^T$ is the velocity vector, p is the pressure, ρ is the constant fluid mass density, and ν is the kinematic viscosity. The momentum equations can be written as a general conservation equation of a quantity ϕ

$$\nabla \cdot (\boldsymbol{u}\phi) = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \phi, \tag{2.3}$$

where ϕ is the velocity component corresponding to a given momentum equation. To discretise these equations using the finite volume method, we consider the integral form of these equations, given by

$$\int_{\partial\Omega} \boldsymbol{u} \cdot \hat{\boldsymbol{n}}\, \phi \, dS = -\frac{1}{\rho} \int_{\partial\Omega} p \, dS + \nu \int_{\partial\Omega} \nabla\phi \cdot \hat{\boldsymbol{n}} \, dS, \tag{2.4}$$

$$\int_{\partial\Omega} \boldsymbol{u} \cdot \hat{\boldsymbol{n}} \, dS, \tag{2.5}$$

where $\hat{\boldsymbol{n}}$ is the outward pointing unit normal vector, and Ω is a control volume, taken to be a finite volume cell. These equations are discretised on a rectilinear collocated finite volume grid. On a collocated grid, velocity and pressure variables are both stored at the cell centre location. This is opposed to a staggered grid, where the velocities are stored at the cell faces, meaning they represent face fluxes directly (Figure 2.1). While staggered grids are more natural mathematically, the implementation is much simpler in collocated grids, especially when using the techniques introduced in this thesis, as will be discussed. The drawback of a collocated grid is the pressure-velocity decoupling issue, which is addressed in Section 2.5.

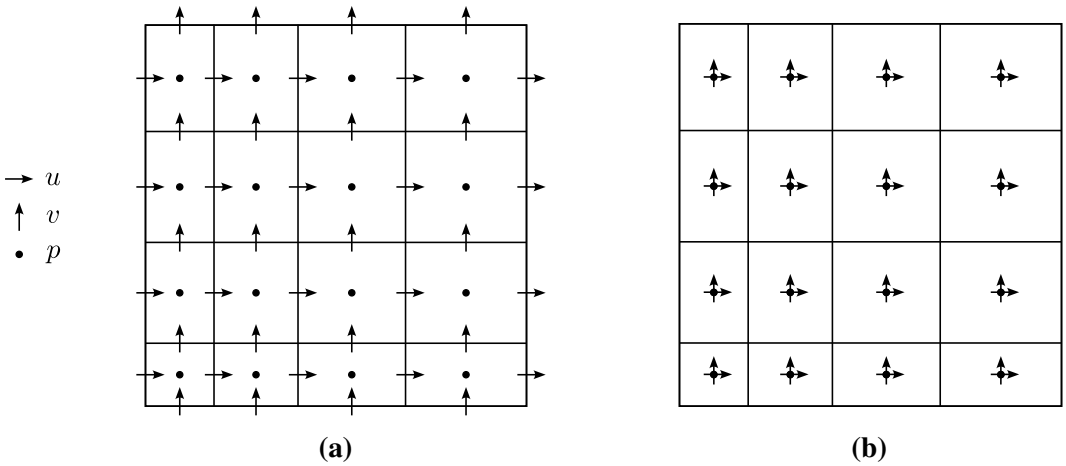


Figure 2.1: Staggered grid where velocities are stored at cell faces and pressure at cell centre (a), and collocated grid where both velocities and pressures are stored at cell centres (b).

The integrands of the integrals are approximated with the cell face centre values (which is an approximation to the average value over the cell face) of the quantities of interest. Rewriting the integrals as discrete summations over the cell faces gives

$$\sum_f (\boldsymbol{u} \cdot \hat{\boldsymbol{n}})_f \, \phi_f A_f = -\sum_f p_f (\hat{\boldsymbol{e}} \cdot \hat{\boldsymbol{n}})_f A_f + \nu \sum_f (\nabla\phi \cdot \hat{\boldsymbol{n}})_f A_f, \tag{2.6}$$

$$\sum_f (\boldsymbol{u} \cdot \hat{\boldsymbol{n}})_f A_f = 0, \tag{2.7}$$

where the subscript $f = \{n, e, s, w, t, b\}$ indicates a cell face (Figure 2.2), A_f is the cell face area, and V_P is the volume of the cell P. The set of coupled algebraic equations to be solved is obtained

by approximating the cell face values in terms of the cell centre values using interpolation. The approximations used to do this are described in Section 2.4.

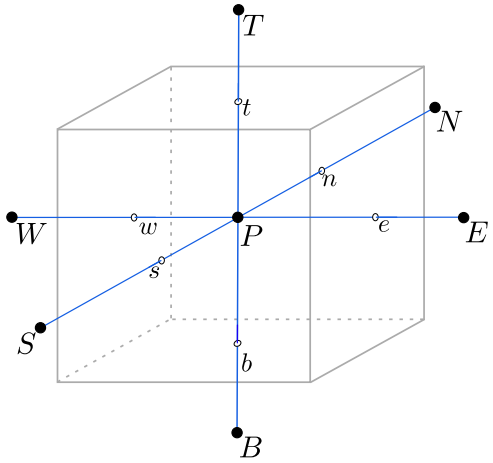


Figure 2.2: Finite volume cell at location P with its neighbouring node labels. Lowercase letters are used to label cell faces.

2.3 Linearisation

The advective term in Eq. 2.6 is nonlinear – it contains two velocity components multiplied together. The equations need to be linearised so that they can be solved iteratively. There are a number of approaches one can take, including Newton linearisation. Here, the nonlinear advective term is treated using Picard iteration due to its simplicity. In Picard iteration, iterations are performed, called "outer iterations", where the advecting velocity is taken as fixed from the previous outer iteration, resulting in a linear system. That is:

$$\sum_f (\boldsymbol{u} \cdot \hat{\boldsymbol{n}})_f \, \phi_f A_f \approx \sum_f (\bar{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}})_f \, \phi_f A_f \tag{2.8}$$

where $\bar{\boldsymbol{u}}$ is the velocity from the previous outer iteration. This linear system can then be solved, and if an iterative method is used for this, the iterations performed within each outer iteration are called "inner iterations". It is usually not necessary to solve the linearised system accurately at each outer iteration, and generally only a small number of iterations is needed. Outer iterations are stopped when the desired residual tolerance is met for the full nonlinear system. After each outer iteration, the newly obtained velocities are used to calculated the advective velocities for the next outer iteration.

2.4 Discretisation of Terms

Below, the approximations used to write each of the individual terms in the linearised form of Eq. 2.6 and 2.7 in terms of the cell centres are given. It should be noted that the methods developed throughout this thesis are not specific to the particular approximations chosen here. Furthermore, to simplify notation, the discretisation will be shown for the e face of the cell (Figure 2.2). The application to all other faces is identical.

2.4.1 Diffusion

The diffusive flux on each face is approximated using a central difference scheme for the derivative, which is second order accurate:

$$\left.\frac{\partial \phi}{\partial x}\right|_e = \frac{u_E - u_P}{\Delta x_e}. \tag{2.9}$$

where $\Delta x_e = x_E - x_P$ is the distance between the cell centres of the cells that straddle the e face (E and P). Considering the 1D case, the diffusive term would then be approximated by

$$\nu \sum_f (\nabla \phi \cdot \hat{\boldsymbol{n}})_f A_f = \nu \left(\left.\frac{\partial \phi}{\partial x}\right|_e A_e - \left.\frac{\partial \phi}{\partial x}\right|_w A_w \right) \tag{2.10}$$

$$\approx \nu \left(\frac{A_e}{\Delta x_e}\phi_E - \left(\frac{A_e}{\Delta x_e} + \frac{A_w}{\Delta x_w} \right) \phi_P + \frac{A_w}{\Delta x_w}\phi_W \right). \tag{2.11}$$

Adopting the following stencil notation,

$$\nu \sum_f (\nabla \phi \cdot \hat{\boldsymbol{n}})_f A_f = a_E \phi_E + a_P \phi_P + a_W \phi_W \tag{2.12}$$

we have

$$a_E = \nu \frac{A_e}{\Delta x_e}, \qquad a_W = \nu \frac{A_w}{\Delta x_w}, \tag{2.13}$$

$$a_P = -\left(a_E + a_W\right). \tag{2.14}$$

The property that the diagonal coefficient a_P is equal to the sum of the negative of the off-diagonal coefficients is a requirement to guarantee physically consistent results with diffusion [64], and is true in general for higher dimensions. This property will also be important below regarding the advective terms.

2.4.2 Advection

In the advective term $\sum_f (\bar{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}})_f \, \phi_f A_f$, the advecting velocity $(\bar{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}})_f$ is approximated using weighted linear interpolation. Again for the e face:

$$(\bar{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}})_e = \bar{u}_e \approx (1 - \lambda_e)\bar{u}_P + \lambda_e \bar{u}_E \tag{2.15}$$

where the overbar indicates that the quantity is taken from the previous outer iteration due to the Picard linearisation (Section 2.3), and λ_e is the interpolation factor associated with the e face, given by

$$\lambda_e = \frac{x_e - x_P}{x_E - x_P}. \tag{2.16}$$

Note that for x-normal faces, the advecting velocity is the u component, while for y and z-normal faces, the v and w velocities are the advective components respectively.

The advected velocity ϕ_f is treated using a deferred correction to achieve second order accuracy. Implicitly, first order upwind is used:

$$\phi_e \approx \begin{cases} \phi_P & \text{if } \bar{u}_e \geq 0 \\ \phi_E & \text{if } \bar{u}_e < 0 \end{cases} \tag{2.17}$$

The upwind scheme for a single cell can be expressed more simply using the min and max functions. In 1D, this can be written as

$$\begin{aligned} \sum_f (\bar{\boldsymbol{u}} \cdot \hat{\boldsymbol{n}})_f \, \phi_f A_f \approx & A_e \text{min}\left\{\bar{u}_e, 0\right\} \phi_E - A_w \text{max}\left\{\bar{u}_w, 0\right\} \phi_W \\ & + \left(A_e \text{max}\left\{\bar{u}_e, 0\right\} - A_w \text{min}\left\{\bar{u}_w, 0\right\}\right) \phi_P. \end{aligned} \tag{2.18}$$

Using the stencil notation that was introduced in Eq. 2.12, the stencil coefficients can be written as

$$a_E = A_e \text{min}\left\{\bar{u}_e, 0\right\}, \qquad a_W = -A_w \text{max}\left\{\bar{u}_w, 0\right\}, \tag{2.19}$$

$$a_P = -\left(a_E + a_W\right) + \left(A_e \bar{u}_e - A_w \bar{u}_w\right). \tag{2.20}$$

Unlike diffusion, the diagonal advective coefficient cannot be written directly as the negative sum of the off-diagonal terms. Note however, the extra term added, $(A_e \bar{u}_e - A_w \bar{u}_w)$ is the local divergence, which is zero if continuity is satisfied. As a result, this term can be left out, since once a converged solution is reached, it will become zero. Note however, since a_E and a_W are both negative, if this local divergence is positive for a given cell, it acts to increase the diagonal dominance of the system during convergence when continuity has not yet been achieved. Hence

it is advantageous in terms of the stability and robustness of an iterative solver to include this term for a particular cell only when it is positive. When this term is negative, it decreases the diagonal dominance, and should be left out. Strictly speaking, due to the addition of Momentum Weighted Interpolation (MWI) in the continuity equation, this local divergence term will not be exactly zero. However, this divergence error is very small, and decreases as Δh^2. More details regarding MWI and continuity are given in Section 2.5.

Using higher order advection schemes – apart from the central difference scheme, which is not considered here due to its less favourable stability properties – lead to a wider computational molecule. This can be expensive computationally, but also leads to higher code complexity. However, it is generally quite simple to calculate these higher order schemes explicitly. Combining this with an implicit low order approximation (upwind in this case) gives the deferred correction approach. The advected velocity approximated with deferred correction is given by

$$\phi_e \approx \phi_e^{\text{upwind}} + \beta \left(\bar{\phi}_e^{\text{HOS}} - \bar{\phi}_e^{\text{upwind}} \right) \tag{2.21}$$

where HOS stands for "Higher Order Scheme", the overbar indicates the solution from the previous outer iteration, and β is a blending factor which set to 1 for simplicity. The overbar terms are included as source terms in the discretised equations, while the first term makes up part of the computational stencil, as described above. In this work, the QUICK (Quadratic Upwind Interpolation for Convective Kinematics) scheme is used to achieve second order accuracy [44]. The face advected velocity for the QUICK scheme when $\bar{u}_e \geq 0$ is given by [64]

$$\phi_e = \phi_P + \frac{(x_e - x_P)(x_e - x_W)}{(x_E - x_P)(x_E - x_W)}(\phi_E - \phi_P) + \frac{(x_e - x_P)(x_E - x_e)}{(x_P - x_W)(x_E - x_W)}(\phi_P - \phi_W). \tag{2.22}$$

For the case where $\bar{u}_e < 0$, the stencil can simply be mirrored about the cell face. Notice that Eq. 2.22 contains values of ϕ from cells that are not immediately adjacent to the cell face e. This can result in a wide stencil for the cell P, where values of ϕ from locations to cells away from P are required. That is, cells at locations NN, EE, SS, WW, TT, and BB..

Finally, at the boundary faces no deferred correction is used. Instead the stencil for the advected velocity is changed to the central scheme:

$$\phi_e \approx (1 - \lambda_e)\phi_P + \lambda_e \phi_E. \tag{2.23}$$

This means that when using ghost cells that are obtained by linear extrapolation from the fluid domain, the advected velocity will be set exactly to the boundary condition value. This also simplifies boundary condition treatment since higher order schemes with wide molecules such as QUICK do need to be evaluated at the boundary. Boundary conditions and the use of ghost cells is detailed further in Section 2.6.

2.4.3 Pressure Gradient and Velocity Divergence

In the finite volume method, the pressure gradient is treated as a surface force on the cell face area. This requires evaluation of the pressure on the cell faces. Similarly, the divergence from the continuity equation results in the evaluation of the velocity fluxes on the cell faces. For both these terms, weighted linear interpolation is used to approximate quantities on cell faces:

$$p_e \approx (1 - \lambda_e)p_P + \lambda_e p_E, \tag{2.24}$$

$$u_e \approx (1 - \lambda_e)u_P + \lambda_e u_E. \tag{2.25}$$

2.5 Momentum Weighted Interpolation

Since a collocated grid is used, MWI is required to prevent pressure-velocity decoupling in the solution which can lead to non-physical pressure oscillations and solution instability [65]. The MWI results in a correction to the face velocity in the discrete continuity equation given by

$$\boldsymbol{u}_f = \overline{\boldsymbol{u}}_f - d_f \left(\nabla p_f - \overline{\nabla p}_f \right), \tag{2.26}$$

where the overbar indicates a weighted linear interpolation from the neighbouring cells, $d_P = V_P/a_P$, and d_f is calculated by a weighted linear interpolation from the neighbouring cells. The coefficient a_p in d_P is the coefficient containing both the advective and diffusive terms from the momentum equations. The second term on the RHS of Eq. 2.26 will be called the "MWI correction". At the domain boundaries, the MWI correction is set to zero. The pressure gradients in the x-direction on the e cell face are

$$\left.\frac{\partial p}{\partial x}\right|_e = \frac{p_E - p_P}{x_E - x_P}, \tag{2.27}$$

$$\overline{\left.\frac{\partial p}{\partial x}\right|_e} = (1 - \lambda_e)\frac{p_E - p_W}{x_E - x_W} + \lambda_e \frac{p_{EE} - p_P}{x_{EE} - x_P}, \tag{2.28}$$

where EE is the next cell over from the E cell. The use of MWI introduces pressure terms into the discrete continuity equation, and results in a wider stencil due to the sparse averaged pressure gradient on the right hand side of Eq. 2.26. It is possible to treat the sparse gradient term $\overline{\nabla p}_f$ explicitly (from the previous outer iteration), and the compact gradient term ∇p_f implicitly to avoid the need for a wide stencil. In the authors experience, this results in reduced robustness of the solver, but may be worth it in cases where a wide stencil is impractical to use. In this work, all terms are treated implicitly. Complete explicit treatment of the MWI correction however leads to very poor stability of the solver.

On a collocated grid, when central and linear approximations are used, the formation of a Poisson equation results in sparse pressure gradients, which is the source of the odd-even pressure decoupling. Although a Poisson equation is not explicitly formed here or as part of the coupled solver process, it is still implicitly created during the solution process, as will be shown in Chapter. 3. The role of the MWI is to reduce this sparse pressure gradient into a compact gradient Laplacian approximation [66]. This can be seen in Eq. 2.26 where the sparse gradient term $\overline{\nabla p}_f$ is subtracted away, while the compact gradient ∇p_f is added back in. This process results in the addition of a term that is proportional to the fourth-order derivative of the pressure into the continuity equation, meaning that strictly speaking, continuity is no longer satisfied. This additional term is proportional to Δh^2 however, so the second order accuracy of the scheme is still preserved [65].

Although the MWI is formulated as a correction to the face velocity flux, it is only used for the face velocity fluxes in the discrete continuity equation, Eq. 2.7. All face velocities in the momentum equation, Eq. 2.6 are calculated using a simple weighted linear interpolation. This simplifies the face flux calculation in the Picard linearisation since it eliminates the need to recalculate the MWI correction in Eq. 2.26 each time the face flux is updated. This means that the face fluxes used in the momentum equations do not satisfy the continuity equation, since they are not defined according to Eq. 2.26, but a simple weighted linear average. It is for this reason that the local divergence term in Eq. 2.20 will not go to zero. However as mentioned previously, the error in continuity introduced is proportional to Δh^2, and so this error is second order accurate.

2.6 Boundary Condition Treatment

Boundary conditions are dealt with through the assignment of ghost cells which are placed around all the domain boundaries. Since the mesh is nonuniform, these ghost cells are given the same dimensions as the interior domain cell adjacent to them for simplicity. This is illustrated in Figure 2.3. The interpolation factors and the the cell centre distance at these boundary faces is calculated using the ghost cells in exactly the same way they are calculated for interior cells. This means that the calculation of the above approximations at boundary cells is identical to interior cells. Ghost cells are set such that when a linear profile is assumed between cell centres, calculation of cell face values or gradients using these ghost cells results in the correct boundary conditions. Four common boundary conditions and their application with ghost cells will be discussed here. Their application to all variables is the same.

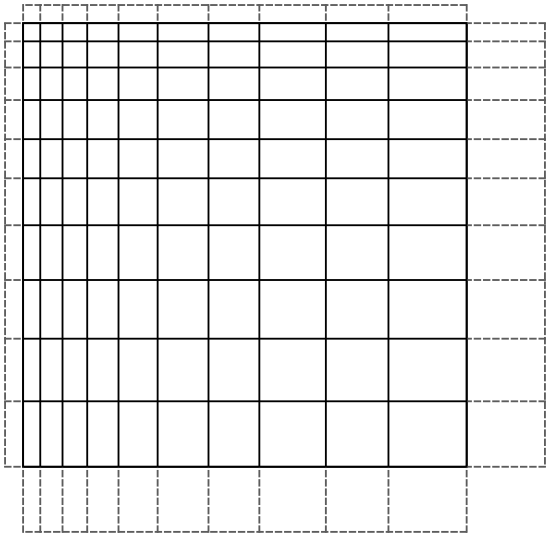


Figure 2.3: 2D rectilinear mesh with ghost cells (shown with grey dashed lines) on domain boundaries. The ghost cells have the same dimensions as the interior cell adjacent to them.

2.6.1 Dirichlet Boundary Conditions

For Dirichlet boundary conditions, the value of the field variable at the boundary cell face is specified. Assuming the boundary face is the e face, linear interpolation onto the boundary cell face, is given by

$$\phi_e = (1 - \lambda_e)\phi_P + \lambda_e \hat{\phi}_E, \tag{2.29}$$

where $\hat{\phi}_E$ is a ghost cell, and ϕ_e is known from the boundary condition. Note that for the boundary faces with ghost cells, $\lambda_e = \frac{1}{2}$. The ghost cell value is then obtained by rearranging this equation to give

$$\hat{\phi}_E = \frac{1}{\lambda_e}\phi_e - \frac{1 - \lambda_e}{\lambda_e}\phi_P. \tag{2.30}$$

2.6.2 Neumann Boundary Conditions

For Neumann boundary conditions, the value of the gradient in the normal direction to the domain boundary at the cell face is specified. Assuming the boundary face is the e face, the gradient at the cell face is given by

$$\left.\frac{\partial \phi}{\partial x}\right|_e = \frac{\hat{\phi}_E - \phi_P}{\hat{x}_E - x_P}. \tag{2.31}$$

As above, $\hat{\phi}_E$ is the ghost cell value, $\hat{x}_E$ is the coordinate of the ghost cell, and $\left.\frac{\partial \phi}{\partial x}\right|_e$ is known from the boundary condition. For faces with normals in the y and z-directions, gradients would be evaluated in those directions respectively. For boundary faces with ghost cells, $\hat{x}_E - x_P = \Delta x_P$, since ghost cells have the same dimensions as the interior cell. Eq. 2.31 can then be rearranged

for the ghost cell value

$$\hat{\phi}_E = \phi_P + (\hat{x}_E - x_P) \left.\frac{\partial \phi}{\partial x}\right|_e . \tag{2.32}$$

2.6.3 Extrapolated Boundary Conditions

In extrapolated boundary conditions, the cell face value is obtained by linear extrapolation from the interior of the domain. The ghost cell is then set using Eq. 2.30. For a boundary face on the e face, linear extrapolation, using the two internal cells is given by

$$\phi_e = \frac{2\Delta x_P + \Delta x_W}{\Delta x_P + \Delta x_W}\phi_P - \frac{\Delta x_P}{\Delta x_P + \Delta x_W}\phi_W. \tag{2.33}$$

If the boundary cell face was on the w face, then the extrapolation formula can simply be mirrored. That is, the coefficient for ϕ_W would become the coefficient for ϕ_E, and Δx_W would be replaced by Δx_E.

2.6.4 Periodic Boundary Conditions

For a periodic boundary condition, the variables on cell faces on opposite ends of the domain are equal to eachother, that is $\phi_e = \phi_w$ if ϕ_e is on the eastmost face of the domain and ϕ_w is on the westmost face in the domain (Figure 2.4). Ghost cells on each side of the domain need to be set in order to enforce this.

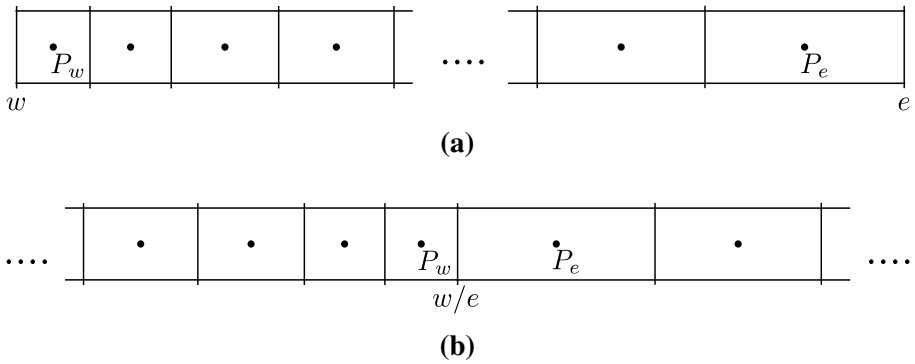


Figure 2.4: Periodic boundary conditions on a solution domain (a) imply that the e and w faces are shared (b), or that the field value on that face are identical.

Periodic boundary conditions are equivalent to "wrapping" the domain around so that the e and w faces are shared, as shown in Figure 2.4b. In periodic boundary conditions $\phi_e = \phi_w$, which we will denote by $\phi_{w/e}$. With reference to Figure 2.4b, $\phi_{w/e}$ can be obtained via linear interpolation as

$$\phi_{w/e} = (1 - \lambda_{w/e})\phi_{P_w} + \lambda_{w/e}\phi_{P_e} \tag{2.34}$$

where the interpolation factor is given by

$$\lambda_{w/e} = \frac{\Delta x_{P_w}}{\Delta x_{P_w} + \Delta x_{P_e}}. \tag{2.35}$$

Once the face value at the boundaries $\phi_{w/e}$ is obtained, the corresponding ghost cell value on each side can be obtained in the same way it is done for Dirichlet boundary conditions using Eq. 2.30.

2.6.5 General Remarks

Ghost cells should be updated as frequently as possible. After the field values are updated in an iterative solver, the ghost cells will cease to give the correct boundary condition when the stencil is evaluated until they are updated again. Using "old" ghost cell values can affect the convergence rate of the iterative solver, as well as the stability. Generally, this means ghost cells should be updated after every inner iteration.

An alternative approach to using ghost cells is to set the boundary conditions directly within the discretisation of the finite volume equations. For example, in the case of the diffusion in Eq. 2.10, this would involve setting the value of the derivative within the discretisation if it is a Neumann condition, or approximating the derivative using the specified face value. This results in different stencil coefficients for the cells at the domain boundary. Treating boundary conditions in this way has the benefit that ghost cells do not need to be updated at each iteration, and it is guaranteed that the boundary condition is always satisfied exactly. The drawback is that this significantly increases the complexity of the code, since each boundary condition for each term has to be set individually instead of simply treating boundary cells in the same way internal cells are treated. Furthermore, the computational cost of updating ghost cells is quite small compared other solver operations, so the cost savings are typically not worth it.

One final, subtle point that should be brought to attention is the d_f term in Eq. 2.26, which contains the a_P coefficient. If boundary conditions are treated directly in the stencil (i.e. not using ghost cells), then the coefficient a_P will differ at the boundaries to the coefficient that would be used if ghost cells were used. In other words, the a_P coefficient in Eq. 2.26 when ghost cells are used will not contain boundary condition information. To the authors knowledge, this difference has not been discussed in literature, however its effect on the solution is likely very small, especially considering that the coefficient is obtained by a linear averaging of the two neighbouring cells, and the MWI correction is set to zero at the boundary.

2.7 Finite Volume Stencil on A Rectilinear Mesh

The complete discretisation of the Navier-Stokes equations using the above methods can be represented using stencil notation for each equation at a cell location P. For the discrete momentum equations, this is given by

$$a_P^{uu}u_P \; + \sum_{n(P)} a_F^{uu}u_F + \\ a_W^{up}p_W + a_P^{up}p_P + a_E^{up}p_E = B_P^u, \tag{2.36}$$

$$a_P^{vv}v_P \; + \sum_{n(P)} a_F^{vv}v_F + \\ a_S^{vp}p_S + a_P^{vp}p_P + a_N^{vp}p_N = B_P^v, \tag{2.37}$$

$$a_P^{ww}w_P + \sum_{n(P)} a_F^{ww}w_F + \\ a_B^{wp}p_B + a_P^{wp}p_P + a_T^{wp}p_T = B_P^w. \tag{2.38}$$

Where u, v, w are the x, y, and z components of velocity, p is the pressure, $n(P)$ indicates the neighbouring nodes of the cell at location P, and B_P is a source term. The first superscript in the coefficients indicates the equation the coefficient belongs to, and the second one denotes the variable the coefficient multiplies with. The subscripts indicate the location in space relative to the cell P of a quantity (Figure 2.2). The velocity terms come from the discretisation of the advection and diffusion, while the pressure terms come from the pressure gradient. The discretised continuity equation is given by

$$\begin{aligned} &a_W^{cu}u_W + a_P^{cu}u_P \; + a_E^{cu}u_E \; + \\ &a_S^{cv}v_S \; \; + a_P^{cv}v_P \; + a_N^{cv}v_N \; + \\ &a_B^{cw}w_B + a_P^{cw}w_P + a_T^{cw}w_T + \\ &a_P^{cp}p_P + \sum_{n(P)} a_F^{cp}p_F = 0. \end{aligned} \tag{2.39}$$

Here the velocity terms come from the divergence of the velocity, while the pressure terms appear due to the presence of the MWI. Throughout this work, the convention adopted is that terms on the left hand side of the equals sign are treated implicitly in the solver, while terms on the right hand side are treated explicitly. For a coupled solver, all variables (velocity and pressure) in the momentum and continuity equations are treated implicitly. This will come into play in the following chapter where the coupled smoother is introduced.

When a rectilinear mesh is used with collocated variables, a number of simplifications for the storage of the finite volume stencil coefficients can be made that result in a smaller memory footprint and more efficient memory access of the finite volume coefficients.

Firstly, notice that on a rectilinear mesh, the area of cell faces do not vary in the normal direction of the cell face. For example, in 3D the area of e cell face only varies in the y and z directions, and it is given by $A_e(i,j,k) = \Delta y_j \Delta z_k$, where i, j, and k denote the cell index in the x, y and z directions respectively. This means that all the coefficients above have a dependence of Δx_i, Δy_j, and Δz_k. By dividing all coefficients by the cell volume $V_P = \Delta x_P \Delta y_P \Delta z_P$, they will only have a dependence on the cell dimension that is normal to the cell face for which they are approximating. For example, we shall consider the pressure gradient terms for the x-momentum equation, which is given by

$$\sum_f p_f (\hat{\boldsymbol{e}} \cdot \hat{\boldsymbol{n}})_f A_f = p_e A_e - p_w A_w \tag{2.40}$$

$$= \Delta y_P \Delta z_P \left[\lambda_e p_E + (1 - \lambda_e - \lambda_w) p_P - (1 - \lambda_w) p_W \right]. \tag{2.41}$$

The interpolation coefficients λ_e and λ_w vary in the face normal direction x only. Dividing through by the cell volume $\Delta x_P \Delta y_P \Delta z_P$, the coefficient becomes

$$\sum_f p_f (\hat{\boldsymbol{e}} \cdot \hat{\boldsymbol{n}})_f A_f = \frac{1}{\Delta x_P} \left[\lambda_e p_E + (1 - \lambda_e - \lambda_w) p_P - (1 - \lambda_w) p_W \right]. \tag{2.42}$$

The dependence of the pressure coefficients in the y and z directions has disappeared. The implication is that a unique value of the coefficient does not need to be stored at every cell location in the mesh, in 3D. Instead, they can be stored in a 1D array corresponding to each i location, where they are constant. This can be done for all other coefficients in the discrete equations. Unfortunately, since the advective terms depend on the cell face velocity, which varies throughout the domain, the advective coefficients must be stored in 3D. The same is true for the MWI pressure coefficients in the continuity equation, which depend on the diagonal velocity momentum coefficient. Nonetheless, this means that the pressure coefficients in the momentum equations, and the velocity coefficients in the continuity equation can all be stored in 1D. In 3D, this is a total of 18 3D arrays that can be stored in 1D. If each coefficient is an 8 B floating point number, this is a saving of approximately 144 MB per million cells.

The next simplification is with regards to the advection and diffusion coefficients. Notice, that when Picard linearisation is used the advection and diffusion coefficients for all three momentum equations are identical i.e. $a_F^{uu} = a_F^{vv} = a_F^{ww}$ for all cell locations F. This means that all the velocity coefficients in the discrete momentum equations (Eqs. 2.36 - 2.38) can all refer to the same memory in a finite volume code. For the 7 coefficients required in each of the two momentum equations, stored as 8 B floating point numbers, this is a saving of 112 MB per million cells.

The final simplification comes from the similarity between the pressure gradient terms and the velocity divergence terms in the continuity equations. As mentioned in Section 2.4.3, both

these terms are of the form where the field variable (velocity flux or pressure) is evaluated at the cell face and multiplied by the cell face area. The only difference being that the pressure gradient contains a factor of $\frac{1}{\rho}$. For an incompressible flow, we can assume $\rho = 1$. Then the coefficients corresponding to each direction become identical. More specifically

$$a_W^{up} = a_W^{cu}, \qquad a_P^{up} = a_P^{cu}, \qquad a_E^{up} = a_E^{cu}, \tag{2.43}$$

$$a_S^{vp} = a_S^{cv}, \qquad a_P^{vp} = a_P^{cv}, \qquad a_N^{vp} = a_N^{cv}, \tag{2.44}$$

$$a_B^{wp} = a_B^{cw}, \qquad a_P^{wp} = a_P^{cw}, \qquad a_T^{wp} = a_T^{cw}. \tag{2.45}$$

Since these coefficients were already stored in 1D, the memory savings from this optimisation are not significant. However, since the same memory will be used more often, this results in a improvement in data locality which is more cache efficient. This is particularly important for a coupled solver where both pressure and velocity stencils will be evaluated together throughout the solution process. The same principle applies for the shared advection coefficients.

Given these simplifications, it is possible to simplify the notation in Eqs. 2.36 - 2.39 to take into account coefficients that are identical. This will not be done here to maintain generality of the methods that will be discussed. But it should be realised that when the method is implemented into code, these optimisation should be made. It should also be made clear that these optimisation are only really possible for matrix-free methods. If a sparse matrix solver is used, coefficients (or at least references to the coefficients) must be stored for each non-zero element of the linearised system. Finally, analogous optimisations are not possible on a staggered rectilinear grid, since the grid used for each of the variables is geometrically different, and the interpolation between variables is no longer identical.

2.8 Convergence Residuals

With any iterative solution procedure, a measure of some kind is required to judge how well the discrete equations have been satisfied. This is done based on the residuals of the discrete equations. The residuals of the discrete momentum equations (Eqs. 2.36 – 2.38) at cell P are

defined as

$$r_P^u = B_P^u - a_P^{uu} u_P - \sum_{n(P)} a_F^{uu} u_F - a_W^{up} p_W - a_P^{up} p_P - a_E^{up} p_E, \tag{2.46}$$

$$r_P^v = B_P^v - a_P^{vv} v_P - \sum_{n(P)} a_F^{vv} v_F - a_S^{vp} p_S - a_P^{vp} p_P - a_N^{vp} p_N, \tag{2.47}$$

$$r_P^w = B_P^w - a_P^{ww} w_P - \sum_{n(P)} a_F^{ww} w_F - a_B^{wp} p_B - a_P^{wp} p_P - a_T^{wp} p_T, \tag{2.48}$$

and the residual for the discrete continuity equation (Eq. 2.39) is

$$\begin{aligned} r_P^c = & a_W^{cu} u_W + a_P^{cu} u_P + a_E^{cu} u_E + \\ & a_S^{cv} v_S + a_P^{cv} v_P + a_N^{cv} v_N + \\ & a_B^{cw} w_B + a_P^{cw} w_P + a_T^{cw} w_T + \\ & a_P^{cp} p_P + \sum_{n(P)} a_F^{cp} p_F. \end{aligned} \tag{2.49}$$

Note that the continuity residual (Eq. 2.49) is the local divergence in each cell. If the discrete equations are satisfied exactly, then these residuals are zero everywhere on the domain. The convergence of the momentum equations is measured by taking the L_1-norm of the residuals over the entire domain, and normalising by the L_1-norm of the diagonal coefficient:

$$R^u = \frac{\sum_{P \in \mathcal{D}} |r_P^u|}{\sum_{P \in \mathcal{D}} |a_P^{uu} u_P|} \tag{2.50}$$

$$R^v = \frac{\sum_{P \in \mathcal{D}} |r_P^v|}{\sum_{P \in \mathcal{D}} |a_P^{vv} v_P|} \tag{2.51}$$

$$R^w = \frac{\sum_{P \in \mathcal{D}} |r_P^w|}{\sum_{P \in \mathcal{D}} |a_P^{ww} w_P|}. \tag{2.52}$$

This normalisation is chosen as the velocity in the cell P is representative of flow rate through that cell. It is multiplied with the diagonal coefficient to make it dimensionally consistent with the residual. The L_1-norm of this quantity is then essentially the average flow rate in the domain. Normalising the residual in this way makes the residual more physically meaningful. That is, it represents the error of the velocity relative to the magnitude of velocity in the domain.

For the continuity equation, the residual is normalised by the number of cells in the domain:

$$R^c = \frac{\sum_{P \in \mathcal{D}} |r_P^c|}{N_x N_y N_z}, \tag{2.53}$$

which means this is a measure of the average local divergence in the domain. This value on its own can be difficult to use as a judgement of convergence however, since what constitutes a small divergence is problem dependent. Unfortunately, there is no representative value with which this could be normalised in the same way the momentum equations were. Instead, the continuity residual is scaled by the continuity residual of the initial condition:

$$R^c_{\text{scaled}} = \frac{R^c}{R^c_{\text{initial}}}. \tag{2.54}$$

It is possible however for the initial residual to be zero – which can happen for example when the initial conditions satisfy continuity and the problem contains no inlets or outlets, such as a lid driven cavity. In these cases, to avoid the division by zero, the residual is not scaled at all, and the raw value is used. The approach described here for normalising and scaling the residuals is used in many commercial CFD packages, including ANSYS Fluent [67].

The convergence residuals described above were used for all tests done throughout this thesis, and from this point onwards are simply referred to as "residuals" for brevity. Determining the appropriate residuals to use is generally problem dependent, and is dictated by the requirement that the residuals should be chosen to be small enough such that errors in the solution of the discrete equations are smaller than the discretisation errors themselves. This is done by testing a range of residuals until the particular quantity of interest in the solution is no longer dependent on the residual. These quantities of interest could be the order of accuracy obtained from grid refinement tests, the drag over an object, or the error from an experiment or reference solution to name a few.

2.9 Summary

A brief but complete description of the underlying discretisation used throughout this thesis for the incompressible Navier-Stokes equations has been given. This forms the basic foundation upon which the coupled smoother, and immersed boundary method developed in this thesis build upon. As will become evident, many of the issues that this work addresses in the following chapters are tied to the underlying discretisation used. Additional details regarding various simplifications that can be made to the discretisation and storage of coefficients that arise by virtue of the fact that a rectilinear mesh on a collocated grid was used were given. These simplifications are one of the principle factors that contribute to the smaller memory footprint and computational efficiency of the complete method developed here.

CHAPTER 3

The Coupled Solver

3.1 Introduction

Segregated solution algorithms for the Navier-Stokes equations such as the well known Semi-Implicit Method for Pressure-Linked Equations (SIMPLE) [68] are the dominant approaches for the solution of steady incompressible flows. However, the more implicit pressure-velocity coupling in coupled methods leads to more efficient and robust solutions [69, 70]. The primary drawback being larger computer memory requirements since the coefficients for the discretised momentum and continuity equations must all be stored at the same time. Nevertheless, the declining cost and increasing availability of computer hardware means that these methods are more feasible [71].

These coupled methods are commonly implemented in a “black-box" manner: by assembling the linearised set of momentum and continuity equations into a matrix system and making use of an existing general sparse matrix solver. Darwish et al. implemented such an approach within the OpenFOAM framework on both structured [72] and unstructured [73] grids using the finite volume method on a collocated grid. A similar method can be found in Deng et al. [74]. In these formulations, the continuity equation in its primitive form is used in the discretised system as opposed to a Poisson equation for pressure. The use of the continuity equation leads to a smaller system size and simpler implementation, however, it results in poorer conditioning of the linear system due to the pressure not appearing explicitly along the main diagonal in the rows corresponding to the continuity equation of the linear system. This is true for both staggered and collocated grid arrangements. For this reason Ammara and Masson [75] use the discretised Poisson equation for pressure in their coupled procedure. The relative performance of the two approaches is not clear in the literature, however the above authors have demonstrated in any case that these coupled methods solve incompressible steady flows with significantly lower

computational cost than segregated methods.

Vanka [76] introduced a coupled block implicit approach on staggered grids which simultaneously updates a pressure node and its neighbouring velocity nodes. This smoothing technique is usually called Symmetrically Coupled Gauss-Seidel (SCGS) or the "Vanka smoother". The Vanka smoother has demonstrated its effectiveness over segregated techniques such as the SIMPLE algorithm on staggered grids both as a solver within a multigrid procedure, but also as a preconditioner, particularly for steady problems [76–80]. The Vanka smoother also has the added benefit of a simpler matrix-free implementation compared to the aforementioned coupled techniques. One drawback is that the velocity on each face is evaluated twice since each velocity node is shared by two continuity cells, resulting in a larger number of floating point operations per sweep of the solution domain. Vanka found that using an asymmetrically coupled version of the Vanka smoother to avoid this results in poor convergence and robustness [76], however Claus and Bolten [81] were able to get good convergence properties for an asymmetric Vanka smoother for the Stokes equations by updating the domain using various sweep directions and orientations. The Vanka smoother has also only been applied to a staggered grid.

Paisley and Bhatti [82] considered a line-wise coupled version of the Vanka smoother where each row of cells were solved directly using a tridiagonal matrix solver. While for the 2D lid driven cavity, this approach led to an improvement in performance, the convergence rate was found to deteriorate for more complex 2D flows over obstacles. Compared to the standard Vanka smoother, this approach is more efficient in terms of the number of evaluations and leads to a reduction in computing time per iteration. They also found that there was a significant effect of the sweeping direction on convergence rate, with sweeping across the dominant flow direction giving the best convergence rates. Paisley [77] extended this idea to 3D with a planewise method, where each plane was solved using the line sweeping method. For the 3D lid driven cavity, the spanwise direction was optimal for sweeping, however only marginal improvement was found against the SIMPLE scheme in this optimal direction in terms of computational time. Their manner of data storage however had an impact on the solution algorithm, meaning the most efficient sweeping direction in terms of convergence rate was not necessarily the most efficient in terms of solver time. Consequently, it is difficult to come to a conclusion on the relative efficiency of such an approach. Bruneau and Khadra [83] parallelise a multigrid solver based on a line smoother similar to that of Paisley and Bhatti, and show that it is possible to reach high scalability with this method. They do not perform direct comparisons with other smoothers however. It is important to note that all these methods have only been successfully applied to staggered grids.

The idea of orienting a solution algorithm with the dominant flow direction to accelerate convergence has been explored extensively in the past. In particular for subsonic incompressible flows with no recirculation, it is possible to solve the flow in a single spatial march by making use of a "parabolising" approximation for the flow [84, 85]. For regions with small amounts of recirculation and little upstream influence, multi-sweep methods based on the parabolised or reduced Navier-Stokes equations have also found some success [86, 87]. Unfortunately, these methods fail to be efficient when large amounts of upstream recirculation are present due to the use of single sided approximations for the pressure gradients and the divergence in the continuity equation. Nonetheless, the concept of aligning the solution procedure with the physics is a useful one that is explored here.

Clearly for incompressible steady flows, coupled approaches are superior in terms of computational cost, especially considering the greater availability and reduced cost of computer memory in recent times. Furthermore, the work of Paisley and Bhatti demonstrate that there is potential in line/plane sweeping methods, particularly in the choice of sweeping direction with respect to problem geometry. All these techniques however have been based on the Vanka smoother for staggered grids, and to the authors knowledge, there is no analogue for collocated grids.

In this work, collocated grids are favoured for their simpler implementation, especially when used for multigrid and immersed boundary methods. Grid transfer and reconstruction operators are easier to construct, and the same operators can be applied to all variables directly. This fact will become more apparent throughout this chapter and Chapter 4 where the immersed boundary method is introduced. Multigrid methods are a very effective framework for accelerating solution convergence and have become widely adopted in CFD, including in many commercial CFD codes. For incompressible problems, these methods have been applied in various different ways. Most commonly, it is used within a linear solver to solve the Poisson equation for pressure in segregated methods, or to solve the complete linearised equations in coupled methods [72, 73]. On the other hand, Full Approximation Scheme (FAS) multigrid methods have been applied to the complete non-linear equations resulting in better convergence rates [88]. Typically this has been done using segregated algorithms such as the SIMPLE scheme as the smoother at each level [89, 90]. It is reasonable to assume then that using a fully coupled smoother at each level will give superior convergence rates. This is in fact what is done in this work.

Collocated grids are already widely adopted in many commercial CFD packages, so the development of more efficient methods is of interest to the general CFD community. Of course, the principle drawback of collocated grids is the pressure velocity decoupling which can lead to

nonphysical pressure oscillations and instability. This is addressed however by using MWI [65], which was introduced in Chapter 2. The addition of MWI does break mass conservation up to a second order error [44], and consequently results in the appearance of artificial kinetic energy in the solution. However, authors are continuing to develop techniques in kinetic energy preserving schemes [66, 91, 92], which continues to make collocated grids more favourable, especially in the context of turbulent flows.

In this chapter a matrix-free coupled block implicit smoother for solving the Navier-Stokes equations on a collocated finite volume grid in a very similar manner to the Vanka smoother is introduced. The method is simple to implement and solves the coupled momentum and continuity equations within a molecule in a block Gauss-Seidel like manner. The basic method is described in detail in Section 3.2. Due to the nature of the method, some special considerations are needed at domain boundaries, which is addressed in Section 3.3. Then in Section 3.4, a number of strategies for parallelising the basic method are given and discussed. The focus is on shared memory parallel machines. The FAS multigrid implementation of the method is then given in Section 3.5. The accuracy and performance of the method is tested using two laminar steady test cases: the 3D lid driven cavity, and the 3D backwards facing step. The effect of sweeping direction for updating the variables on convergence rate and robustness is also explored. The accuracy and performance of the method is also compared with the coupled solver in ANSYS Fluent [93] and the SIMPLE scheme in OpenFOAM [94]. The shared memory parallel performance of the present method is also tested on two different computer systems: a high end consumer desktop, and an High Performance Computing (HPC) node. The results of all these tests are given in Section 3.7.

3.2 The Coupled Smoother Technique

For a uniform grid when second order central approximations are used, the central coefficients a_P for the pressure terms in the momentum equations and the velocity terms in the continuity equation are zero. If the grid is non-uniform, these coefficients will be small except for grids with very high growth rates, so their coupling is very small. This fact is the main contributor to the pressure checkerboarding problem on collocated grids. This also means that if the coupled equations were assembled into a block matrix structure, with each block corresponding to a given cell, the velocities at cell location P in the continuity equation and the pressures at cell location P in the momentum equations do not appear implicitly in each local block. Furthermore, only the central pressure coefficient from the MWI would appear along the diagonal in the rows

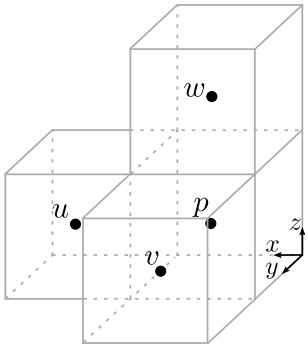


Figure 3.1: Offset triad showing momentum equations and cells coupled together implicitly. Shown is the case where all momentum equations are offset in the positive coordinate direction, however the offset direction will depend on the sweeping direction.

corresponding to the continuity equation, and the system will not be diagonally dominant. This diagonal term is not significant enough to allow robust solution of the coupled system through a block Gauss-Seidel method, and in general leads to poor conditioning of the system. As a result, such an approach is unstable.

The approach proposed here addresses this issue by offsetting the momentum equations that are coupled with the continuity equation in each local block (Figure 3.1). This idea was inspired by the coupled space marching method used for the parabolised Navier-Stokes equations by Zhu [95]. Instead of forming the block coupled system with all variables at location P, only the pressure is treated implicitly at location P, with the other variables offset in the sweeping direction. As an example, we consider the case where the x-momentum equation is taken at the E cell, the y-momentum equation is taken at the N cell, and the z-momentum is taken at the T cell. The local block matrix system becomes

$$\begin{pmatrix} (a_P^{uu})_E & 0 & 0 & (a_W^{up})_E \\ 0 & (a_P^{vv})_N & 0 & (a_S^{vp})_N \\ 0 & 0 & (a_P^{ww})_T & (a_B^{wp})_T \\ a_E^{cu} & a_N^{cv} & a_T^{cw} & a_P^{cp} \end{pmatrix} \begin{pmatrix} u_E \\ v_N \\ w_T \\ p_P \end{pmatrix} = \begin{pmatrix} B_E^u - \sum\limits_{n(E)} a_F^{uu} u_F - (a_P^{up})_E p_E - (a_E^{up})_E p_{EE} \\ B_N^v - \sum\limits_{n(N)} a_F^{vv} u_F - (a_P^{vp})_N p_N - (a_N^{vp})_N p_{NN} \\ B_T^w - \sum\limits_{n(T)} a_F^{ww} u_F - (a_P^{wp})_T p_T - (a_T^{wp})_T p_{TT} \\ B_P^c \end{pmatrix}. \tag{3.1}$$

Variables on the right hand side are taken to be frozen at their most recent values, while values

on the left hand side are solved for. With this arrangement, the dominant pressure terms in the momentum equations, and velocity terms in the continuity equation appear implicitly. We will then define the explicit (frozen) part of the momentum equations as

$$\begin{aligned} G^u_E &= B^u_E - \sum_{n(E)} a^{uu}_F u_F - (a^{up}_P)_E p_E - (a^{up}_E)_E p_{EE}, \\ G^v_N &= B^v_N - \sum_{n(N)} a^{vv}_F u_F - (a^{vp}_P)_N p_N - (a^{vp}_N)_N p_{NN}, \\ G^w_T &= B^w_T - \sum_{n(T)} a^{ww}_F u_F - (a^{wp}_P)_T p_T - (a^{wp}_T)_T p_{TT}. \end{aligned} \tag{3.2}$$

Through Gaussian elimination, the system in Eq. 3.1 can be written into the upper triangular form

$$\begin{pmatrix} (a^{uu}_P)_E & 0 & 0 & (a^{up}_W)_E \\ 0 & (a^{vv}_P)_N & 0 & (a^{vp}_S)_N \\ 0 & 0 & (a^{ww}_P)_T & (a^{wp}_T)_T \\ 0 & 0 & 0 & K \end{pmatrix} \begin{pmatrix} u_E \\ v_N \\ w_T \\ p_P \end{pmatrix} = \begin{pmatrix} G^u_E \\ G^v_N \\ G^w_T \\ B^c_P - \frac{a^{cu}_E}{(a^{uu}_P)_E} G^u_E - \frac{a^{cv}_N}{(a^{vv}_P)_N} G^v_N - \frac{a^{cw}_T}{(a^{ww}_P)_T} G^w_T \end{pmatrix}, \tag{3.3}$$

Where

$$K = a^{cp}_P - a^{cu}_E \frac{(a^{up}_W)_E}{(a^{uu}_P)_E} - a^{cv}_N \frac{(a^{vp}_S)_N}{(a^{vv}_P)_N} - a^{cw}_T \frac{(a^{wp}_B)_T}{(a^{ww}_P)_T}. \tag{3.4}$$

The first three rows in Eq. 3.3 are identical to those in Eq. 3.1. Equation 3.3 is upper triangular and can easily be solved for the coupled variables p_P, u_E, v_N, w_T. First, the terms G^u_E, G^v_N, G^w_T are evaluated and stored. Then they are used to solve for the pressure from the bottom row of Eq. 3.3. This pressure is then used, along with the already calculated quantities of G^u_E, G^v_N, G^w_T to update the velocity within the block. After this, the next block in the solution sweep can be solved. Note that in a practical implementation, the matrices above do not need to be explicitly formed. The update equations that result from Eq. 3.3 can simply be applied directly.

The Gaussian elimination process used to obtain Eq. 3.3 from Eq. 3.1 is akin to the derivation of the Poisson equation for pressure in SIMPLE-like methods where the expression for the linearised diagonal momentum equations is substituted into the continuity equation. That is, the

Schur complement of the linearised Navier-Stokes equations is approximated with its diagonal elements. This results in a global decoupling of the pressure and momentum equations at each iteration. However in the case of the present method, the substitution is done at a local level for each block, and only for one term in the continuity equation in each direction. This has the advantage of having the diagonal dominance of a Poisson equation for pressure, while retaining the simple discretisation of the continuity equation. The resulting pressure terms due to the MWI in the continuity equation further increase the diagonal dominance of the system. In terms of the linear system comprised of the discrete continuity and momentum equations, this local Gaussian elimination process can essentially be thought of as a preconditioner.

It is common in CFD schemes to apply under-relaxation to the equations to help stabilise the solution process. Typically, this is done implicitly by modifying the diagonal finite volume coefficient and adding a source term which contains the solution from the previous iteration. However, since the diagonal coefficient of the momentum equations is used to form the row from which the pressure is obtained (Eq. 3.3), under-relaxation should not be applied in this manner for the same reasons that the under-relaxed diagonal coefficient should no be used in the formation of the Pressure equation for SIMPLE type schemes. Namely, it results in poor convergence and robustness of the pressure solver. Instead, under-relaxation should be performed in an explicit manner. This is discussed in more detail in Section 3.5.6.

Naturally, one might consider a symmetric coupling of the momentum equations where in the case of Figure 3.1, the momentum equations at locations P, S, W, and B are also coupled in. In principle, this introduces strong pressure velocity coupling which will improve convergence and robustness of the scheme. Practically speaking however, it has a number of drawbacks. Firstly, symmetric coupling significantly increases the size of the stencil for each molecule, which makes parallelisation of the method more difficult – which is the subject of Section 3.4. Secondly, symmetric coupling means that the same momentum equation will be updated multiple times throughout the solution sweep since it can be shared by multiple continuity equations. This involves the re-evaluation of the terms in Eq. 3.2 for each cell multiple times per iteration, which can become costly. Finally, symmetric coupling will increase the size of the local block matrix (Eq. 3.1) from 4×4 to 10×10 if all terms in the continuity equation are included. This can be quite complex to implement.

The description of the method here was restricted to structured rectangular grids. It should be clear that the extension to curvilinear orthogonal and non-orthogonal structured grids is relatively straightforward, but outside the scope of this work. The procedure is essentially the same as above, with the exception of a larger molecule for each cell that will contribute

to the right hand side of Eq. 3.3. In principle, it is also possible to extend the method to unstructured grids by coupling the continuity equation of a cell with the momentum equations of the surrounding cells in the same way, however the choice of which cells to couple is no longer obvious. Additionally, the direction and order in which to update each molecule in the unstructured mesh not immediately clear. Some further work is needed if this technique is to be applied to unstructured meshes.

3.3 Boundary Condition Treatment and Sweeping

Due to the offsetting, and the fact that the momentum equations coupled with a continuity equation at a given cell come from cell neighbours, a single block coupled system cannot be used to obtain all variables at the boundaries. There are a number of possible ways one can deal with this, but here, simplicity in implementation is prioritised. Firstly, we allow the momentum equations to exit the domain when solving a molecule, as shown in Figure 3.2 for the case of the v-momentum equation. A momentum equation that has exited the domain no longer contributes to the solution, and the boundary condition through the ghost cell is used instead. This then means that the pressure p and the u velocity can be solved for at the boundary cells without having to form a special case for the molecule.

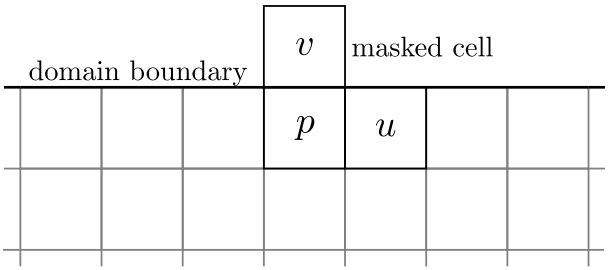


Figure 3.2: An offset molecule at the boundary with the v-momentum equation "hanging" out of the domain boundary, allowing for the solution of the u-momentum and continuity at the boundary cells. The same concept can be extended to 3D.

The correct boundary condition for the v velocity in this case is achieved by a simple masking. The mask is defined as

$$\text{mask}(P) = \begin{cases} 1 & \text{if cell } P \text{ is inside the fluid region} \\ 0 & \text{if cell } P \text{ is outside the fluid region} \end{cases} \tag{3.5}$$

Equation 3.4 must then be modified so that the momentum equation that is outside the domain is removed from the coupled system:

$$K = a_P^{cp} - a_E^{cu}\frac{(a_W^{up})_E}{(a_P^{uu})_E}\text{mask}(E) - a_N^{cv}\frac{(a_S^{vp})_N}{(a_P^{vv})_N}\text{mask}(N) - a_T^{cw}\frac{(a_B^{wp})_T}{(a_P^{ww})_T}\text{mask}(S). \tag{3.6}$$

The last row of Eq. 3.3 is changed so that the continuity equation uses the ghost cell value at the boundary instead of the momentum equation:

$$\begin{aligned} B_P^c &- \frac{a_E^{cu}}{(a_P^{uu})_E}\left[G_E^u \text{mask}(E) + (1-\text{mask}(E))u_E\right] \\ &- \frac{a_N^{cv}}{(a_P^{vv})_N}\left[G_N^v \text{mask}(N) + (1-\text{mask}(E))v_N\right] \\ &- \frac{a_T^{cw}}{(a_P^{ww})_T}\left[G_T^w \text{mask}(T) + (1-\text{mask}(E))w_T\right] \end{aligned} \tag{3.7}$$

where u_E, v_N, w_T will only be used when they are ghost cells. Finally, the update of the momentum equation that is hanging out the domain should be masked out as to not overwrite the ghost cell value. For example, for the v velocity, this would become

$$v_N \leftarrow (1-\text{mask}(N))\, v_N + \text{mask}(N) v_N^{\text{new}} \tag{3.8}$$

where v_N^{new} is the velocity obtained from inverting the system in Eq. 3.3. To allow the momentum equation to exit the domain safely, dummy cells must also be added to all the coefficient arrays so that out of bounds memory accesses do not occur. The value of these dummy cells can be set to zero as they will not contribute to the solution, however the central coefficient a_P at these dummy cells should be set to a non-zero value (such as 1) to avoid division by zero. The use of masking to achieve this means that the same code can be used for the boundaries and internal fluid cells – no special treatment is required.

The above procedure does not allow the continuity equation to exit the domain. With the offsetting, this still means that there are variables that still cannot be solved, in particular, the velocities in the corners of the domain that lie on the boundaries that oppose the offset direction. For example, in Figure 3.3a, the velocities in the bottom left corner cell cannot be solved for. This is overcome by performing a reverse sweep, where the offsetting in the molecule is reversed in each direction, as shown in Figure 3.3. This reversing of the molecule will also help solution convergence and robustness as it introduces more symmetry into the procedure.

In 2D, there are two ways in which the sweeping can be oriented, either vertically in the y-direction, as shown in Figure 3.3, or horizontally in the x direction. In 3D, there is six possible ways the sweeping can be oriented: two directions in each of the x, y, and z normal planes. A depiction of this in 3D is shown in Figure 3.4. For problems with a dominant flow direction, or some kind of asymmetry, the choice of these respective sweeping directions may have an effect on convergence rate and solver robustness. This is investigated in Section 3.7.

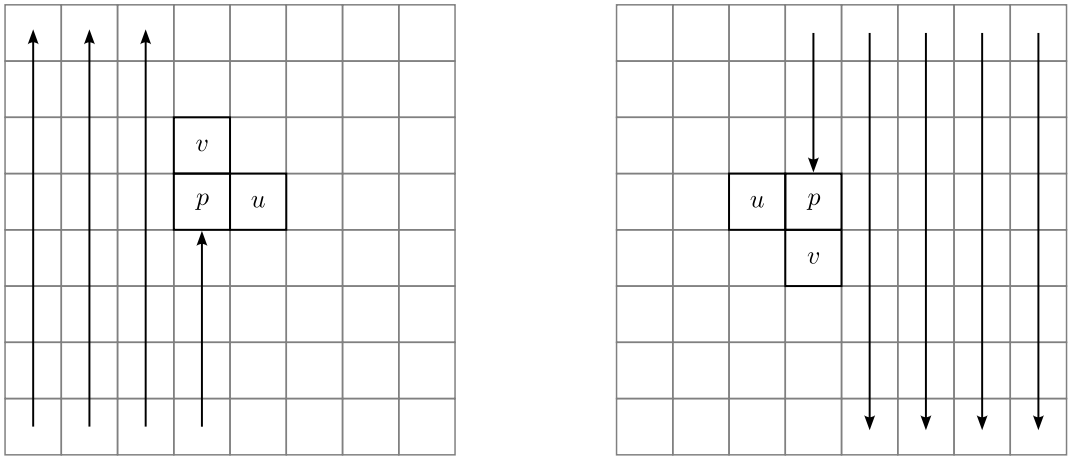


Figure 3.3: The forward (a) and backward (b) update sweeps used to ensure that all boundary cells are solved.

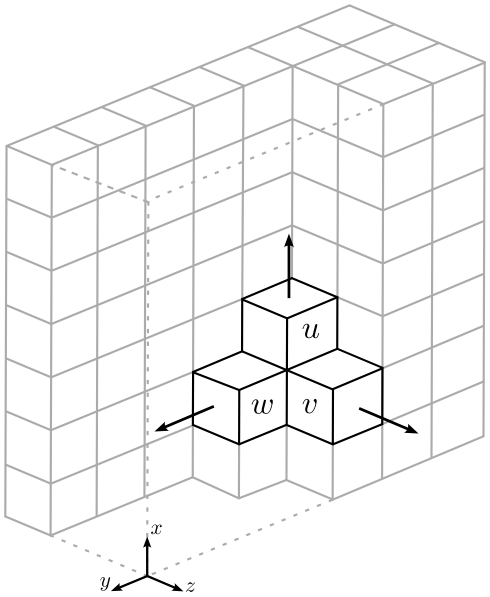


Figure 3.4: An instance where a line is updated by sweeping a triad in the x-direction. These lines are swept in the y-direction to update the plane, and the plane is swept in the z direction to update the domain. This is then done in reverse with the molecule flipped in all directions to enforce the boundary conditions.

3.4 Parallelism

As with any implicit stencil based method, the update of the variables in a single offset molecule require the field values of cells immediately adjacent to it. Therefore, updating the field values in those adjacent cells in parallel would lead to a race condition. To parallelise the method using a domain decomposition approach, colour ordering is required. The stencil dependency (Eqs. 2.36 – 2.39) of a single offset molecule on its neighbouring cells in 2D is shown in Figure 3.5. From this, it can be seen that the next nearest molecule that can be solved in parallel,

without referencing any variables that are in the current molecule, must be at least two cells away in any direction. This means that a three colour partitioning of the domain is required to allow parallel execution. The ordering of the colours must be chosen such that any two cells of the same colour have at least two different cell colours between them to ensure they can be solved in parallel safely.

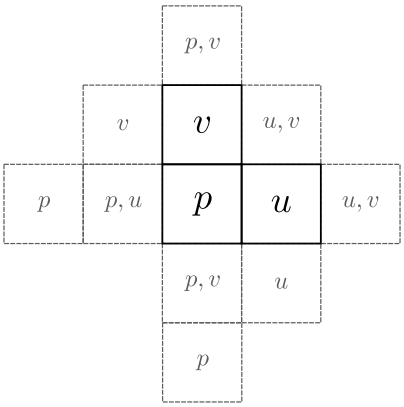


Figure 3.5: A single offset molecule (black solid lines) being solved with all with the variables required but not updated (grey dashed lines). The stencil dependence in 3D is the same.

The standard approach for multi-colour ordering for parallel methods is a 3D checkerboard, shown in Figure 3.6a. Here we use white, orange, and blue. Since the molecule is offset, and covers multiple cells, it is easiest to interpret the colouring as the cell corresponding to the continuity cell in the molecule. The offset momentum molecules will automatically satisfy the colouring. First, all the white cells are solved in parallel, followed by the orange cells, and finally the blue cells. Each colour is done using the sweeping described in Section 3.3.

Using multi-colour ordering for parallelism is applicable to essentially any parallel programming paradigm. However in this work, only shared memory parallelism on CPUs is considered (using OpenMP). This comes with an important practical consideration: memory bandwidth and cache contention. From this perspective, a pointwise 3D checkerboard (Figure 3.6a) may not be the most efficient choice. For example, once a cell of a particular colour is solved by a processor, the field variables (among other quantities) in the surrounding cells required to solve that cell will be in cache – this can be either cache private to the processor or shared, however in both cases, the following reasoning holds. The next molecule to be solved by that processor is two cells away, meaning that much of what is available in cache is unlikely to be used, resulting in a cache miss. Stencil based methods (such as the present method) are typically limited by memory bandwidth, so frequent cache misses and the resulting references to main memory can significantly degrade performance.

For this reason, we also consider partitioning each colour as lines (Figure 3.6b) and planes

(Figure 3.6c). The lines and planes are chosen such that they are contiguous in memory, and any given processor can only have complete lines and planes. That is, a processor will never be assigned only part of a line or plane. This has the advantage that each processor will have a more contiguous memory access pattern, with the plane colouring having a higher proportion of contiguous memory access than the line colouring. Of course this approach means that each line or plane is solved in serial. The increased proportion of computation being in serial means that parallel efficiency may be lower when this parallel strategy is used. The net effect on performance of using lines or planes is difficult to predict and will be dependent on the problem size, and the hardware used. It should be clarified that the three colour ordering is used to parallelise each forward and reverse sweep (see Section 3.3) separately, as opposed to performing each forward and reverse sweep within each colour. That is, all colours are first solved with a forward sweep, then they are solved with a reverse sweep.

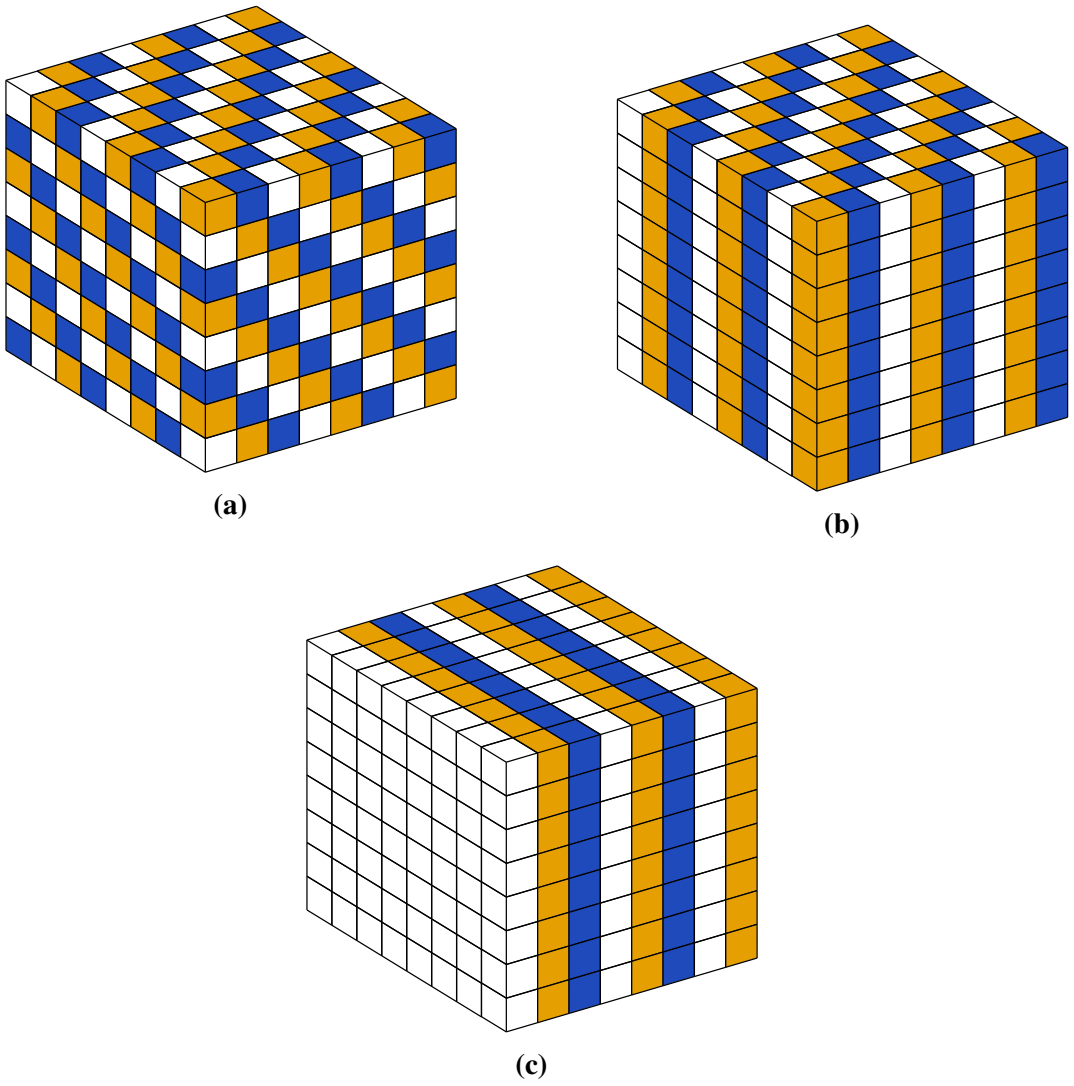


Figure 3.6: Three colour ordering required to parallelise the smoother. Parallel regions can be points (a), contiguous lines (b), or contiguous planes (c).

It is worth discussing some of the differences between the present method and the Vanka smoother for staggered grids with regards to parallelisation. The standard Vanka smoother on staggered grids, which symmetrically couples all variables requires five colours to parallelise using a domain decomposition approach [96]. This can be quite complex to implement, so a number of authors have taken the approach of a 1D plane parallelisation (similar to what is done in Figure 3.6c) with good results [83, 97]. Claus and Bolten [81] introduced an asymmetrically coupled Vanka smoother for the Stokes equations which only couples a single momentum equation with each continuity cell. This has some similarities to the present method, but on a staggered grid. For the case of the asymmetrically coupled Vanka smoother, only two colours is required for parallelisation. They show that with some algorithmic modifications (such as reorienting the asymmetric molecule) convergence rates almost as good as the symmetric Vanka smoother can be achieved. However they did not implement the method in parallel, and only used it to solve the Stokes equations, not the Navier-Stokes equations, so the comparison is not complete.

The various colouring patterns in Figure 3.6 all result in a different update ordering of the cells in the solution domain. It is reasonable to expect that this would affect the convergence rate of each approach. For example, in the plane parallel method (Figure 3.6c), each plane is updated by contiguously sweeping through the points in the domain (as is shown in Figure 3.3). This means each subsequent point in the plane that is being updated is always using the most recently available field variables in the particular colour that is being solved, which also happens to be the plane. For the line parallel method (Figure 3.6b) only the most recently available field variables in a particular line will be used. Finally, for the point parallel case (Figure 3.6a), no recently updated variables from the colour that is being solved will be used in subsequent molecules. Thus one would expect that the plane parallel method would give the best convergence rate, followed by the line parallel method, and finally the point parallel method.

Finally, apart from the solution update sweeping. Other parts of the code are also parallelised. In particular, the updating of coefficients at each outer iteration, since this is accounts for the majority of the runtime after the solution update sweeping. Fortunately, these calculations, when done cell by cell in a loop, have no dependencies between iterations, and can be trivially parallelised. In the present implementation, essentially all loop calculations in the solver have been parallelised using OpenMP. It is not within the scope of this work however to deeply analyse the parallel performance of these sections, particularly because compared to the solution update sweeping, they account for a relatively small amount of the run time.

Since the code developed here is intended to run on commodity desktop hardware – which

tend to be shared-memory CPU systems – careful consideration was given to how the problem was parallelised. As discussed above, many factors influence the choice of parallel strategy, which depends on both the problem and the hardware that will run the code. Tests on different domain sizes and computer systems were conducted, with results presented in Section 3.7, later in this chapter.

3.5 FAS Multigrid

Multigrid methods are a powerful acceleration technique for solving elliptic problems with iterative solvers. With relaxation-type iterative methods, large wavenumber errors in the solution are damped very rapidly, however error components associated with small wavenumbers are difficult to damp. This becomes a significant bottleneck when solving larger problems. Multigrid methods accelerate convergence of these small wavenumbers by moving the solution to coarser grids, where small wavenumbers on a fine grid are mapped to large wavenumbers on a coarse grid. Not only does this result in faster solver time, but linear scaling with problem size for an ideal implementation [98].

When applying multigrid methods to nonlinear equations, there are two approaches one can take. The first is to linearise the equations (using Picard iteration or Newton's method for example) and use multigrid Correction Scheme (CS) to solve the linear equations. The other approach is to apply the multigrid concept to the complete nonlinear equations and treat the nonlinearity of the problem on all grid levels – the FAS scheme [99]. The first approach has the benefit of being relatively simple to implement, especially if the nonlinear operator is complex. Furthermore, it means that a general multigrid algorithm for any set of linear equations can be used on a range of different nonlinear problems. However, since it does not treat the nonlinearity directly, its convergence rate is slower. In the context of the Navier-Stokes equations, this is important for highly advective (high Reynolds number) flows where advection dominates. Furthermore, textbook linear scaling cannot be achieved since multigrid is only being used on the linearised equations. For these reasons, the FAS scheme is adopted here, and its implementation with the above solver technique is described below.

3.5.1 The FAS Algorithm

We will represent the discretised Navier-Stokes equations as

$$A(\boldsymbol{x}) = \boldsymbol{f}, \tag{3.9}$$

where the operator A represents the discretised momentum and continuity equations including boundary conditions, $\boldsymbol{x}$ is a vector containing the velocities and pressure at the cell centres, and $\boldsymbol{f}$ are source terms. The details of how the discrete operator $A(\boldsymbol{x})$ is represented are not important, so long as it can be calculated. However in this work, and as is the case with all finite volume methods, $A(\boldsymbol{x})$ is calculated through stencil operations (Eqs.2.36 – 2.39). The residual is defined as

$$\boldsymbol{r} = \boldsymbol{f} - A(\boldsymbol{x}). \tag{3.10}$$

The notation $\boldsymbol{x}^{l+1} = I_l^{l+1}\boldsymbol{x}^l$ represents the relation between a quantity $\boldsymbol{x}$ on the fine grid level l and the coarse grid level $l+1$ using the operator I_l^{l+1} (restriction). The operator I_{l+1}^l would then represent the transfer from the coarse grid to the fine grid (prolongation). These operators are described in detail in Section 3.5.5. For a two level multigrid solver, the FAS scheme is given in Algorithm 1 [99].

Algorithm 1 Two level FAS multigrid

1:	Pre-smoothing:	$A^l(\boldsymbol{x}^l) = \boldsymbol{f}^l$
2:	Restriction:	$\boldsymbol{r}^{l+1} = I_l^{l+1}(\boldsymbol{f}^l - A^l(\boldsymbol{x}^l))$
3:		$\boldsymbol{x}^{l+1} = I_l^{l+1}\boldsymbol{x}^l$
4:	Solve:	$A^{l+1}(\boldsymbol{y}^{l+1}) = A^{l+1}(\boldsymbol{x}^{l+1}) + \boldsymbol{r}^{l+1}$
5:	Compute error:	$\boldsymbol{e}^{l+1} = \boldsymbol{y}^{l+1} - \boldsymbol{x}^{l+1}$
6:	Correct:	$\boldsymbol{x}^l \leftarrow \boldsymbol{x}^l + I_{l+1}^l\boldsymbol{e}^{l+1}$
7:	Post-smoothing:	$A^l(\boldsymbol{x}^l) = \boldsymbol{f}^l$

Note that the nonlinear equations to be solved on the coarse grid (Line 4 in Algorithm 1) are almost identical to the original problem on the fine grid given by Eq. 3.9, except for the source term on the right hand side. Fortunately, this source term is simple to calculate since it uses operators that are already available from a standard single grid solver.

3.5.2 Multigrid Cycles

To solve the problem on a hierarchy of more than two grid levels, the algorithm is recursively repeated. Direct recursive application of Algorithm 1 leads to V-cycle multigrid (Figure 3.7b). However, more complex cycle types are considered in this work, namely the F-cycle (Figure 3.7c) and the W-cycle (Figure 3.7d). These more complex cycles can be created through the composition of V-cycles and since they spend more time on coarser grid levels, result in a faster convergence rate at a lower computational cost. On every grid level except the coarsest grid level – where the solution is solved to a low residual tolerance – pre-smoothing iterations are performed prior to restriction, and post-smoothing iterations are performed after the solution is corrected, as shown

in Algorithm 1. For pre and post-smoothing, a fixed number of outer iterations is performed to avoid overhead of calculating a stopping criterion. Through trial and error, it was also found that generally only a small number of pre and post-smoothing iterations is needed – typically around 3. On the coarsest grid level, a predefined residual tolerance or maximum number of iterations is set. In the case that the coarsest grid level diverges, the solution is discarded and not used for the correction. This may occur on the first iteration for higher Reynolds number flows. In subsequent iterations where the restricted solution more closely matches the final solution, the coarsest level will converge.

For all cycle types considered, the solution is initialised using Full Multigrid (FMG) initialisation (Figure 3.7a). In FMG, the problem is first solved on the coarsest grid then prolongated to the next finest level where a V-cycle is performed. This is prolongated to the next finest grid and the process is repeated until the finest grid is reached. FMG initialisation uses the solution on the coarsest grid to provide a good initial condition to the first multigrid cycle.

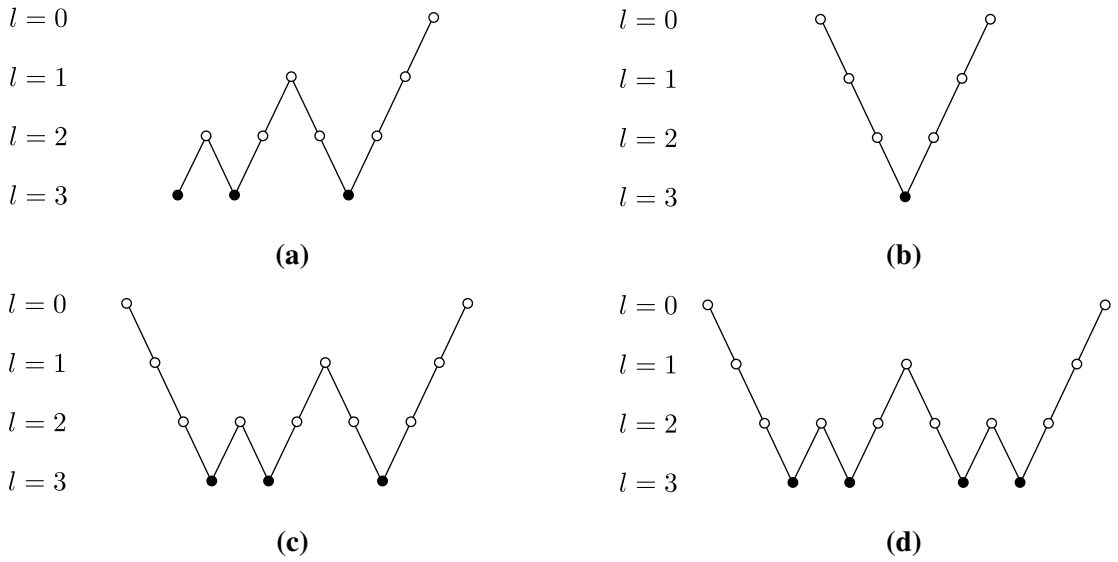


Figure 3.7: FMG initialisation (a), V-cycle (b), F-cycle (c), and W-cycle (d) for a 4 level multigrid. Filled in circles indicate that the solution is solved, while hollow circles indicate that a fixed number of smoothing iterations are performed.

3.5.3 Calculation of Discrete Operators

At a given grid level l, the discrete Navier-Stokes equations $A^l(\boldsymbol{x}) = \boldsymbol{f}^l$ need to be calculated. For the purposes of this discussion, the form of $\boldsymbol{x}$ is not relevant. It can be thought of as a vector of velocities and pressures at each cell in the domain in an arbitrary order. Upon linearisation, this can be represented as

$$A^l\boldsymbol{x} + \boldsymbol{b}^l = \boldsymbol{f}^l, \tag{3.11}$$

where A^l is the linear operator that comes from the linearisation and the vector $\boldsymbol{b}^l$ contains the constants that may arise from the linearisation process, the application of boundary conditions, a deferred correction, or any other source terms. The operator A^l contains the the advection, diffusion, pressure gradients, divergence from the continuity equation, and the MWI. Each of these terms is calculated on the coarse grid in the exact same way they are calculated for a single grid method. For the case of the advection terms which contains the face velocity fluxes, these are calculated using the most recently available velocity values in $\boldsymbol{x}$, or in the case of the first iteration, the restricted velocity field. If boundary conditions are applied through the use of ghost cells, – which are a part of $\boldsymbol{x}$ – then boundary conditions are completely described within the term $A^l\boldsymbol{x}$. However if boundary conditions are applied through modification of the finite volume equations, then this involves modification of the operator A^l and possibly the addition of terms in the vector $\boldsymbol{b}^l$, as would be the case for Dirichlet boundary conditions.

Source terms in the equations which do not come directly from the discrete governing Navier-Stokes equations can be included in $\boldsymbol{f}^l$. An important distinction to make is that $\boldsymbol{b}^l$ is recalculated at each coarse level while $\boldsymbol{f}^l$ appears in the coarse grid equations through the restricted residual. In the case of the problem considered here the only contribution to $\boldsymbol{f}^l$ is the terms on the right hand side of the coarse grid equations on Line 4 of Algorithm 1. This means that the formation of the terms $A^l\boldsymbol{x} + \boldsymbol{b}^l$ is identical between a single and multigrid solver, the only difference is the addition of $\boldsymbol{f}^l$ at the coarse levels which transforms the problem into the coarse grid problem. As a result, it is relatively simple to convert an existing single grid solver into an FAS solver. While it is possible to include some or all of the contents of $\boldsymbol{b}^l$ in $\boldsymbol{f}^l$, this introduces extra complications since the definition of the restriction and prolongation operators for these terms may not be well defined.

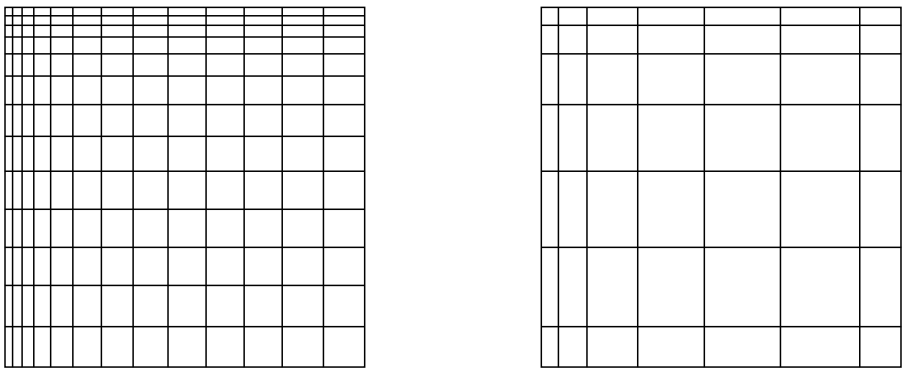
3.5.4 Smoothing

Smoothing at each grid level, including solving at the coarsest grid level is done by direct application of the method introduced in Section 3.2. Being matrix-free, the application of this method to FAS multigrid is very simple, and requires no modification of the basic method apart from the addition of the right hand side terms that arise from the formation of the coarse grid equations. Outer iterations are performed at each grid level, meaning the full non-linear problem is treated on all grid levels. For the pre and post-smoothing steps, a fixed number of outer iterations is performed, generally with a small number of inner iterations (usually less than 10). On the coarsest grid, the problem is solved to a predefined residual. Details on the residual tolerances used for testing are given in Section 3.7. Parallel implementation is done

in a straightforward manner, with the sweeping on each grid level being parallelised using the methods discussed in Section 3.4.

3.5.5 Grid Hierarchy and Transfer Operators

The problem is initialised by first specifying a mesh for the final solution – this will be the finest grid level. The coarse grids are then formed during the solver initialisation by agglomerating two cells together in each direction from the fine grid. In 2D, this means four cells are agglomerated on the fine grid to form a single cell on the coarse grid. In 3D it is eight. Each subsequent coarse level is formed by agglomerating the cells of the next finer grid. In the case where an odd number of cells exists in a particular direction, cells on the domain boundary which are the largest are not agglomerated (Figure 3.8). On a uniform grid, the choice of boundary which is not agglomerated is arbitrary.



(a) **(b)**

Figure 3.8: Fine grid (a) and coarsened grid (b) through agglomeration. Since there is an odd number of cells in each dimension, the cells on the right and bottom boundaries are not agglomerated.

For restriction from the fine grid to the coarse grid, a volume weighted average over all the fine cells that make up a coarse grid is used. Letting J and j be indices of cells on the coarse and fine grids respectively, the restriction is given by

$$\phi_J^{l+1} = \frac{1}{V_J} \sum_{j \in \mathcal{A}(J)} V_j \phi_j^l. \tag{3.12}$$

where V is the volume of a cell, the $\mathcal{A}(J)$ is the set of fine cells that make up the coarse cell J.

Prolongation is done through weighted trilinear interpolation of the cell centres. On a rectilinear finite volume grid, the cell centre of an agglomerated cell on the coarse grid always lies within the cell centres of the fine cells that make up that coarse cell. In 1D, this is given by

$$\phi_j^l = \frac{x_{J+1} - x_j}{x_{J+1} - x_J} \phi_J^{l+1} + \left(1 - \frac{x_{J+1} - x_j}{x_{J+1} - x_J}\right) \phi_{J+1}^{l+1}, \tag{3.13}$$

where $x_J \leq x_j \leq x_{J+1}$. To extend this to 2D and 3D, multiple 1D interpolations are done in stages. First the interpolation is performed in the x direction only to obtain semi-interpolated values. These semi-interpolated values are then interpolated in the y direction, and again in the z direction to complete the interpolation. The only exception is cells on the edge of the fine grid. Since these do not lie between two coarse cells, interpolation is not possible. Instead, injection is used from the corresponding coarse cell.

3.5.6 Under-relaxation

The use of under-relaxation is common practice in CFD solvers to help stabilise the solution, particularly for higher Reynolds numbers. Applied explicitly to an updated variable ϕ, it is given by

$$\phi_P^{\text{new}} = (1-\omega)\phi_P^{\text{old}} + \omega\phi_P^{\text{new}^*}, \tag{3.14}$$

where ϕ^{new} is the final updated variable, ϕ^{old} is the variable from the previous iteration used for under-relaxation, and ϕ^{new^*} is the updated variable without under-relaxation. More commonly however, Eq. 3.14 is applied implicitly by modifying the linearised stencil equations to

$$\frac{a_P}{\omega}\phi_P + \sum_{C\in n(P)} a_C\phi_C = B_P + \frac{1-\omega}{\omega}a_P\phi^{\text{old}}, \tag{3.15}$$

which helps to increase the diagonal dominance of the system. Moreover, the addition of implicit under-relaxation has the same effect as the inclusion of an unsteady term, with timestep dependent on the a_P coefficient [44].

In the FAS multgrid scheme, the choice of ϕ^{old} can have significant effects on the stability and convergence of the method. The obvious choice is to set ϕ^{old} to the value obtained from the previous multigrid cycle for each corresponding grid level. Whilst this is effective at stabilising the solution, it results in significantly reduced convergence rate. As stated before, the addition of under relaxation has a similar effect to adding an unsteady term to the equations, and in this case, the new time level is updated after every cycle. This means every multigrid cycle at best is solving the solution up to the current time level, and not the final steady solution. A slightly better option is to use the restricted value from the finer grid within the same multigrid cycle. While this does result in a more frequent update in ϕ^{old}, the convergence rate was not observed to be significantly better in our tests.

Setting ϕ^{old} to be the solution from the previous outer iteration for a given grid level was found to be optimal. By updating to a value within the current grid level, the solver will always be progressing towards the final steady solution. At the initial iteration, ϕ^{old} could be assigned to

the restricted solution, or to the solution from the last time that grid level was solved. Neither was found to be preferential over the other.

3.6 Implementation Details

The solver is implemented in C++, and makes use of the Eigen 3.4.0 Tensor library [100] for multidimensional array data structures and algorithms. The solver is implemented in a completely matrix free approach, with all coefficients in Eqs. 2.36 – 2.39 stored in their own multidimensional arrays. As described in Section 2.7, since a rectilinear grid is used, simple array data structures are used for coefficient storage and coefficients which do not change with each nonlinear iteration can be stored in 1D arrays. These include the pressure coefficients in the momentum equations, and the velocity coefficients in the continuity equation. The result is a smaller memory footprint when compared to coupled methods that require the assembly of a sparse matrix. Counting only coefficients that need a unique value stored for each cell, the total required storage on a given grid for coefficients is 48 words/cell, compared to 66 words/cell if every coefficient is stored uniquely for each cell.

For each grid level, the mesh, solution fields, restricted solution, residual, finite volume coefficients and any other relevant information unique to a given mesh are stored together in a vector, where each element of the vector corresponding to a different grid level. The single grid smoother is then used at each grid level to solve the coarse grid equations. The solution at a given grid level is obtained in essentially the same way that that would be done for a single grid code. Agglomerating cells means that the number of cells in each dimension is halved. This means that the total number of cells is decreased by a factor of 4 in 2D and factor of 8 in 3D when going to the next coarse grid level. Thus the memory required to store all data for a given grid also decreases by a factor of 4 in 2D and factor of 8 in 3D. An upper bound on the extra memory required relative to a single grid solver can be estimated by considering an infinite number of grid levels with the total memory represented as a geometric series. This upper bound is $\sim 1.33\times$ in 2D and $\sim 1.14\times$ in 3D. In the present implementation of the code, the memory usage is approximately 700 MB/million cells.

Since multidimensional arrays are stored as 1D arrays in memory. As discussed in Section 3.4, it is most cache efficient to sweep the solution update in the direction where data is stored contiguously in memory. In order to allow efficient sweeping in all directions, the boundary conditions and geometry are transformed prior to solution calculation such that the sweep direction is always contiguous in memory. This allows the choice of the most efficient sweeping

direction in terms of the problem, without sacrificing cache efficiency.

3.7 Numerical Tests

Two canonical laminar flow cases were used to demonstrate the accuracy and performance of the method: the 3D lid driven cavity at Reynolds numbers 200 and 1000 and the 3D backwards facing step at Reynolds numbers 100 and 200. These Reynolds numbers were chosen such that the solution does no contain any unsteady behaviour, since attempting to solve an unsteady problem using a steady solver may result in poor or stalled convergence, irrespective of the type of solver used. An unsteady solver within OpenFOAM was use to verify that the solution at these Reynolds numbers is in fact steady. Solver performance and relative accuracy was compared to the coupled solver in ANSYS Fluent, and the SIMPLE scheme in OpenFOAM v9 (simpleFoam). Second order approximations were used for all terms. The QUICK scheme was used for advection in the Fluent coupled solver, while second order upwind was used for the OpenFOAM SIMPLE scheme since QUICK there was found to be unstable for some of the test cases used here. The present coupled method uses the QUICK scheme, applied through a deferred correction to a first order upwind scheme, as described in Chapter 2. All solver tests were run using a single CPU core on a desktop computer with an Intel Core i9-11900F CPU that has a max turbo frequency of 5.20 GHz [101] with 64 GB of dual channel DDR4 memory running at 3200 MHz. Testing was done on a single core to eliminate the possible effect of parallel scaling of the different codes used, so that any difference in performance observed will be largely due to the scheme used. Of course, the relative performance is also implementation dependent i.e. a good implementation of the same scheme will perform better than a poor implementation. Unfortunately this is difficult to measure, and assessing the implementation of other codes is not within the scope of this work, and it is assumed that ANSYS Fluent and OpenFOAM have reasonably efficient implementations of their respective schemes.

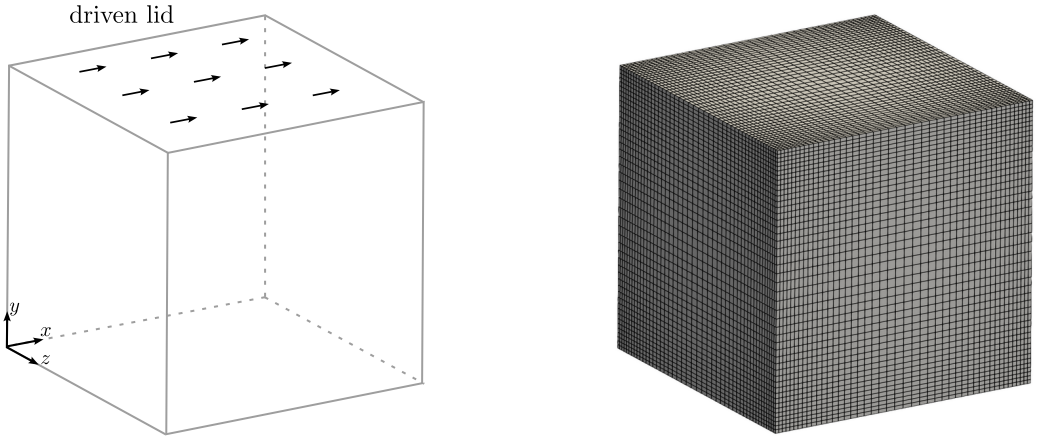
Default settings were used in ANSYS Fluent, and the coupled solver uses Algebraic Multigrid (AMG) to solve the coupled linearised momentum and pressure equations. In the OpenFOAM SIMPLE scheme, Geometric Agglomerated Algebraic Multigrid (GAMG) with a Gauss-Seidel solver was used for the solution of the pressure equation, and Preconditioned Bi-Conjugate Gradient Stabilised with a diagonal based incomplete lower upper factorization preconditioner was used for the momentum equations. It was found that best performance was achieved with the SIMPLE solver when only a small number of linear inner iterations were performed at each outer iteration, typically 1 to 3. For the SIMPLE scheme, standard relaxation factors of 0.3 for

the pressure and 0.7 for the momentum were used. For the coupled solver in ANSYS Fluent, no under-relaxation was used at lower Reynolds numbers. Some under-relaxation had to be used however at higher Reynolds numbers to achieve a stable solution, which will be discussed in detail in the following sections.

In the present method, the coarsest grid was solved to a residual tolerance that is set at least as small as the overall required residual tolerance. For example, if the complete steady equations were to be solved to a tolerance of 10^{-10}, then the coarsest grid was solved to this tolerance too. There was also a maximum number of allowable outer iterations at the coarsest grid set, typically around 200, however the residual tolerance was usually met before this iteration count, especially later in the solution process. A fixed number of smoothing outer iterations between each coarse level was also specified. Two was found to be sufficient. For each outer iteration, inner iterations were performed until a specified target of 10^{-3} was met based on the L_1 norm between the current and previous iterators, or if a maximum number of iterations is reached. This maximum iteration limit was set to 3, and in most cases was reached before the residual tolerance. Convergence residuals were calculated using the approach described in Section 2.8.

3.7.1 Lid Driven Cavity

The 3D lid driven cavity (Figure 3.9a) was tested for all solvers at Reynolds numbers 200 and 1000. The length of all cavity edges was 1. The velocity boundary conditions were Dirichlet on all faces, with all components being set to zero except the driven lid, where the x-velocity was set to 1. The pressure was set to zero normal gradient on all boundary faces.



(a) **(b)**

Figure 3.9: Schematic diagram of 3D lid driven cubical cavity (a) and nonuniform rectilinear finite volume mesh used for 46^3 cells (b).

The second order accuracy of the solver was validated using order of accuracy testing on a

uniform grid. Results were obtained on a uniform mesh of sizes 20^3, 40^3, 80^3, 160^3 and 320^3. The finest grid (320^3) was taken as a reference solution and the order of accuracy was verified by calculating the L_1, L_2, and L_∞ norms of the difference between the velocity magnitude and the pressure on each coarser grid and the reference grid. Since a reference solution was used, the error will approach zero as the grid approaches the reference grid, so the grid size 160^3 was excluded from the calculation since it would approach zero error faster than the convergence rate. In other words, we must ensure the fine grid is sufficiently finer than the coarse grids used to determine the order of accuracy. Results for $Re = 200$ are shown in Figure 3.10 which show that second order accuracy is achieved under all norms for both velocity magnitude and pressure on the 40^3 ($\Delta x = 0.025$) and 80^3 ($\Delta x = 0.0125$) grids.

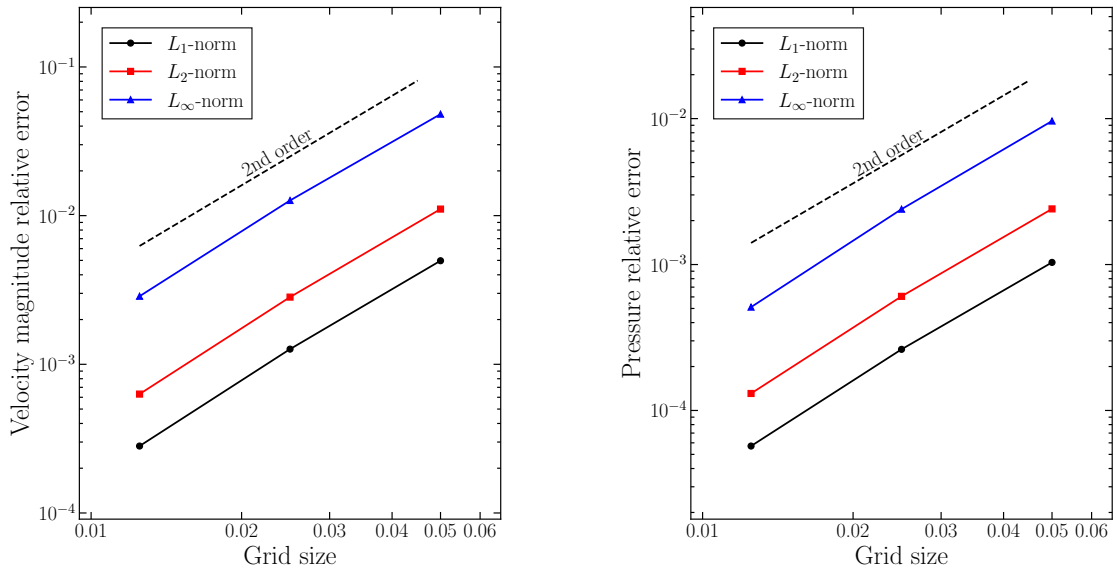
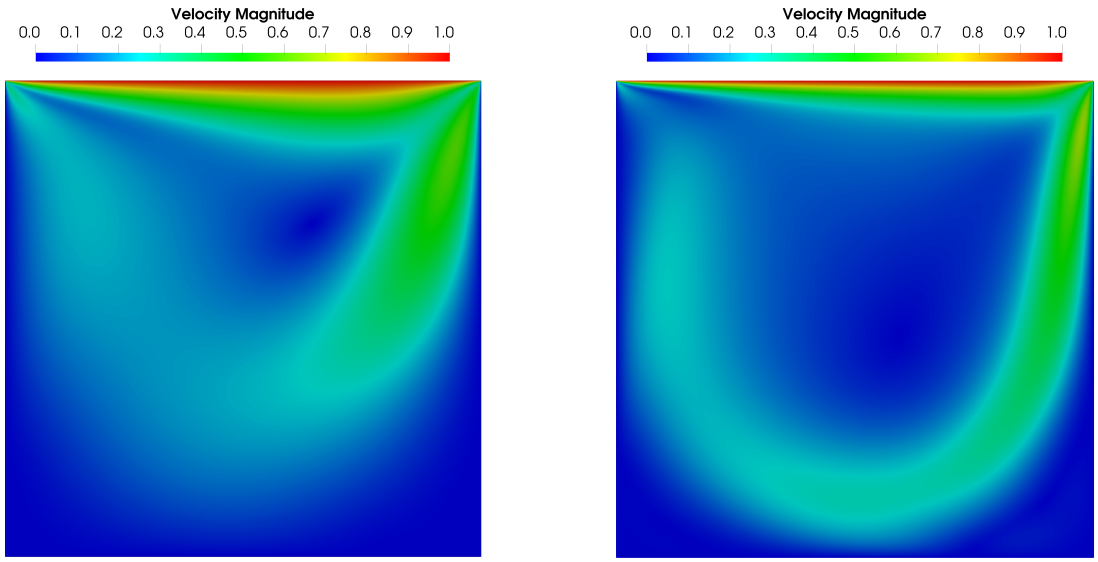


Figure 3.10: Velocity magnitude (a) and pressure (b) error norms relative to solution computed on a 320^3 uniform grid ($\Delta x = 0.003125$) for the 3D lid driven cavity on a uniform mesh with $Re = 200$.

Validation and benchmark tests were also conducted on nonuniform rectilinear meshes with an overall cell growth ratio of 2.5 from the edge to the centre of the domain (Figure 3.9b). This results in a maximum growth ratio between adjacent cells of 1.09596 on the coarsest grid and 1.0086 on the finest grid, and a maximum cell aspect ratio of 2.5. Results were obtained for grid sizes 22^3, 46^3, 100^3, and 216^3. For the FAS scheme, the coarsest grid (22^3) was solved on a single grid level, that is, multigrid was not used. The next finest grid was solved using 1 multigrid level, and 1 grid level was added to each subsequent grid, so the finest grid was solved using 3 coarse grid levels (4 grid levels total). The same mesh was used for all solvers, including ANSYS Fluent and OpenFOAM. Velocity magnitude contours for the solution obtained using

the present method on the 216^3 grid are shown in Figure 3.11. The L_1-norm of the relative velocity difference between the present method and ANSYS Fluent and OpenFOAM for each of these grid sizes, for both $Re = 200$ and $Re = 1000$ are shown in Figure 3.12. The results show good agreement of the solution with these solvers, with the errors decreasing as the grid is refined. The small errors observed likely come from numerical differences between each code implementation despite all three codes solving the same discrete equations. Such errors are expected however, and their root cause is difficult to determine without detailed analysis of the source code and numerical implementation.



(a) **(b)**

Figure 3.11: Velocity magnitude contours at centre plane for 3D lid driven cavity with $Re = 200$ (a) and $Re = 1000$ (b) obtained with the present method on a 216^3 grid.

The relative convergence rate between the different multigrid cycles in Figure 3.7 for the lid driven cavity on the finest grid is shown in Figure 3.13 with the CPU time for all grid sizes tested given in Table 3.1. The F and W-cycles significantly outperform the V-cycle in terms of number of iterations and solver time. However the difference in performance between F and W-cycles is very small, with the W-cycle slightly outperforming the F-cycle. Despite each W-cycle containing more computational work than a single F-cycle, the W-cycle spends most of this work on the coarser grids, which are relatively cheap. Also, for all cycle types, as the Reynolds number is increased, the number of multigrid cycles required to reach a given residual is higher.

Figure 3.14 shows the time for each solver to reach a residual of 10^{-10} for all grid sizes tested for the nonuniform grid. Also shown is the present method without multigrid, i.e. solved on a single grid. Even on the single grid, the present method outperforms the OpenFOAM SIMPLE

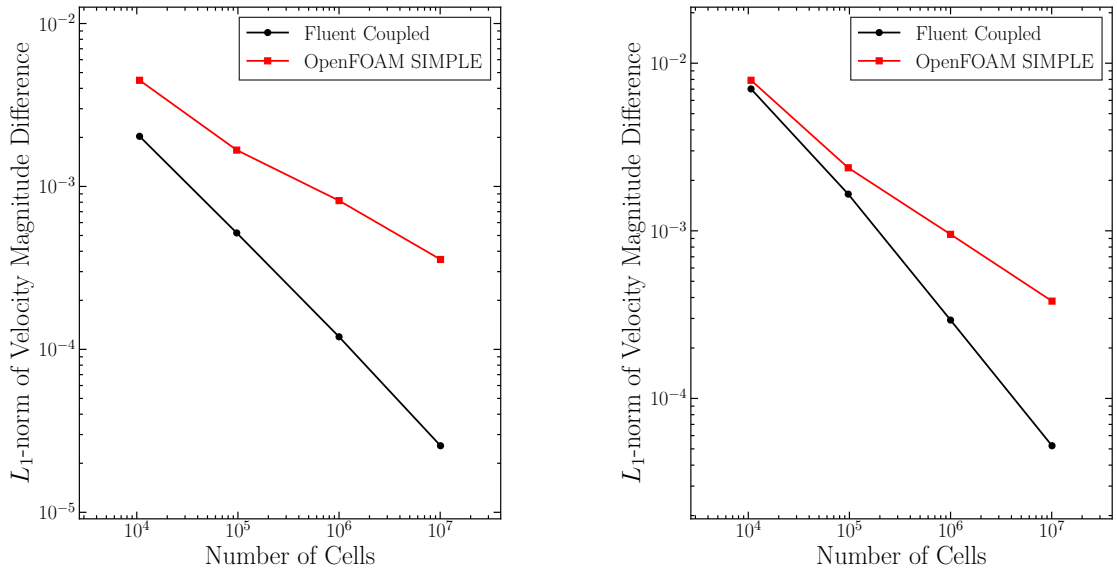


Figure 3.12: L_1-norm of velocity magnitude difference vs grid size between present method and other standard solvers for $Re = 200$ (a) and $Re = 1000$ (b) for 3D lid driven cavity.

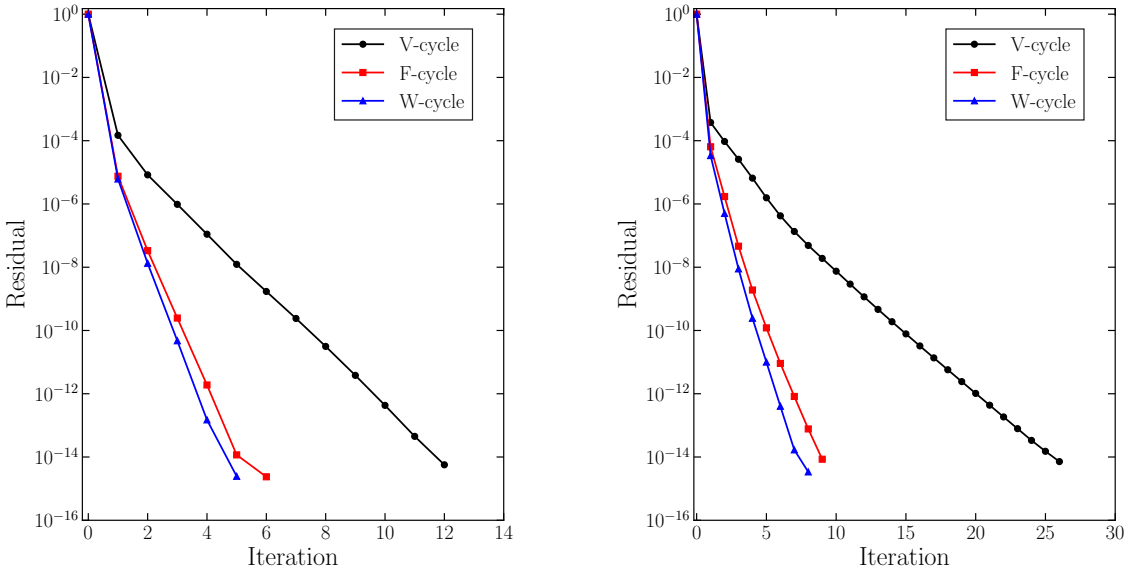


Figure 3.13: Local continuity residual convergence history for 3D lid driven cavity on a 312^3 nonuniform grid for each multigrid cycle with $Re = 200$ (a) and $Re = 1000$ (b).

scheme and Fluent coupled solvers. Using it with FAS multigrid gives further significant performance improvements, with nearly two orders of magnitude speed-up on the finest grid. Furthermore, when multigrid is used near linear scaling of solver time with problem size is observed, demonstrating ideal performance of the multigrid method. This is not the case with the other methods. Even though the coupled solver in Fluent uses AMG, it is only for the linearised

Table 3.1: Solver time vs. grid size for each multigrid cycle for the 3D lid driven cavity at $Re = 200$ (a) and $Re = 1000$ (b). All cases were solved to a residual of 10^{-14}.

	Solver Time (s)		
Grid Size	**V-cycle**	**F-cycle**	**W-cycle**
22^3	0.36	0.39	0.36
46^3	14.50	14.96	14.62
100^3	64.4	61.56	67.77
216^3	558.98	452.65	428.69

(a)

	Solver Time (s)		
Grid Size	**V-cycle**	**F-cycle**	**W-cycle**
22^3	1.20	1.23	1.23
46^3	23.03	24.48	24.3
100^3	120.22	96.98	96.88
216^3	1210.14	677.22	678.99

(b)

equations at a given outer iteration, so the non-linearities are not accounted for on the coarser grids. This fact demonstrates the utility of the FAS multigrid method.

Figure 3.15 shows the continuity residual convergence history on a single grid (with no multigrid) for the different possible plane and line sweeping directions. For both Reynolds numbers, there appears to be at least one direction of updating which provides a better convergence rate. Interestingly however, the optimal direction differs with Reynolds number.

Figure 3.16 shows the same results with the FAS multigrid solver. When the FAS scheme is used with the smoother, the effect of direction becomes very small – almost negligible. When the problem is solved on the coarse grid, it is solved to a small residual in a small number of iterations (relative to the number of iterations required on the fine grid), regardless of sweep direction. Even if a particular sweep direction is more efficient on the coarse grid, the difference is very small since the cost of solving the coarse grid is small anyway. As a result, the effect of sweep direction is not as significant when an FAS multigrid scheme is used with the 3D lid driven cavity case.

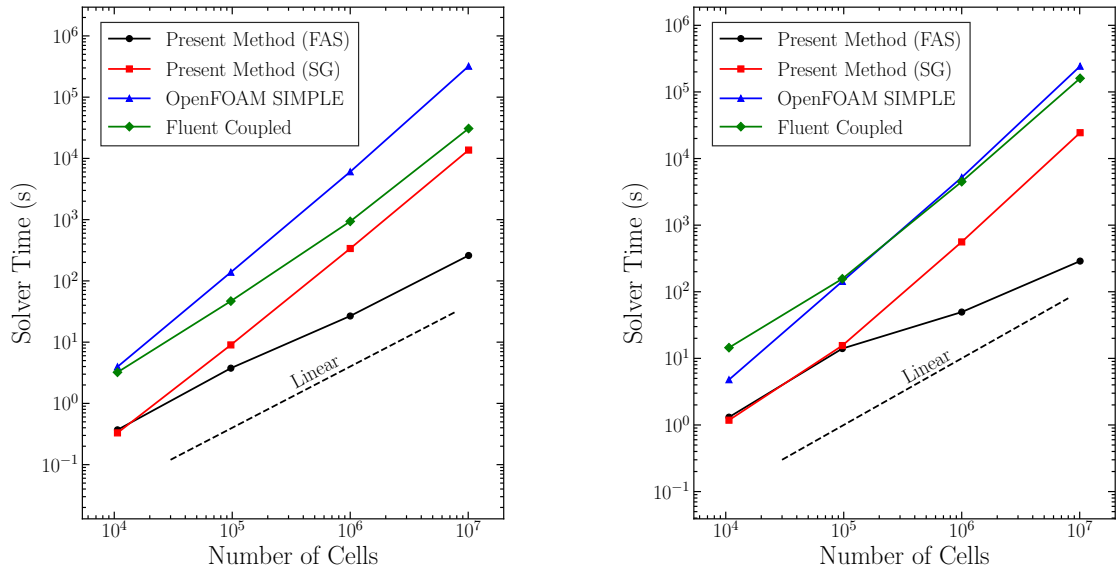


Figure 3.14: Comparison of solver times with other standard steady solvers at $Re = 200$ (a) and $Re = 1000$ (b) for the 3D lid driven cavity problem. "SG" refers to the coupled smoother on a single grid using the optimal plane and line sweeping direction. All solvers were set to solve the problem to a residual of 10^{-10}.

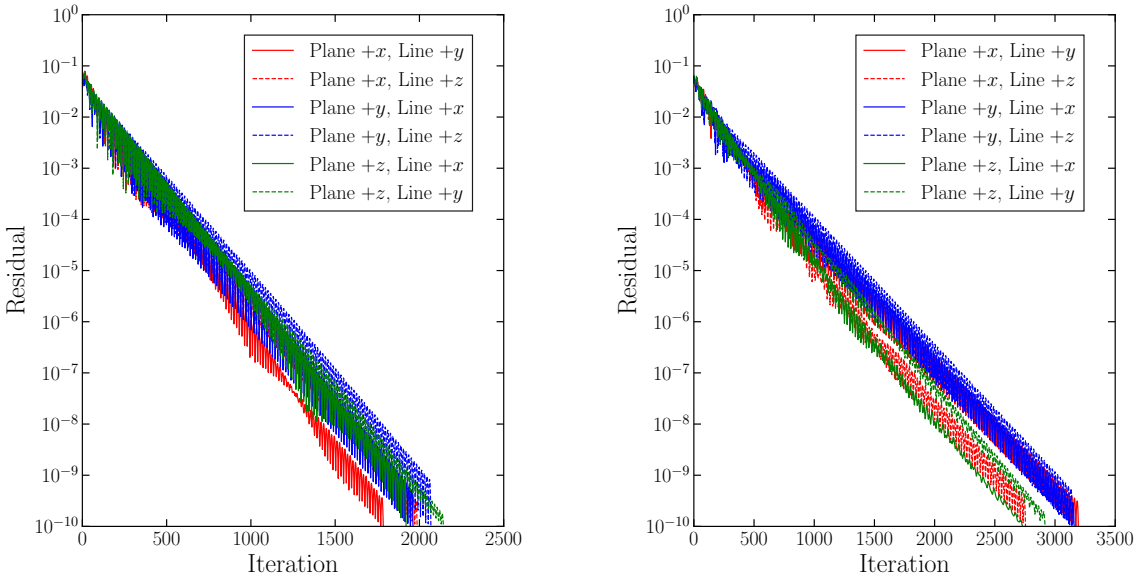
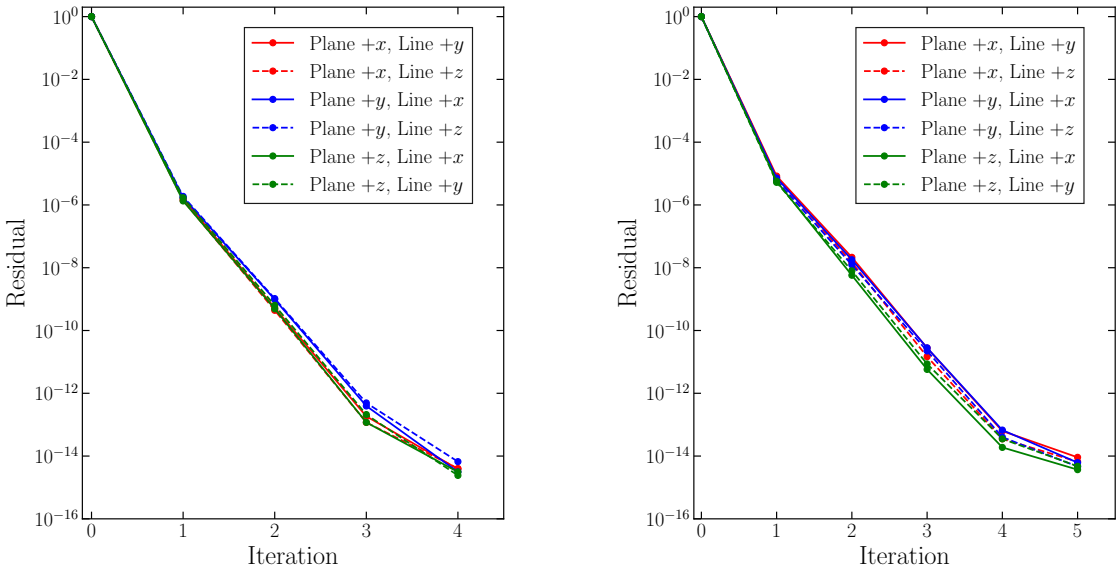


Figure 3.15: Lid driven cavity continuity residual convergence history for $Re = 200$ (a) and $Re = 1000$ (b) for different line and plane sweeping directions when solved without FAS multigrid, on a single grid.



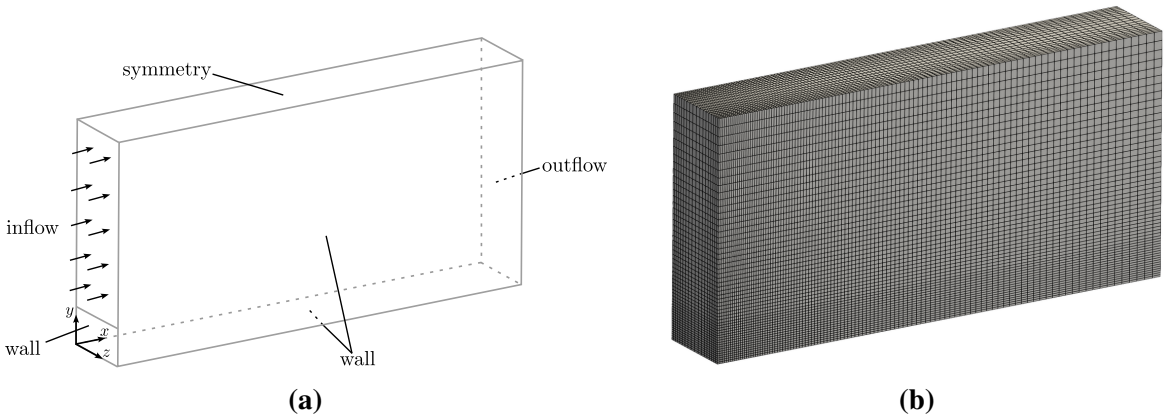
(a) **(b)**

Figure 3.16: Lid driven cavity continuity residual convergence history for $Re = 200$ (a) and $Re = 1000$ (b) for different line and plane sweeping directions when solved with FAS multigrid using W cycles.

3.7.2 Backwards Facing Step

The same validation tests were conducted for the 3D backwards facing step (Figure 3.17a) for the Reynolds numbers 100 and 200. The backwards facing step has no entrance region, and the effect of the wall comes from the inflow boundary face where the velocity of the cell faces corresponding to the step are set to zero. At the inflow, the velocity is fixed to 1, and the pressure is set to be zero gradient. At the outflow, all velocity variables are set to be zero normal gradient, and the pressure is set to zero. At the walls, all velocity components are set to zero, and the pressure is set to have a zero normal gradient. On the symmetry face, all variables are set to have zero normal gradient except the normal component of velocity to the face, which is fixed to be zero. The domain has a length of 6 in the x direction, a width of 1 in the z direction, and a height of 3 in the y direction, with the step height being 0.5.

The four nonuniform grid sizes tested were $48 \times 24 \times 8$, $108 \times 54 \times 18$, $228 \times 114 \times 38$, and $492 \times 246 \times 82$. The maximum growth ratio between adjacent cells in the mesh used was 1.03252 on the coarsest grid and 1.00307 on the finest grid, with maximum cell aspect ratios of 4.78049 on the coarsest grid and 4.79227 on the finest grid. No under-relaxation was applied for either Reynolds numbers. Like the lid driven cavity, the coarsest grid ($48 \times 24 \times 8$) was solved on a single grid level (no multigrid was used). The next finest grid was solved using 1 multigrid level, and 1 grid level was added to each subsequent grid, so the finest grid was solved using 3



(a) **(b)**

Figure 3.17: Schematic diagram of 3D backwards facing step (a) and nonuniform rectilinear finite volume mesh used for $108 \times 54 \times 18$ cells (b).

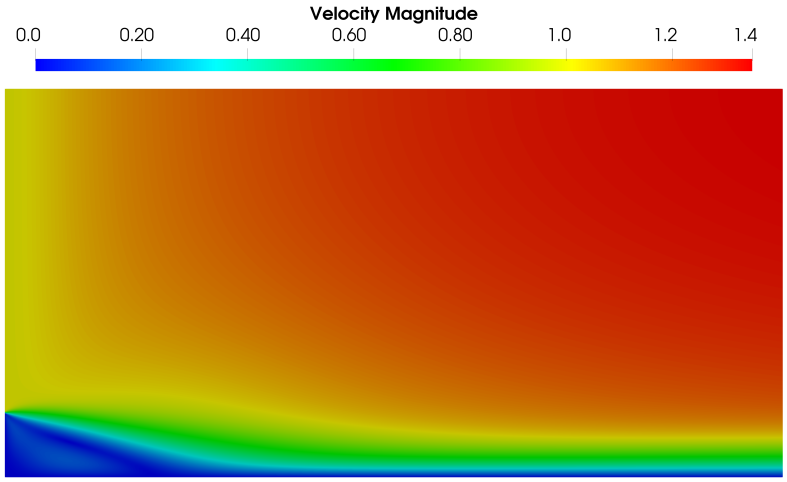
coarse grid levels (4 grid levels in total). Again, the same mesh was used for the ANSYS Fluent and OpenFOAM solutions. Velocity magnitude contours for the solution obtained using the present method on the $492 \times 246 \times 82$ grid are shown in Figure 3.18.

The L_1-norm of the relative velocity difference between the present method and ANSYS Fluent and OpenFOAM for each of these grid sizes, for both $Re = 100$ and $Re = 200$ are shown in Figure 3.19. As before, their magnitudes decrease to zero as the grid is refined, indicating good agreement with all solvers.

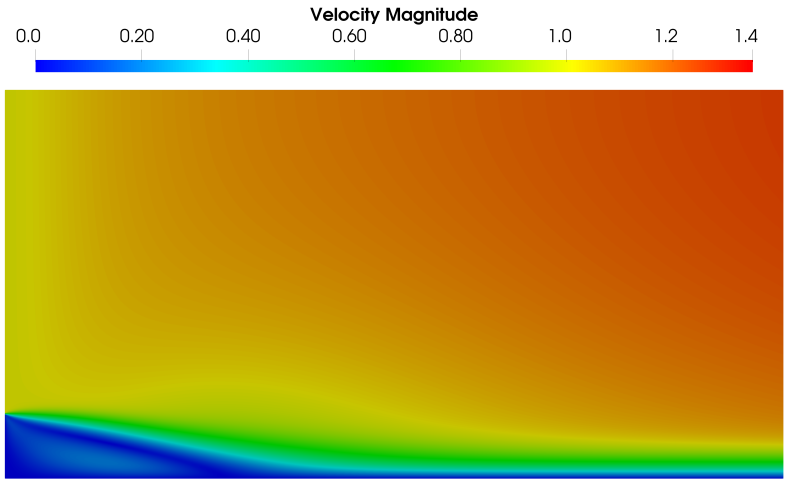
Convergence rates for each multigrid cycle are given in Figure 3.20, and the solver times in Table 3.2. Similar relative performance between the cycle types to the 3D lid driven cavity are observed, with the F and W-cycles performing similarly, and outperforming the V-cycles. Interestingly, for the $Re = 100$ case, despite the convergence being similar or slightly better for the W-cycle compared to the F-cycle in terms of number of iterations, the F-cycle outperforms in terms of solver time due to its lower cost per cycle.

Solver time comparisons for the 3D backwards facing step with Fluent and OpenFOAM are shown in Figure 3.21. As with the lid driven cavity, there is approximately a one order of magnitude speed up when going from a single grid to the FAS scheme with the present method. At $Re = 100$ on the finest grid, the single grid solver is comparable to the Fluent coupled solver, but at $Re = 200$ the single grid method outperforms. It can also be seen that on the two coarsest grids, the single grid method outperforms the FAS scheme. At such coarse grids, the overhead associated with the FAS scheme (restriction, prolongation, correction) is significant relative to the cost of solving the problem, which is on a coarse grid anyway.

Figures 3.22 and 3.23 show that the choice of plane and line sweeping direction for the 3D backwards facing step seems to have a greater effect on convergence rates at higher Reynolds numbers and stability. Unlike the 3D lid driven cavity, some plane and line sweeping combinations



(a)



(b)

Figure 3.18: Velocity magnitude contours at centre plane of backwards facing step with $Re = 100$ (a) and $Re = 200$ (b) obtained with the present method on a $492 \times 246 \times 82$ grid.

were unstable. For the single grid case, there are two unstable sweeping directions for both Reynolds numbers. When multigrid was used, there was still two unstable sweeping directions when $Re = 100$, but for $Re = 200$ there was two unstable directions, and one direction which had stalled convergence (which can be said to be unstable as well). This indicates that the FAS scheme can have a negative effect on the stability and robustness of the method in some cases. As was found with the lid driven cavity, the difference in convergence rate between the sweeping directions becomes smaller when the FAS scheme was used.

3.7.3 Discussion

Testing on a single grid indicated that the choice of sweeping direction had an impact on the convergence rate, and this choice was problem dependent. However, when using a multigrid

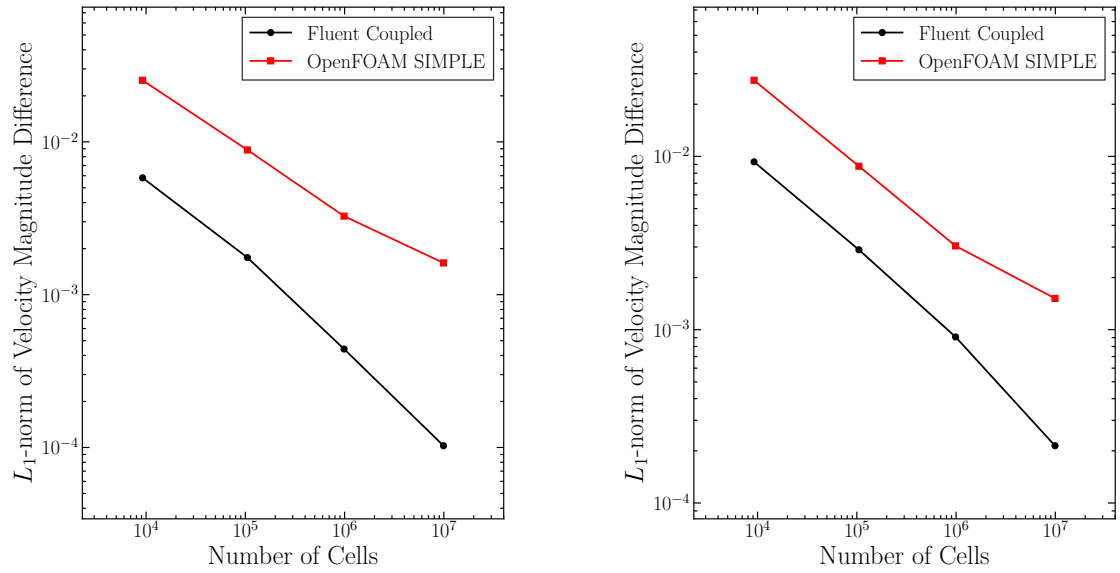


Figure 3.19: L_1-norm of velocity magnitude difference vs grid size between present method and other standard solvers for $Re = 100$ (a) and $Re = 200$ (b) for 3D backwards facing step.

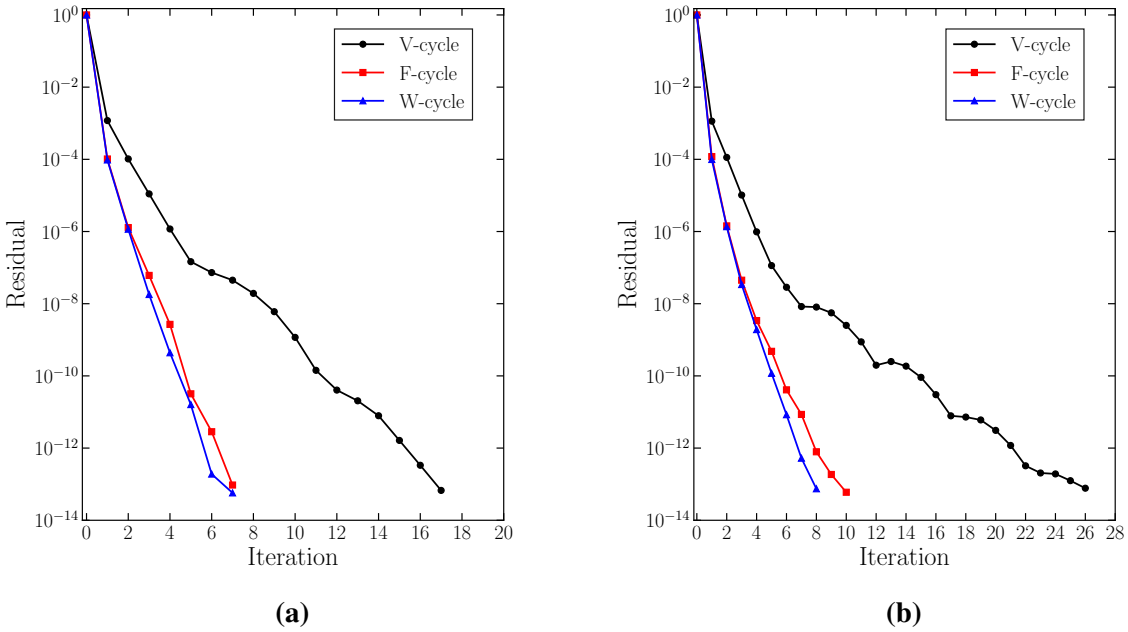


Figure 3.20: Local continuity residual convergence history for 3D backwards facing step on a $492 \times 246 \times 82$ nonuniform grid for each multigrid cycle with $Re = 100$ (a) and $Re = 200$ (b).

scheme, this choice no longer has a significant effect on convergence rate – at least for the problems in this chapter. In a sense, this is good as it simplifies the setup of the solver, particularly since the optimal sweeping directions for the single grid case was problem dependent, and is difficult to know a priori. There is no longer a need to determine the optimal sweeping direction. However, as was shown in the case of the backwards facing step, choice of sweeping direction

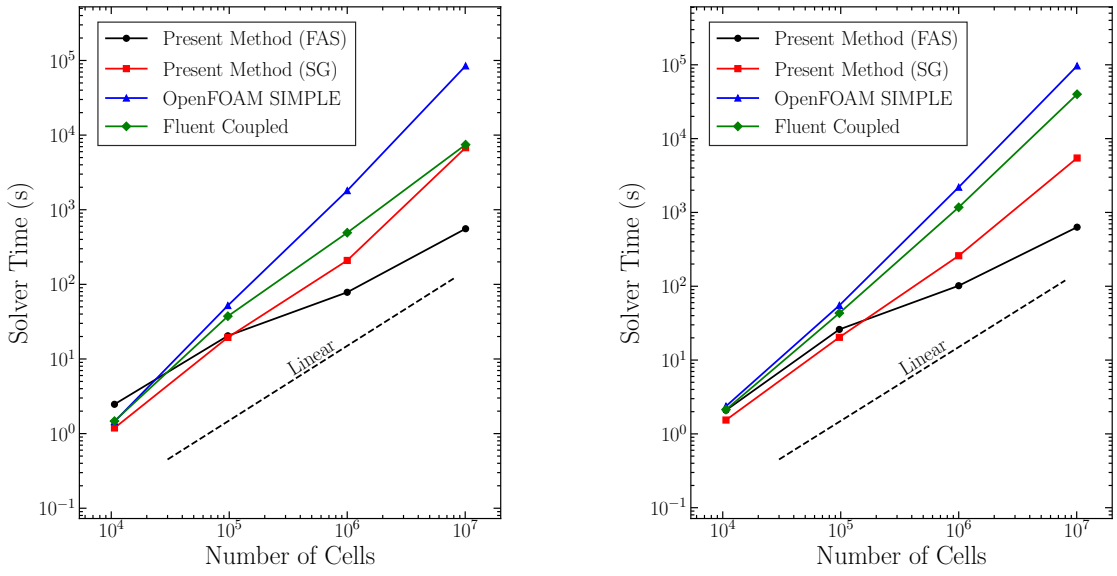
Table 3.2: Solver time vs. grid size for each multigrid cycle for the 3D backwards facing step at $Re = 100$ (a) and $Re = 200$ (b). All cases were solved to a residual of 10^{-14}.

	Solver Time (s)		
Grid Size	**V-cycle**	**F-cycle**	**W-cycle**
$48 \times 24 \times 8$	2.49	2.45	2.47
$108 \times 54 \times 18$	20.87	20.37	20.44
$228 \times 114 \times 38$	102.93	80.63	78.30
$492 \times 246 \times 82$	782.36	524.67	556.32

(a)

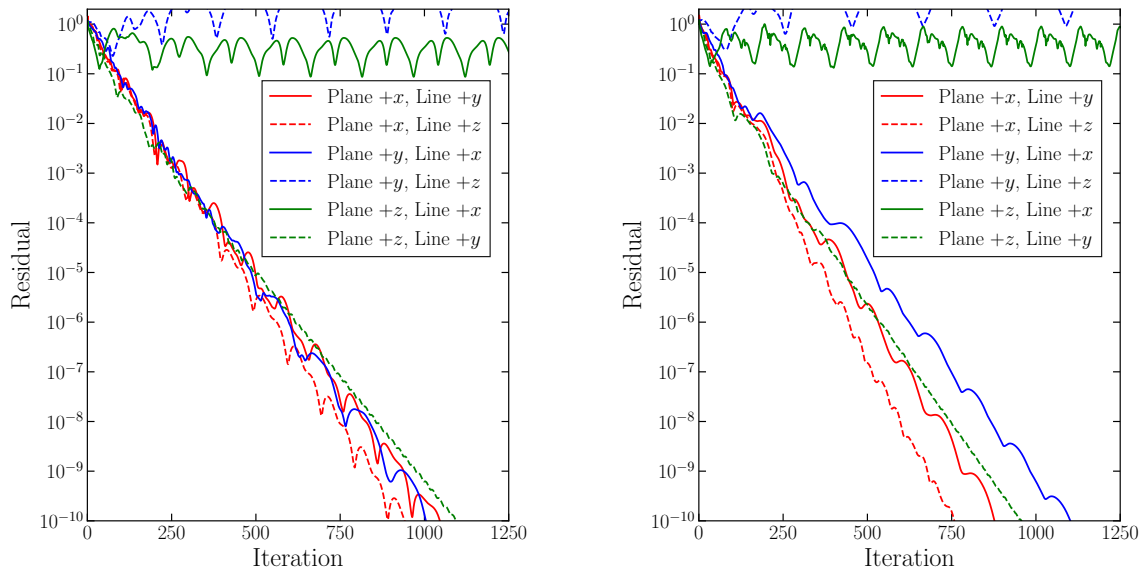
	Solver Time (s)		
Grid Size	**V-cycle**	**F-cycle**	**W-cycle**
$48 \times 24 \times 8$	2.11	2.08	2.09
$108 \times 54 \times 18$	26.48	25.94	26.01
$228 \times 114 \times 38$	138.85	106.42	101.66
$492 \times 246 \times 82$	1201.97	755.14	631.41

(b)



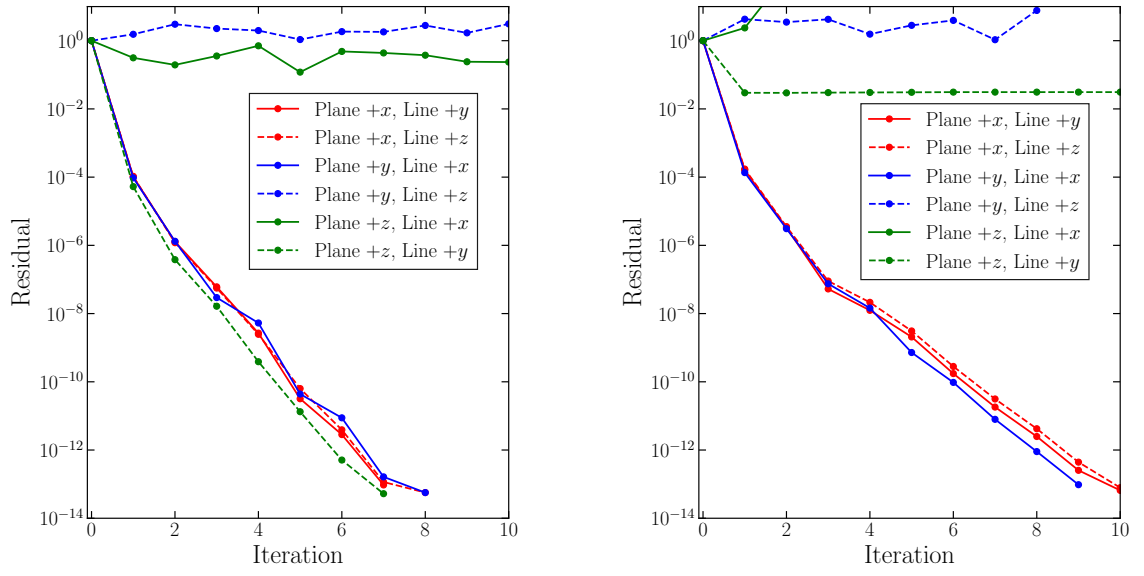
(a) **(b)**

Figure 3.21: Comparison of solver times with other standard steady solvers at $Re = 100$ (a) and $Re = 200$ (b) for the 3D backwards facing step problem. "SG" refers to the coupled smoother on a single grid using the optimal plane and line sweeping direction. All solvers were set to solve the problem to a residual of 10^{-10}.



(a) **(b)**

Figure 3.22: Backwards facing step continuity residual convergence history for $Re = 100$ (a) and $Re = 200$ (b) for different line and plane sweeping directions when solved without FAS multigrid, on a single grid.



(a) **(b)**

Figure 3.23: Backwards facing step continuity residual convergence history for $Re = 100$ (a) and $Re = 200$ (b) for different line and plane sweeping directions when solved with FAS multigrid using W cycles.

can result in some directions being unstable. The backwards facing step is a more directionally dominant flow when compared to the 3D lid driven cavity, so this behaviour is likely to be observed again for other problems which have a dominant flow direction. This is an important factor to be aware of when using such methods.

It should be noted, that the solver was always performing solution sweeps in the direction that provided greatest data locality. That is, variables were updated such that solution and coefficient arrays were traversed contiguously in memory. This was done by rotating the problem coordinates. This means that irrespective of sweep direction, the cost per iteration is always the same. Not doing so will result in an increase in time per iteration, and likely result in slower solver times.

The two test cases above used relatively low Reynolds numbers, and grid expansion ratios that were quite small. Increasing the Reynolds number, or using highly stretched grids make the discrete system more ill-conditioned, resulting in worse convergence rates and stability of the solver. This behaviour is common to all iterative solvers, including the present method. Indeed the results above indicate that with increased Reynolds number, the number of iterations, and hence solver time increased, and the same would be observed with more stretched grids. Nonetheless, compared to segregated methods, the convergence rate and robustness of the present method will be better, for the same reasons that it is in the cases above.

There is an important distinction to be made between the present method and other approaches which solve the equations by lines and planes, such as that by Paisley and Bhatti [77, 82]. In their approach, each plane (in 3D) and line (in 2D) was solved to a relatively large degree of accuracy, and in the 2D case when a tridiagonal matrix solver was used, the equations at each line were solved exactly. Spending large amounts of computation time solving a single line has been found here to be inefficient, as these operations contribute little to the propagation of elliptic information throughout the domain in the swept direction. It is for this reason that Paisley and Bhatti observed that the most optimal sweeping orientation was across the dominant flow direction, since most elliptic pressure information is coupled in the streamwise direction. In the present method, it was found through testing that it was most optimal to perform a single sweep when solving each line and plane. This is likely why the disparity in convergence rate between sweeping directions for the 3D lid driven cavity was not as significant as those observed by Paisley and Bhatti.

As mentioned above, optimal convergence was achieved by performing a relatively small number of sweeps within each outer iteration, typically this was 3 symmetric sweeps. While doing even fewer sweeps resulted in a smaller number of total sweeps to reach convergence, the cost of updating non-linear coefficients so frequently became significant. This general observation indicates that within each outer iteration, little computational effort should go into solving the linearised system, and that closer coupling between the linear solver, and the non-linear problem leads to faster convergence. Since reduction in number of sweeps per outer

iteration no longer reduces computational cost, one can conclude that perhaps the bottleneck of this present method is now the convergence of the elliptic linear system, namely the solution of the pressure equation. One approach to address this is through multigrid methods.

It is clear that incorporating the coupled smoother method into an FAS scheme yields significant performance improvements – up to two orders of magnitude in some cases. In fact, this would be the case for any solver on a single grid. However, the coupled smoother which already provides good convergence rates on a single grid results in a very efficient method overall when used with FAS multigrid. Being matrix free, the conversion from a single grid method to a multigrid one is relatively straightforward. To further highlight the efficiency improvements obtained by adopting a multigrid method, Table 3.3 shows the percentage of time spent in different parts of the solver for the 3D lid driven cavity for each of the different multigrid cycles tested. Extra operations that come as a result of using an FAS multigrid scheme account for less than 5% of the total solver time, while more than 85% of the solver time is spent performing smoothing. The relative distribution between each of the different multigrid cycle types is roughly the same since the distribution of work on each grid level is identical. The only difference between the cycles is the amount of time spent at a given grid level and the order of traversal between grid levels.

Table 3.3: Percentage of total solver time spent in different parts of the solver for the 3D lid driven cavity with $Re = 200$ on a 216^3 nonuniform grid. "Other" refers to all other operations that occur during a solver such as convergence checks, and writing data to file. "Multigrid Operations" accounts for all operations that arise due to the use of an FAS scheme including restriction, prolongation, calculation of residuals, and application of the coarse grid correction.

	% of Total Solver Time		
Operation	**V-cycle**	**F-cycle**	**W-cycle**
Solution Sweeping	86.80	86.72	86.71
Updating Coefficients	7.76	7.71	7.77
Multigrid Operations	3.37	4.02	3.94
Other	2.07	1.55	1.58

Finally, there are a number of parameters in the FAS scheme which can be tuned to give further improved performance. This includes the number of pre and post-smoothing iterations at each grid level, the number of iterations to perform on the finest grid level, the residual to solve on the coarsest grid, and the maximum allowed number of iterations on the coarsest grid. The values for these parameters described above were obtained from a trial and error approach, and the same parameters were generally used across all problems. However, the optimum for each

problem is different. In any case, the results here show that good results can be obtained across a range of problems for a fixed set of parameters.

3.7.4 Parallel Performance

The laminar 3D lid driven cavity case from Section 3.7.1 is used to test the parallel performance and efficiency of the present implementation. As discussed in Secs. 3.4 and 3.6, the present method is currently only parallelised for shared memory using OpenMP. Testing was done on two different computer systems given in Table 3.4. System 1 is representative of a typical high end consumer grade desktop machine at the time of writing, while System 2 is representative of a modern HPC cluster node. The code was compiled on each of the two systems in Table 3.4 using the same compiler (GCC 12.2.0 [102]), with the same compiler flags. In particular, the only performance related flags used where `-O3` and `-march=native`. Strong and weak scaling tests on these two systems are performed for the three colour orderings in Section 3.4. Identical discretisation and boundary conditions from Section 3.7.1 are used, with only a single sweeping direction used. Though it should be noted that the exact solver parameters used have little effect on the results here so long as they are consistent throughout testing. The problem is solved to a residual of 10^{-10} to ensure a sufficiently large number of operations are performed for accurate scaling results. It should be understood that while the focus of the testing here is the effect of the colour pattern used to parallelise the solver method, this is not the only factor influencing parallel performance, but it is the most significant. Other parts of the code, such as the updating of nonlinear coefficients done in parallel, and will have their own scaling behaviour. Between tests, only the colour ordering is changed, so any differences in performance are solely due to this.

Table 3.4: Specifications of the two computer systems used in parallel scaling tests.

System 1	Intel Core i9-11900F (Rocket Lake) 8 core CPU [101] with 64 GB of dual channel DDR4 RAM.
System 2	2×24 core Intel Xeon Platinum 8274 (Cascade Lake) [103], each with 2 NUMA nodes with 48 GB RAM, giving 192 GB RAM total (provided by the National Computational Infrastructure (NCI) Australia Gadi research supercomputer [104]).

System 2 in Table 3.4 contains two Non-Uniform Memory Access (NUMA) nodes per socket [105]. While from a programming point of view all memory is accessed in the same way, this means that each core belongs to one of four NUMA nodes that have a 48 GB bank of RAM that can be accessed with much lower latency and higher throughput than memory that belongs to other nodes. An efficient (NUMA aware) parallel program will ensure that data is stored on the

NUMA node which contains the processor that accesses it the most. There are various ways to do this, however in the present OpenMP implementation, this is done using the "First Touch Data Placement Policy" (first touch policy). In the first touch policy, memory is allocated to the NUMA node which contains the processor that initialised the memory [106]. Regarding the current use case, this means that the initialisation of data should be done in parallel, so that cores which will be most likely to use this data allocate them on the memory bank which resides on the same NUMA node. In practise, whether this results in an efficient implementation depends on how the data is accessed. Certainly, the different colour orderings presented in Section 3.4 make things more complicated, and it is likely that cores will have to frequently access memory on other nodes.

Processors that reside on the same NUMA node share memory bandwidth to that node's local RAM, and they may also share some levels of cache (typically the last-level cache), depending on the processor architecture. This means that the particular NUMA nodes that a thread is bound to, and the distribution of threads across NUMA nodes is important. This is called "thread affinity". Two simple cases are considered in the testing done here which is controlled with the OpenMP `PROC_BIND` environment variable: when all the processors are grouped nearby on physical nodes which will improve cache performance, but reduce memory bandwidth (`PROC_BIND=close`, referred to as "close processor affinity"); and when all the processors are grouped as far apart as possible, which maximises memory bandwidth and cache size, with potential greater synchronisation overhead (`PROC_BIND=spread`, referred to as "spread processor affinity"). Close processor affinity can be beneficial if processors access lots of shared data, while a spread processor affinity will give better performance for memory intensive workloads that do not access much shared data between nodes. In the case of a CFD solver, with many different types of operations (solution sweeping, updating coefficients, calculation of other relevant quantities) it is possible that certain parts of the code will benefit from different thread affinity and memory layouts, and obtaining an optimal implementation requires a more detailed analysis of the implementation and data access patterns. A detailed analysis of this aspect lies outside the scope of this thesis, and therefore a higher-level analysis is presented.

Strong scaling test results using a range of grid sizes for System 1 and System 2 with both `PROC_BIND=close` and `PROC_BIND=spread` are given in Figs. 3.24, 3.25, and 3.26. For System 1, smaller grid sizes show better speedup over all, with the speedup being better when point colouring is used, followed by lines, then planes. For larger grid sizes (which are of more interest) better speedup is obtained with plane parallel ordering, but only by a small amount. For System 2 we see better overall scaling as the processor count is increased, but

as with System 1, there is eventually a point where no more speedup is obtained when more processors are used. This is typical behaviour expected from shared memory parallel systems where eventually memory bandwidth becomes the bottleneck in scaling rather than processor performance. System 2 has significantly greater memory bandwidth than System 1 (with the Intel Xeon 8274 offering more DDR4 memory channels than the i9-11900F), and System 1 has better single threaded performance (the i9-11900F has a higher max turbo frequency than the Xeon 8274) and so we expect better scaling from System 2. As will be shown below, this does not necessarily mean better overall performance in terms of CPU time.

Interestingly, while System 1 had better speedup for smaller grid sizes, System 2 on the other hand gets better speedup with larger grid sizes. Although the Xeon 8274 has greater total last level cache compared to the i9-11900F (35.75 MB vs. 16 MB), the lower core count of the i9-11900F means that each core gets a greater slice of this cache. This coupled with the higher core frequency means that for smaller problems where the working dataset can fit in cache, speedup will be good for smaller problems. For larger problems where the working dataset significantly exceeds the cache size, the larger memory bandwidth of the Xeon 8274 becomes a more important factor in relative scaling performance. Additionally, the i9-11900F supports higher memory frequencies, which reduces memory access latency. The lower core count of the i9-11900F also means less synchronisation overhead across threads – which for small problems can be significant relative to problem size.

We now consider the effect of having close (Figure 3.25) and spread (Figure 3.26) processor affinity for System 2. For the case of close affinity, we can see a drop in speedup once the number of processors has increased beyond 24 – the number of processors on each socket. With `PROC_BIND=close`, when 24 or less processors are used, all processors are on the same socket, and so all data is stored on the local RAM belonging to that socket. Once the number of processors exceeds 24, half the processors are bound to one socket, and the other half is bound to the other. Any data that is required by a processor on one socket from RAM on the other socket will be transferred with a much lower latency. The drop in speedup in Figure 3.25 indicates that there is quite a significant amount of shared data between processors. With the spread processor affinity (Figure 3.26), this drop is not observed, however, the speedup when 24 or less processors is used is less than that of the close affinity case. Beyond 24 processors, the speedup behaviour is almost identical between the two cases. This indicates that parallel performance is limited by memory bandwidth and latency.

Weak scaling test results for the same three cases are given in Figs. 3.27, 3.28, and 3.29. In general, the results are typical for a memory bound application, where in all cases, the efficiency

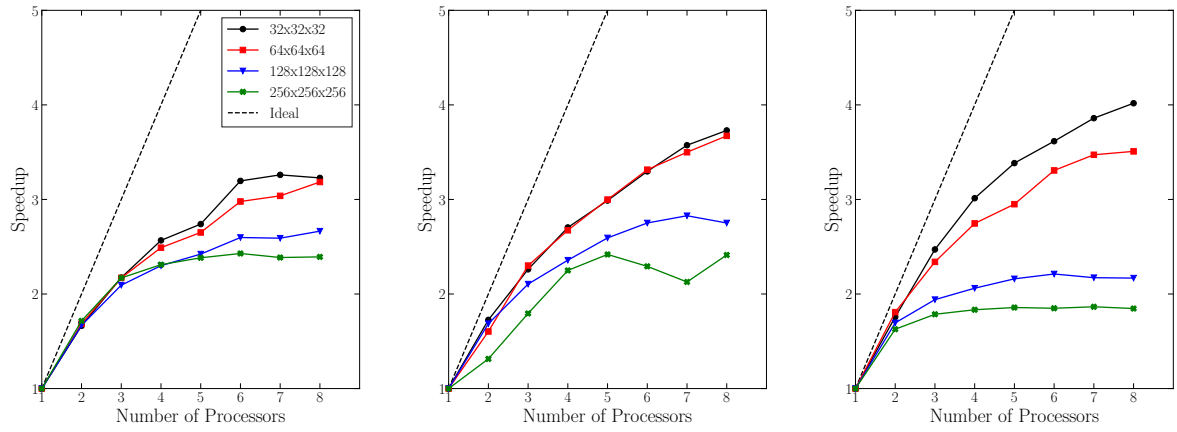


Figure 3.24: Strong scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 1 (Table 3.4).

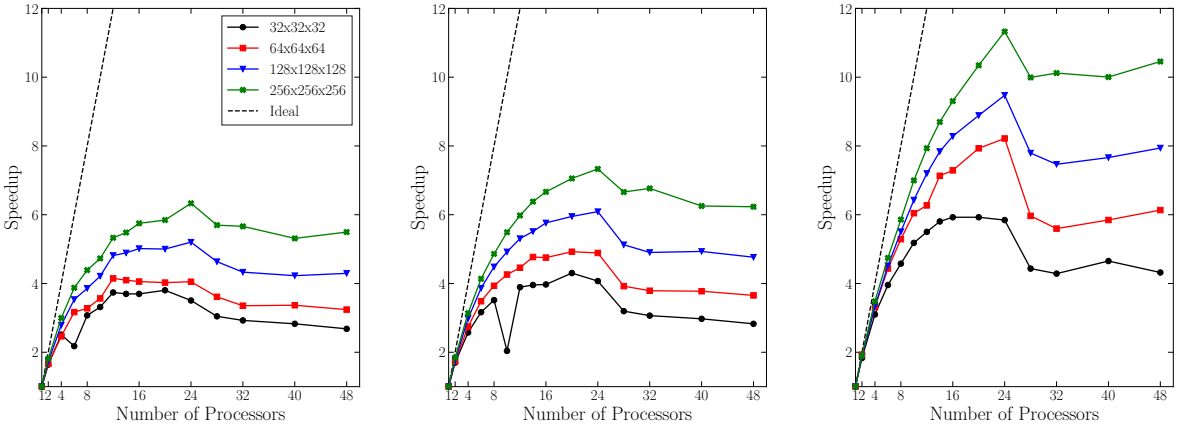


Figure 3.25: Strong scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=close`.

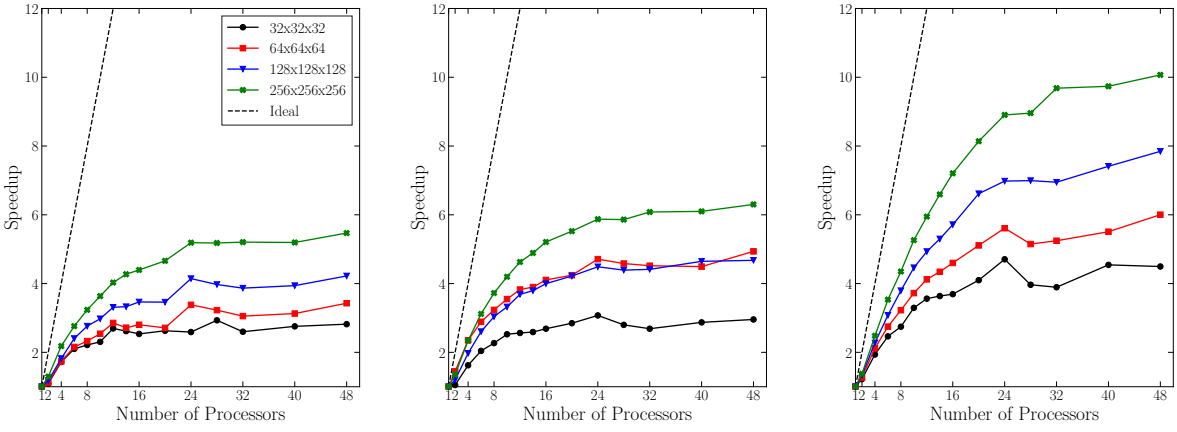
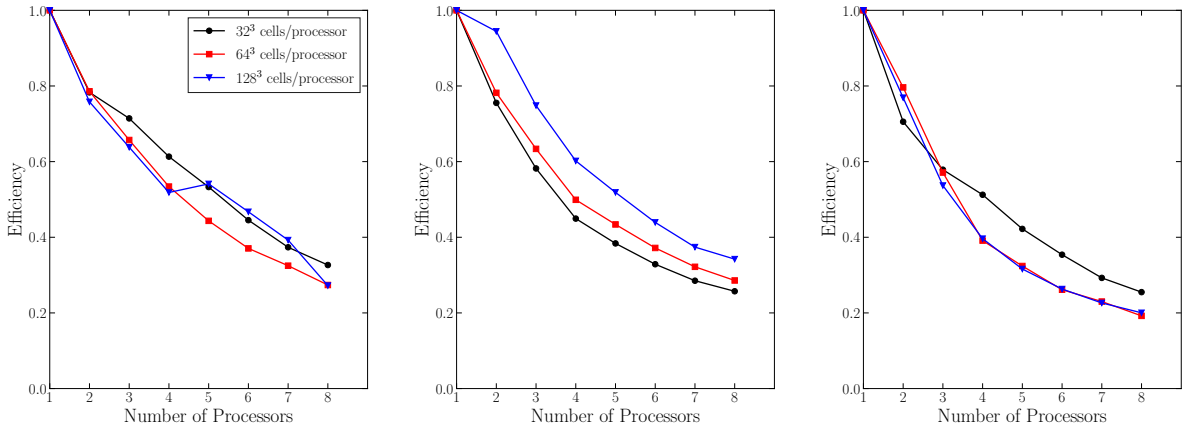
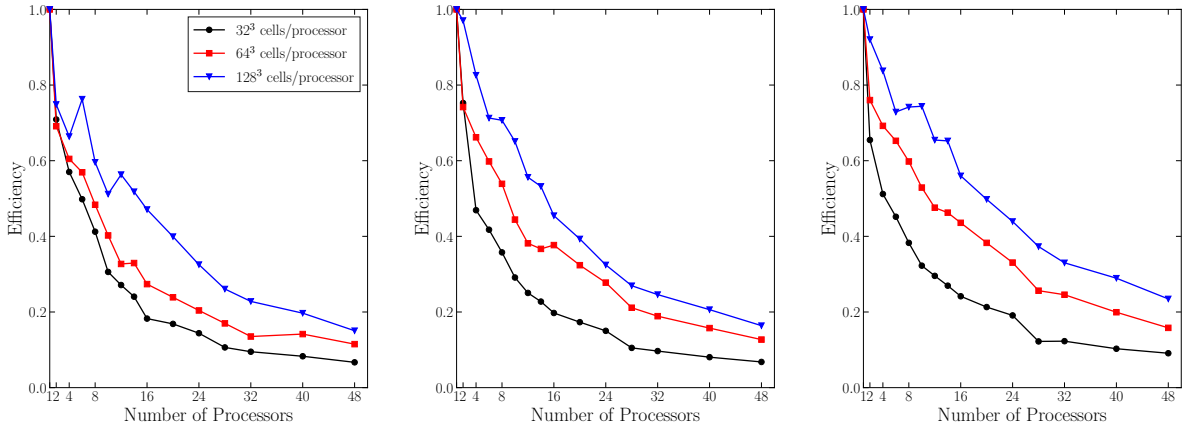


Figure 3.26: Strong scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=spread`.



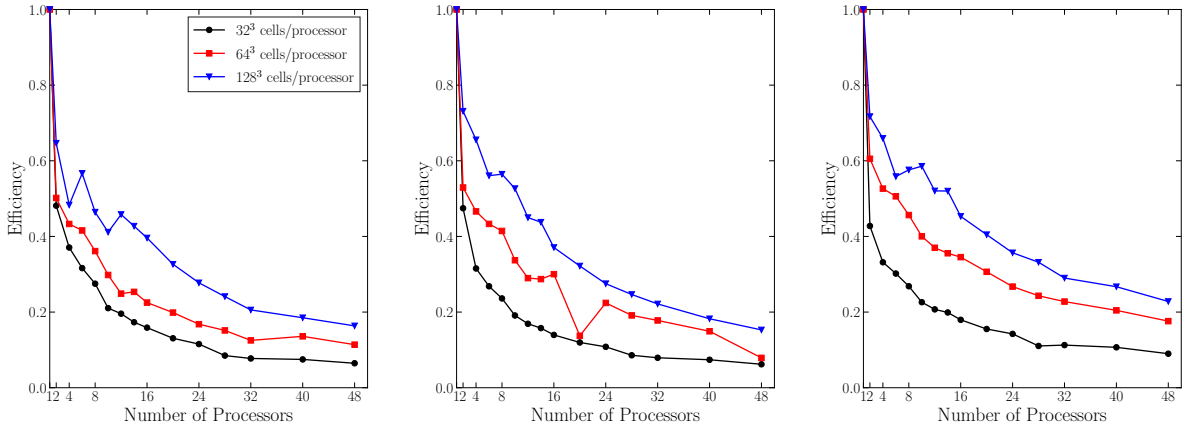
(a) **(b)** **(c)**

Figure 3.27: Weak scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 1 (Table 3.4).



(a) **(b)** **(c)**

Figure 3.28: Weak scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=close`.



(a) **(b)** **(c)**

Figure 3.29: Weak scaling tests for the 3D lid driven cavity parallelised with planes (a), lines (b), and points (c) for System 2 (Table 3.4) with `PROC_BIND=spread`.

decreases as more processors are used. As the number of processors is increased (and hence the problem size in the same proportion), so does the amount of data required from memory. This means the fixed memory bandwidth has to be shared among a greater number of processors, leading to a decrease in efficiency. For almost all cases, efficiency increases with greater number of cells per processor. With a greater number of cells per processor there is more work for each processor to do, this decreases the relative cost of thread barriers (which occur at the end of every sweep), memory latency and bandwidth, and any effects of false sharing which can occur more frequently for smaller problem sizes since the relative size of boundaries for each processor is larger. There is an exception however observed on System 1 for plane (3.27a) and point (Figure 3.27c) colour orderings. Reasons for this are not exactly clear.

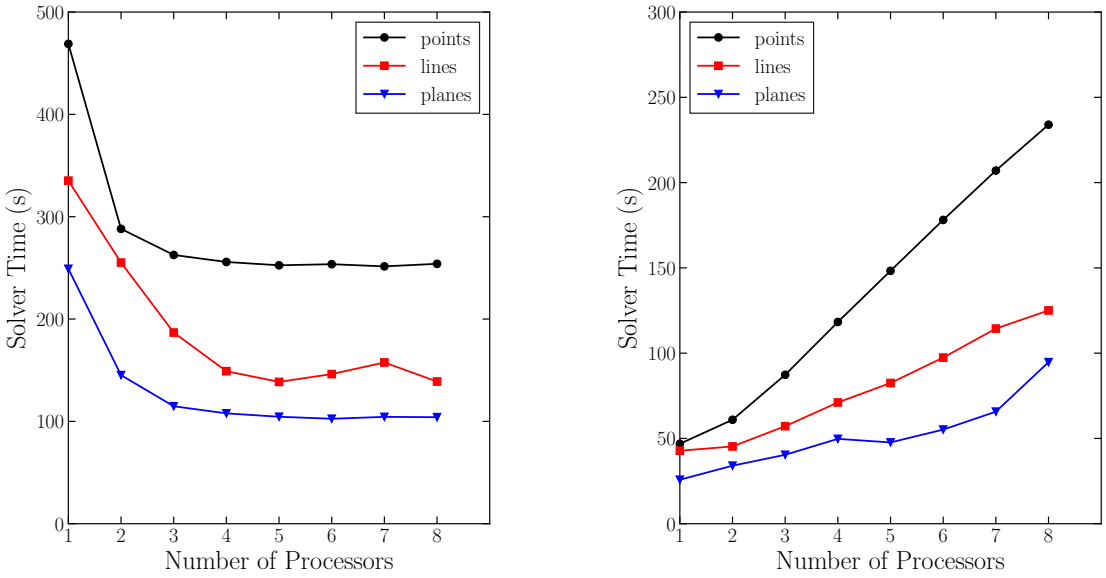
Overall, there is little qualitative difference between each colour ordering, except that the point ordering gives slightly better efficiency, followed by line ordering. As with the strong scaling, close core affinity (Figure 3.28) resulted in better efficiency than spread affinity (Figure 3.29) when less than 24 processors are used. Beyond this, the efficiency between the two is essentially identical.

The tests above have shown that point wise colour ordering of the domain tends to give the best parallel efficiency and scaling. However, solver runtime is the main factor with which we are concerned with. Figs. 3.30, 3.31, and 3.32 show total solver run times for the strong scaling tests with 256^3 cells and the weak scaling tests for 128^3 cells/processor for each of the colour orderings. In all cases, the plane colour ordering gives significantly better performance, followed by the line, and finally the point ordering.

These observations are in line with the reasoning given in Section 3.4, that is, line, and plane ordering results in better data locality for a given processor, and hence is more cache efficient. Point ordering makes less reuse of data already in cache, resulting in more cache misses, and hence more references to main memory, which is expensive. These more frequent references to main memory are more of a fixed cost on performance, which is why parallel efficiency does not suffer much when compared to plane and line ordering.

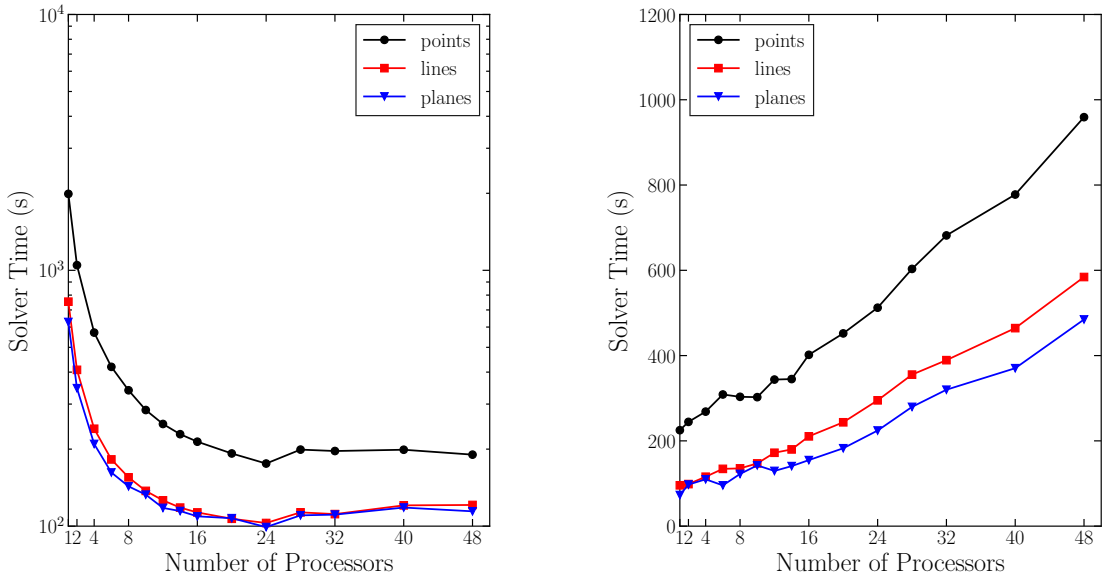
Overall, the results above indicate that parallel scaling for the present method is far from ideal. This is expected for a shared memory implementation of stencil based methods. To update a single molecule, a large number of finite volume stencil coefficients are read from memory and are only used once for simple addition and multiplication operations in the stencil evaluation. The relative cost of reading in the coefficients from memory is significantly higher than the cost of the computation that is done with these coefficients. Hence, as the above discussion has indicated, the application is memory bandwidth limited due to the fundamental nature of the

algorithm. For this reason other parallel approaches such distributed memory parallelism and the use GPUs are more likely to be favoured.



(a) **(b)**

Figure 3.30: Raw solver times for strong scaling test with 256^3 (a) and weak scaling tests with 128^3 cells/processor (b) for System 1 (Table 3.4).



(a) **(b)**

Figure 3.31: Raw solver times for strong scaling test with 256^3 (a) and weak scaling tests with 128^3 cells/processor (b) for System 2 (Table 3.4) with `PROC_BIND=close`.

This is not the only factor affecting solver times however. Again, as was mentioned in Section 3.4, the different update orderings can result in different convergence rates. Table 3.5

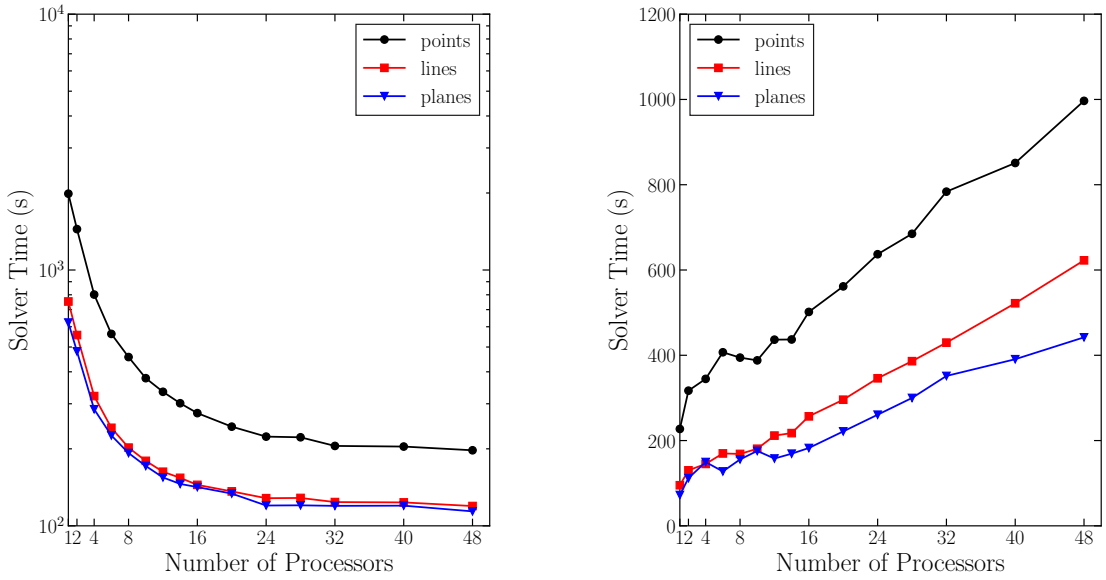


Figure 3.32: Raw solver times for strong scaling test with 256^3 (a) and weak scaling tests with 128^3 cells/processor (b) for System 2 (Table 3.4) with `PROC_BIND=spread`.

shows the number of solution update sweeps (forward and backward), the time spent sweeping in the solver, and the average number of sweeps per second for the 256^3 case on a single processor for each of the colour orderings. A single processor is used to eliminate the effect of any parallel speedup, so the difference in timing is purely due to convergence rate and single threaded performance. The problem was solved to the residual of 10^{-10}. Table 3.5 shows that the plane ordering solved the problem in the least number of sweeps, followed by the line ordering and finally the point ordering. Moreover, plane ordering performs the greatest number of sweeps per second, followed by lines, then planes. So while point sweeping may give better parallel efficiency (only by a small amount), significant speedup can be had by using a more optimal colour ordering due to the better convergence rate and data locality.

Table 3.5: Timing results for solution update sweeps performed to solve the 3D lid driven cavity problem on a 256^3 grid with System 1 (Table 3.4) using a single processor for each colour ordering in Figure 3.6.

	Colour ordering		
	Planes	**Lines**	**Points**
Number of sweeps	25164	29562	33264
Time spent sweeping (s)	158.82	242.13	356.31
Average sweeps/s	158.44	122.09	93.36

The timing results in Figs. 3.30, 3.31, and 3.32 reveal that System 1 significantly outperforms System 2 when the same number of processors are used. Even when significantly more processors on System 2 are used, System 1 still solves the problem in a smaller amount of time. This is due to the nature of the problem and the design focus of the CPUs used in the two systems in Table 3.4. The i9-11900F in System 1 (released in 2021) was designed for consumer desktop applications which benefit from single-threaded performance, while the Xeon 8274 (released in 2019) is geared towards multi-threaded server workloads, hence the significantly higher core count. Despite System 2 offering greater total memory bandwidth, System 1 achieves better performance due to its newer microarchitecture, lower memory latency, and higher per-core bandwidth and last level cache.

The results in this section underscore a number of implementation considerations that have significant effects on performance, especially when executing the code on multiple cores on a shared memory machine. It should be made clear that all results presented here are implementation and architecture dependent, and no claims are made that the methods described here are optimal for the particular computer architectures which the tests were conducted on. Results of these tests are likely to be different for different solver algorithms (e.g. segregated solvers) or for different parallel paradigms such as Graphics Processing Units (GPUs) or distributed memory computer clusters. For the rest of the results in this thesis, the plane colour ordering will be used (Figure 3.6c) as it gives the best performance with regards to solver time.

3.8 Summary

The coupled smoother introduced in this chapter forms the foundation of the complete solver application being developed in this work, and is one of the main novel contributions of this thesis. Everything that follows in this thesis will be built upon the basic technique described and developed here. The method takes the efficiency and algorithmic simplicity of the matrix-free Vanka smoother and applies this to a collocated grid. Being on a collocated grid means that any geometric based computations are greatly simplified, which generally leads to a more efficient implementation. This is particularly true when used with FAS multigrid, as was done in this chapter, and with the Immersed Boundary Method, which is introduced in the following chapter. The code was verified using order of accuracy tests to demonstrate the second order accuracy of the discretisation. Validation and benchmarking of the code in serial was done against the coupled solver in ANSYS Fluent, and the SIMPLE scheme in OpenFOAM for the 3D laminar lid driven cavity and 3D laminar backwards facing step. Excellent agreement was found with these

solvers. When only used on a single grid, the solver was found to outperform the other methods in almost all cases, with up to an order of magnitude speedup. With FAS multigrid up to two orders of magnitude speedup was observed against the other methods. While it was found that in the single grid case the choice of sweep direction can influence convergence rate, with the FAS multigrid scheme, the difference between sweeping directions was minimal, however for the backwards facing step, some sweeping directions were found to be unstable. This indicates that for some flows, the choice of sweeping direction is an important consideration for obtaining stable solutions. It was shown that the method can be parallelised using a three colour ordering. Since the method was to be implemented using shared memory parallel with OpenMP, partitioning the different colours in a pointwise, linewise, and planewise manner was considered with the aim of achieving a more efficient memory access pattern. Testing on two different computer systems showed that while best parallel efficiency was obtained with a pointwise ordering, the performance in terms of runtime was significantly better using planewise ordering, with linewise ordering sitting in the middle.

CHAPTER 4

Immersed Boundary Method

4.1 Introduction

For many applications in Computational Fluid Dynamics (CFD), especially those which involve complex geometries, mesh generation can be very time consuming. Furthermore, robust meshing algorithms which can deal with arbitrary geometries are complex and require significant programming effort to implement within an existing code. Immersed Boundary methods are an approach that has emerged to alleviate these issues. The mesh is not required to conform to the boundaries, and the effect of a solid wall is accounted for through modification of the discretisation at the boundary. This means a Cartesian grid can be used which results in easy mesh generation, savings in memory and computational time, and simpler parallelisation compared to body fitted grids.

The immersed boundary method has developed greatly in terms of technique and applications since its initial introduction by Peskin [107]. Complete overviews are given by Mittal and Iaccarino [108] and more recently Verzicco [109]. There are a few variants of the immersed boundary method. This work focuses on ghost cell methods which are appealing because of their simplicity, especially for the finite volume method. Ghost cell immersed boundary methods, which fall under the category of direct forcing methods, have been used successfully in a range of different compressible [110] and incompressible [111, 112] flows, including those with moving boundaries [113, 114].

In these methods, the momentum forcing is applied through the setting of ghost cells which are reconstructed from the fluid domain using the solid surface in order to enforce a no slip condition at the boundary location. Typically, this reconstruction is done in the direction normal to the immersed boundary surface near the ghost cell. Despite improvements and encouraging results in recent years, a number of issues still remain. One of the primary difficulties is mass

conservation.

This chapter focuses on the implementation of immersed boundary methods in coupled solvers for incompressible flows on collocated grids. To the authors knowledge, there has been no application of an immersed boundary method with a coupled solver, on collocated or staggered grids. Since coupled solvers treat both velocity and pressure terms implicitly the continuity and momentum equations by design, many standard techniques for enforcing global mass continuity [115–117] are difficult and impractical to implement.

The advantages of collocated grids have already been discussed in previous chapters of this thesis. As will become evident in Section 4.3, reconstruction operators are identical for all variables, which simplifies the implementation significantly, especially when coupled with a geometric multigrid method. When a collocated grid is used, the treatment of MWI at the immersed boundary requires some attention since it no longer appears as a source term in the continuity equation, but is included implicitly. A standard ghost cell method cannot correctly account for the MWI at the boundaries. These issues are discussed in more detail in Section 4.2.

A mass conservative immersed boundary method has been developed here which is compatible with coupled incompressible solvers while still having a simple implementation. The method is applied to the scheme developed in Chapter 3. In the multigrid scheme the immersed boundary is resolved on all grid levels. To the authors knowledge, the only other application of FAS multigrid with an immersed boundary method was by Ryan et al. [90], who used the SIMPLE scheme as the smoother – not a coupled scheme. The issues with current methods in relation to coupled solvers are discussed in Section 4.2, and the present method is detailed in Section 4.3. The implementation of the solver is discussed in Section 4.4. Verification studies are carried out using the test case of a steady laminar flow of a 3D sphere in a lid driven cavity, while validation tests are performed by considering the steady laminar flow around a 3D sphere and 45° angled cube in a channel. These results are given in Section 4.5.

4.2 Challenges for Coupled Solvers

By nature, fully coupled solvers treat all terms – pressure and velocity – as implicit throughout the solution procedure. Practically speaking, when implementing a ghost cell immersed boundary method, this introduces two difficulties not seen in segregated solvers that have not been addressed in the current literature. This section introduces these two issues.

The first issue is with regards to global mass conservation. The first step in a ghost cell immersed boundary method is to tag all cells which lie outside the fluid domain but border a

fluid cell as ghost cells. All other cells outside the fluid domain are considered to be “solid cells". This is illustrated in Figure 4.1a. Regarding the solid cells, these can be treated in various ways, including solving for a fictitious flow. It has been demonstrated through numerical experiments that different treatments typically have little to no effect on the flow in the fluid region [112, 118–120]. The setting of ghost cells essentially creates a staircase domain boundary comprised of the cell faces which lie on the border between the ghost cells and fluid cells. This boundary is shown by the bold lines in Figure 4.1a and will be referred to as the Approximated Boundary (AB). Ghost cell values are then set by using some type of reconstruction from the fluid region. The most common approach is to first project an image point for each ghost cell in the direction normal to the immersed boundary surface (denoted by point I in Figure 4.1b). The value of the fluid variable at the image point location is then obtained through interpolation, after which an extrapolation is performed along the normal direction using the immersed boundary intercept (denoted by point C in Figure 4.1b) onto the ghost cell [111, 121, 122].

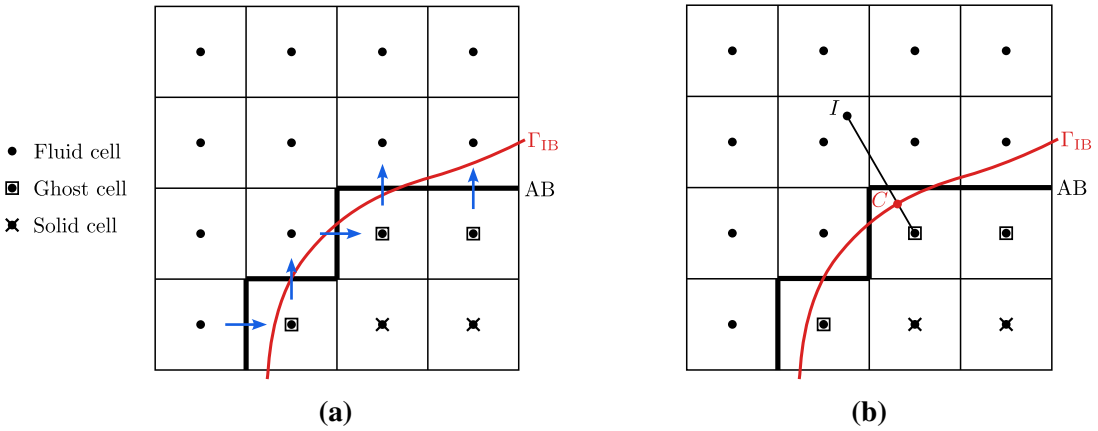


Figure 4.1: Setting of ghost cells within the immersed boundary applies a mass flux on the cells which border the ghost cells (a), this is generally done by extrapolating in the direction normal to the immersed boundary for each ghost cell (b).

The imposition of ghost cells in this manner (or any general reconstruction) implicitly results in mass fluxes appearing on the cell faces of the AB. These mass fluxes however are not derived from a continuity equation, and are set independently from each other. As a result, there is no guarantee that the mass fluxes satisfy global mass conservation. It is important to note that local continuity will still be satisfied for all fluid cells. However, the fluid that leaves the fluid domain through this boundary does not necessarily re-enter the domain, resulting in a non-physical net mass flux across the solid body.

One way to think about this problem is that a continuity equation is not being solved in the ghost cells, when in fact there should be. Many of the standard approaches that have become commonplace in immersed boundary methods are based on this idea. Kim et al. [115] added

mass sources to the continuity equation on a staggered grid. These sources were determined from a continuity equation written just for the fluid region of a fluid cell that was intersected by the solid boundary. While this approach could be applied to a collocated grid, it is not as straightforward since continuity cells are shared by momentum cells, which may be "inactive" if the cell centre is within the solid. Another approach is to modify the face fluxes in the continuity equation (which is done through the Poisson equation for pressure) so that there is no net mass flux across the immersed boundary. This is done on a staggered grid by Mark and Wachem [116] and Shinn et al. [117] and on a collocated grid by Yan et al. [123]. This means that mass is conserved locally on every control volume in the domain, including solid and ghost cells, which results in mass being conserved globally.

These techniques are relatively simple to use in a segregated solver where these face fluxes appear as source terms in the Poisson pressure equation. These source terms can be set explicitly once and for all for each outer iteration before the Poisson equation is solved – they do not change during the linear solver's iterations. Unfortunately, applying a similar approach in a coupled solver where these fluxes become implicit introduces significant complexity into the method.

Moreover, these methods do not satisfy the pressure decoupling constraint given by Kang et al. [124] which requires that solutions in the fluid domain are decoupled from solutions anywhere else in the domain, i.e. the solid domain here. Without this, the pressure at the boundary will not satisfy the correct discrete equations. In the case of the above mentioned methods, the constraint is violated because mass conservation for the solid and fluid cells become coupled together. This idea is also addressed by Yildiran et al. [125] in the context of fractional step methods. While their approach may be applied to coupled methods, it is not as straightforward since an explicit Poisson equation for pressure or pressure correction is not formed, and modification of the pressure gradients in the momentum equations would be required instead. Finally, it should be noted, that it is possible to violate the pressure decoupling constraint, but still conserve mass, as is the case with the above mentioned methods.

Cut-cell type methods avoid this issue by reshaping the control volumes near the immersed boundary so that mass is conserved and the pressure decoupling constraint is satisfied. However, cut cell methods are quite complex to implement [126, 127]. Madabushi and Ghosh [128] use a cut-cell inspired approach in an immersed boundary method to achieve mass conservation during the reconstruction stage by accounting for the volume fraction of the cell that has been intersected. However, like cut-cell methods they suffer from being more complicated than other immersed boundary methods, especially in 3D.

The second issue is exclusive to collocated grids and comes from the application of MWI.

MWI was introduced in Chapter 2.5 and is formulated as a correction to the advecting face velocity in the continuity equation which takes the form

$$\boldsymbol{u}_f = \overline{\boldsymbol{u}}_f - d_f \left(\nabla p_f - \overline{\nabla p}_f \right), \tag{4.1}$$

where the subscript f indicates the quantity is at the cell face, the overbar indicates a weighted average of the cell centre values that straddle the face, and d_f is a momentum weighting. At domain boundaries, the second term on the right hand side of Eq. 4.1, which we will call the MWI correction, is set to be zero [129]. This includes the cell faces on the AB. The discretised MWI correction appears in the pressure terms in the continuity stencil (Eq. 2.39) and a contribution to these terms comes from each of the cell faces.

In segregated methods, MWI is applied to the face fluxes which appear in the Poisson equation, which as discussed before are constant source terms at each outer iteration. This makes the setting of the correction to zero relatively simple: A small number of source terms corresponding to the immersed boundary are manually set to zero once before the linear solver. In a coupled solver this is not an option; the pressure is treated implicitly in the continuity equation and is part of the stencil. Therefore, the correction must be set to zero through the use of ghost cells.

To do this, there are two requirements. First, due to the sparse gradient in Eq. 4.1 the continuity pressure terms create a wide stencil comprising of the cells at the NN, EE, SS, WW, TT, and BB locations. This means a second layer of ghost cells is required. This can be an issue in geometries that are only one cell thick. The second – and more critical – requirement is that ghost cells need to have multiple values depending on which fluid cell requires them. In particular, looking to Figure 4.1a, it is possible for a ghost cell to have more than one face that is part of the AB. In order to set the MWI correction to zero on each of these faces independently, the ghost cells need to have different values for each face that borders them. This cannot be done with a standard ghost cell approach.

One final issue to discuss is the treatment of corners. This is not specific to coupled solvers or collocated grids, but arises due to the reconstruction being done in the normal direction. At the location of a corner, the normal direction is not well defined, so it is not clear how reconstruction should be done. Other degenerate cases exist such as convex regions where there are multiple valid locations for the point C (Figure 4.1b). Authors have proposed approaches to dealing with such cases [121] when performing reconstruction in the normal direction, however, they are typically quite complex and work on a case by case basis. Ideally, an immersed boundary method should be used that does not require special treatment of such cases.

4.3 The Immersed Boundary Technique

In standard ghost cell methods, the fundamental quantity being set by the reconstruction is the cell centre value of the ghost cell. Face fluxes on the AB arise as a by-product. If multiple faces on the AB share a particular ghost cell, they will all be set by this reconstruction, and there is no way to set them individually. Instead, in the present method, face values on the AB (for both velocity and pressure) are set directly in the same way that boundary conditions on domain boundaries are applied. This is achieved by using a directional ghost cell method. Since it is the face fluxes being set on the AB, the issue of global mass continuity is dealt with by adding a correction to these face fluxes to give a mass conservative reconstruction. The resulting face fluxes are used to determine ghost cell values which are simple to implement within an existing solver. The ghost cell concept is only used as a convenient interpretation which simplifies implementation in a finite volume code. Since it is face fluxes being set – of which there may be more than one for a given ghost cell – each ghost cell is required to have a unique value for each face that borders it. This is dealt with by adding the effect of the ghost cells as source terms to cells in the fluid region instead of setting them in the solution field itself. As a result, the solution in the fluid domain is completely decoupled from the solid domain, and the pressure decoupling constraint is fully satisfied. It should be made clear, while ghost cells are not being set explicitly, using the idea of ghost cells allows for a simple formulation and implementation to an existing solver. The method is described below for a general non-uniform Cartesian mesh.

4.3.1 Reconstruction

Figure 4.2 illustrates the directional method in 2D, the extension to 3D is natural. A reconstruction is done for each coordinate direction independently along the cell centres. This means that each face on the AB is set using its own unique reconstruction. This is unlike the ghost cell method discussed in Section 4.2 where the setting of a single ghost cell will set the field variables on all faces which border it. In order to frame this directional method as a ghost cell method, it is required that each ghost cell can take multiple values – one for each face of the cell that lies on the AB. The detailed approach for setting multiple ghost cells in the discrete equations is described later in this section. The procedure will be explained in the horizontal direction, with the application to other directions/faces being the same. This directional reconstruction means corners and other degenerate cases on a geometry are treated naturally since the intersection points X and Y are always well defined assuming the mesh has been refined enough.

Since a correction is to be applied to the resulting mass fluxes on the AB, reconstruction

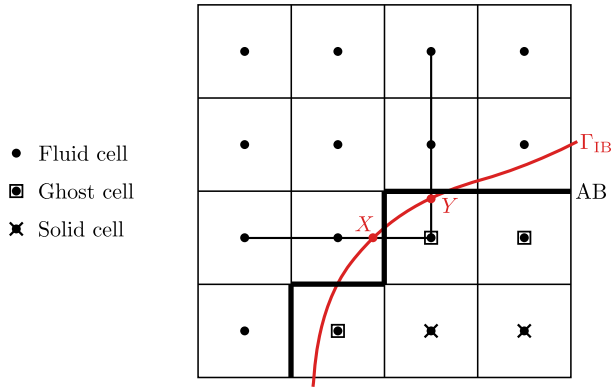


Figure 4.2: Reconstruction for a ghost cell is done in each coordinate direction independently, using the intercept with the immersed boundary in each direction for extrapolation.

of the flow variables is applied at the faces on the AB, not directly to the ghost cell locations. Looking to Figure 4.3, we consider the case where we are reconstructing field variables onto the cell face e, which lies on the AB. The goal is to use the fluid points W and P, and the immersed boundary intercept point X (for which the field variable at this location is assumed to be known) to obtain a value for the field variable on the cell face e. As shown in Figure 4.3, two cases must be considered: one where point X lies within the ghost cell (Figure 4.3a), and one where X lies within the fluid cell (Figure 4.3b). The case in Figure 4.3a represents an interpolation onto the cell face, while Figure 4.3b is an extrapolation onto the cell face.

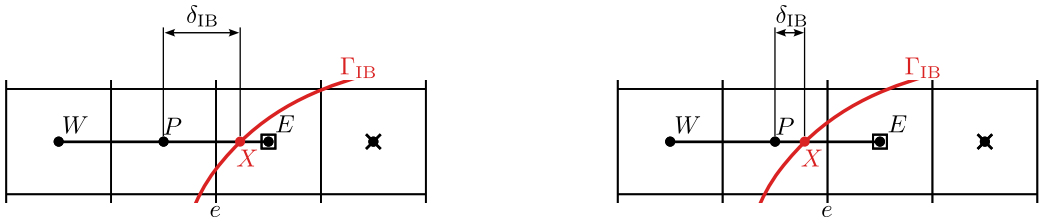


Figure 4.3: Two cases need to be considered for the directional method depending on relative location of the immersed boundary to the face that borders the ghost cell and fluid cell.

For the case in Figure 4.3a, the field variable at cell face e, denoted by ϕ_e, is obtained by performing a linear interpolation between the field variable at cell P and location X, which is given by

$$\phi_e = \frac{2\delta_{IB} - \Delta x_P}{2\delta_{IB}}\phi_P + \frac{\Delta x_P}{2\delta_{IB}}\phi_X, \tag{4.2}$$

where Δx_P is the width of the cell at location P. This formula can be shown to have order of accuracy $O(\Delta x\, \delta_{IB})$. Now if the same two points were to be used to perform the extrapolation for the case in Figure 4.3b, Eq. 4.2 could again be used. However, in this case the point X can be arbitrarily close to the cell centre at P, meaning δ_{IB} can approach zero in certain situations.

Since δ_{IB} is on the denominator of Eq. 4.2, ϕ_e can become non-physically large, and result in instabilities in the solution. This is dealt with by switching to using the fluid value at cell location W instead of P to perform the extrapolation for this case:

$$\phi_e = \frac{\Delta x_P - 2\delta_{\text{IB}}}{\Delta x_P + \Delta x_W + 2\delta_{\text{IB}}}\phi_W + \frac{\Delta x_W + 2\Delta x_P}{\Delta x_P + \Delta x_W + 2\delta_{\text{IB}}}\phi_X. \tag{4.3}$$

Although this uses points further away from the immersed boundary, and is potentially less accurate, it still has the same the formal order of accuracy as Eq. 4.2. An alternative approach used by Chi et al. [130] is to switch from Eq. 4.2 to Eq. 4.3 based on a condition $\frac{\delta_{\text{ib}}}{\Delta x_P/2} < r$, where r is some predetermined parameter. However, in the authors experience, the selection of the parameter r was problem dependent and often had to be chosen close to 1 to allow stable solutions, which is equivalent to the approach described above. The potential improvements in accuracy were not deemed to be worth the increase in complexity and decreased robustness of the method.

4.3.2 Mass Flux Correction

In the case of the Navier-Stokes equations, the variables u_e, v_e, w_e, and p_e are reconstructed onto the cell face e. The normal component of the velocity produces the mass flux through the cell face, and for the case in Figure 4.3, this is u_e. As described in Section 4.2, the face fluxes reconstructed using Eqs. 4.2 and 4.3 will not satisfy global mass conservation. The next step is to correct this face flux to enforce global mass conservation. Here, the correction approach of Kang et al. [124] is used, which has also been used by Khosronejad et al. [131] for deforming boundaries. It should be clarified that the definition of the AB in this work is different to that used by Kang et al. They define the fluid region to only be made up of cells that are not intersected at all by the immersed boundary. This approach was not used here as it can result in some degenerate edge cases when used with a directional method that are difficult to deal with. Furthermore, cell faces on the AB will in general be further away from the immersed boundary, resulting in a less accurate reconstruction.

Again without loss of generality, consider just the cell face e. In this case the velocity flux component is u_e. For faces in the y and z normal directions, the relevant velocity flux components are given by v and w respectively. We will denote the uncorrected face velocity that comes directly from the above procedure (and does not satisfy mass conservation) by u_e^*. The corrected flux will then take the form

$$u_e = u_e^* + \sum_{k=1}^{3} \delta a_{k,e} x_{k,e}, \tag{4.4}$$

where $\delta a_{k,e}$ are weighting coefficients for the correction on cell face e and $x_{1,e}$, $x_{2,e}$, and $x_{3,e}$ are the Cartesian coordinates of the cell face centre of e in a coordinate system that is centred on the immersed boundary surface – point X on Figure 4.3 in this case. In the directional method presented here, only one of these coordinates is non-zero because both the cell face centre and the intersection point lie along a coordinate line. The following procedure is completely independent of how u_e^* is obtained; any other reconstruction can be used. The velocity in Eq. 4.4 is subject to the constraint that the total mass flux through the AB is zero:

$$\sum_{f \in \text{AB}} (\boldsymbol{u}_f \cdot \hat{\boldsymbol{n}})_f A_f = 0, \tag{4.5}$$

where $\boldsymbol{u}_f$ is the velocity vector on the cell face. One can think of Eq. 4.5 as an additional constraint to the system that represents global mass conservation, in addition to the momentum and local continuity equations. We would also like the correction applied to be minimal so as to respect the reconstruction. The objective function to be minimised is

$$J = \sum_{f \in \text{AB}} \left(\sum_{k=1}^{3} \left| \delta a_{k,f} \right|^2 \right). \tag{4.6}$$

The minimisation problem can be solved through the introduction of a Lagrange multiplier. Letting $Q_{\text{err}} = -\sum_{f \in \text{AB}} (\boldsymbol{u}_f^* \cdot \hat{\boldsymbol{n}})_f A_f$, the resulting expression for the correction is given by [124]

$$u_e = u_e^* + \frac{\omega_e A_e \sum_{k=1}^{3} x_{k,e}^2}{\sum_{f \in \text{AB}} \left(\omega_f A_f^2 \sum_{k=1}^{3} x_{k,f}^2 \right)} Q_{\text{err}}. \tag{4.7}$$

where ω_f is the weighting factor of face f. The quantity $\sum_{k=1}^{3} x_{k,f}^2$ is the squared distance of the cell face on the AB to the immersed boundary along the face normal direction. The correction in Eq. 4.7 is essentially a weighted redistribution of the flux error (given by Q_{err}). This redistribution is weighted by the cell face area A_e, and the squared distance of the cell face to the immersed boundary, $\sum_{k=1}^{3} x_{k,e}^2$. The coefficient in front of Q_{err} in Eq. 4.7 is purely geometrical. If the immersed boundary is not moving, then it only needs to calculated once for each face.

As noted by Kang et al. [124], since Eq. 4.7 is a global redistribution of the flux error, it may not be appropriate for situations where mass flux error is known to be localised due to large velocity gradients within a certain region. In such cases, one can divide the AB and apply the correction to each region separately. This requires some knowledge of the solution and the flow field prior to the computation, and can be quite a manual and complex process. An alterative is to set the weighting factor ω_f to be dependent on the local velocity gradients. In this work we take $\omega_f = 1$ for simplicity. For the same reasons, if one is solving for the flow over multiple

non-connected bodies, a separate correction should be applied to each body individually based on the mass flux error for that particular body.

4.3.3 Ghost Cell Extrapolation

Once the fluxes have been corrected, the face quantities are then extrapolated onto the ghost cell locations. It is important that the same interpolation scheme used in the continuity equation be used to extrapolate to the ghost cells. In the case of most finite volume schemes, this is linear interpolation. While it is possible in principle to use the reconstruction procedure used to obtain ϕ_f – or any other reconstruction – this would result in an inconsistency with the continuity equation and the above correction would no longer be guaranteed to conserve mass globally. The weighted linear interpolation used to obtain cell face quantities from cell centres in the present scheme is given by

$$\phi_e = (1 - \lambda_e)\phi_P + \lambda_e \phi_E. \tag{4.8}$$

Where λ_e is the linear interpolation factor (See Eq. 2.16 in Chapter 2). The extrapolation onto the ghost cell, which is the cell at location E in Figure 4.3, is obtained by re-arrangement of this equation, resulting in

$$\phi_E = \frac{1}{\lambda_e}\phi_e - \frac{1 - \lambda_e}{\lambda_e}\phi_P. \tag{4.9}$$

4.3.4 MWI Ghost Cell Determination

In Section 4.2, it was mentioned that a second layer of ghost cells is required to correctly set the MWI correction in Eq. 4.1. Setting the correction to zero requires

$$\nabla p_f - \overline{\nabla p}_f = 0. \tag{4.10}$$

The first gradient term is the compact gradient at the cell face, the second term is the weighted average of the sparse gradients at the cell centres straddling the face. Considering the cell face e to be on the AB, on a rectilinear grid this can be written as

$$\left.\frac{\partial p}{\partial x}\right|_e - \left((1 - \lambda_e) \left.\frac{\partial p}{\partial x}\right|_P + \lambda_e \left.\frac{\partial p}{\partial x}\right|_E \right) = 0. \tag{4.11}$$

The compact gradient is

$$\left.\frac{\partial p}{\partial x}\right|_e = \frac{p_E - p_P}{x_E - x_P}, \tag{4.12}$$

while the sparse gradients on a non-uniform grid are:

$$\left.\frac{\partial p}{\partial x}\right|_P = \frac{\lambda_e p_E + (1-\lambda_e-\lambda_w)p_P - (1-\lambda_w)p_W}{\Delta x_P}, \tag{4.13}$$

$$\left.\frac{\partial p}{\partial x}\right|_E = \frac{\lambda_{ee} p_{EE} + (1-\lambda_{ee}-\lambda_e)p_E - (1-\lambda_e)p_P}{\Delta x_E}. \tag{4.14}$$

The MWI at the face e contains two cells that fall into the solid region: p_E and p_{EE}. The cell value p_E comes from the pressure boundary condition at the immersed boundary (Section 4.3.6) and the reconstruction/ghost cell extrapolation. All that is left is to set the ghost value p_{EE}, which is done directly from the condition in Eq. 4.11. Re-arrangement of Eq. 4.11 yields

$$\begin{aligned} p_{EE} = &\left[\frac{\Delta x_E}{\lambda_e\lambda_{ee}(x_E-x_P)} - \frac{\Delta x_E}{\Delta x_P}\frac{1-\lambda_e}{\lambda_{ee}} - \frac{1-\lambda_{ee}-\lambda_e}{\lambda_{ee}}\right] p_E \\ &+ \left[-\frac{\Delta x_E}{\lambda_e\lambda_{ee}(x_E-x_P)} - \frac{\Delta x_E}{\Delta x_P}\frac{(1-\lambda_e)(1-\lambda_e-\lambda_w)}{\lambda_e\lambda_{ee}} + \frac{1-\lambda_e}{\lambda_{ee}}\right] p_P \\ &+ \left[\frac{\Delta x_E}{\Delta x_P}\frac{(1-\lambda_e)(1-\lambda_w)}{\lambda_e\lambda_{ee}}\right] p_W. \end{aligned} \tag{4.15}$$

The pressure terms p_P and p_W are within the fluid domain, and will make up part of the stencil, while p_E is a ghost cell whose value is known, and its contribution is added as a source term, which will be detailed in the following subsection. Although large and complex, the coefficients for the pressure terms in Eq. 4.15 are purely geometrical, and do not change if the solid body is fixed in space, and hence only need to be calculated once and stored.

4.3.5 Source Term Construction

One of the salient features of the directional method is the fact that each reconstruction direction will result in a different ghost cell value. This cannot be achieved within the finite volume stencil in the typical manner of setting the ghost cell value in the solution field itself. Instead, the ghost cell values calculated using the above methods for each direction are stored separately to the solution field. The discrete transport equations for fluid cells which have a ghost or solid cell part of their stencil will receive a source term. This will be demonstrated by considering a simple 1D stencil located at cell P, with the extension to the complete discrete equations (Eqs. 2.36 - 2.39) being the same:

$$a_W\phi_W + a_P\phi_P + a_E\phi_E = B_P. \tag{4.16}$$

We will call the stored value for the ghost cell at location E, obtained from the x direction $\hat{\phi}_E^x$. The first step is to fix all solid and ghost cell values in the solution field to zero. This automatically eliminates terms corresponding to the ghost cell (and solid cells when a wide

stencil is used, as is the case with MWI) when the stencil is evaluated inside the solver. Then, using the corresponding stencil coefficient which is already known, the ghost cell contribution is added to the right hand side

$$a_W \phi_W + a_P \phi_P = B_P - a_E \hat{\phi}_E^x. \tag{4.17}$$

Similarly, if there was a ghost cell at location S (the cell in the negative y direction relative to the P cell), then a source term corresponding to the ghost cell $\hat{\phi}_S^y$ would be added. It should be noted that due to the MWI, which results in a second ghost cell $\hat{p}_{EE}^x$, a source term will also have to be included for cells that are one cell away from the immersed boundary (cell W in Figure 4.3 for example).

4.3.6 Boundary Conditions

The reconstruction described in Section 4.3.1 assumed that the field variable at the immersed boundary intersection ϕ_X was already known. This value is in fact obtained from the boundary condition at the immersed boundary. For Dirichlet boundary conditions, this can be applied directly. In this work, only non-moving solid boundaries are considered, so for the velocity, we have that, $u_X = v_X = w_X = 0$.

The pressure on the other hand is not known, and is obtained through linear extrapolation from the fluid. One approach that is convenient is to simply extrapolate the pressure onto the immersed boundary and re-use the reconstruction above to obtain a ghost cell value. However, this means that the pressure will be reconstructed using a stencil that depends on the velocity stability condition from Section 4.3.1. This can result in an unnecessarily less accurate extrapolation for the pressure near the boundary. Instead, the pressure is extrapolated using a linear extrapolation in each direction onto the immersed boundary surface. For the situation in Figure 4.3, this is given by

$$p_X = \frac{x_P - x_W + \delta_{IB}}{x_P - x_W} p_P - \frac{\delta_{IB}}{x_P - x_W} p_W. \tag{4.18}$$

This value is then extrapolated onto the ghost cell using Eq. 4.9. Note that since linear interpolation is used onto the cell faces, this is equivalent to using linear extrapolation directly onto the ghost cell. In general however, this should not be done, since if a different reconstruction were to be used, the presence of the immersed boundary would not correctly be accounted for.

Although not used at the immersed boundary in this work, Neumann boundary conditions can be applied through appropriate construction of ϕ_X. The most straightforward approach is to perform a reconstruction in the normal direction from point X using the fact that $\frac{\partial \phi}{\partial n} = 0$. This

can be done by first interpolating onto an image point (Figure 4.4), then extrapolating onto the surface along the normal direction for example.

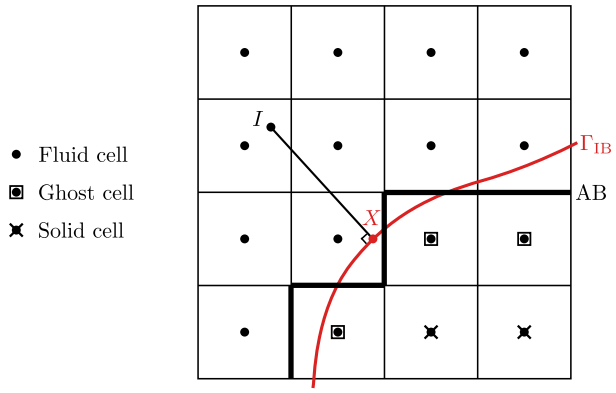


Figure 4.4: Neumann boundary conditions can be applied by constructing the field variable at immersed boundary intersection ϕ_X in the normal direction by first interpolating onto an image point I, then extrapolating along the normal line.

Once ϕ_X has been obtained, the same reconstruction methods described above can be used to obtain the ghost cell values.

4.3.7 Multigrid Transfer Operators

When using the immersed boundary method within a multigrid scheme, it is relatively straightforward to apply the above method on each grid independently to calculate the source terms and add them to the finite volume equations. Transfer of the solution and residual between grids (restriction and prolongation) near the immersed boundary however is not so clear. Ryan et al. [90] did try fully coupling a standard ghost cell approach at all grid levels, however the inter-grid transfer operators were not described near the immersed boundary. They also noted poor stability when the immersed boundary was resolved on the coarser levels, and no improvement on convergence rate. In the cases tested here, stability was not found to be an issue.

Figure 4.5 shows a fine grid overlayed with a coarse grid on a uniform mesh near an immersed boundary. The ghost cells are also marked for each corresponding grid. In the case of the coarse grid cells on the top row, where all fine grid cells are fluid cells, restriction can be performed as normal. The ghost cell on the coarse grid on the bottom right is determined using the reconstruction procedure.

In the case of the coarse grid cell on the bottom left, one of the fine grid cells which is part of the restriction is a ghost cell. Since ghost cells take on multiple values depending on the coordinate direction of reconstruction, it is not clear what value of the ghost cell should be used in the restriction. The same issue exists for the prolongation step.

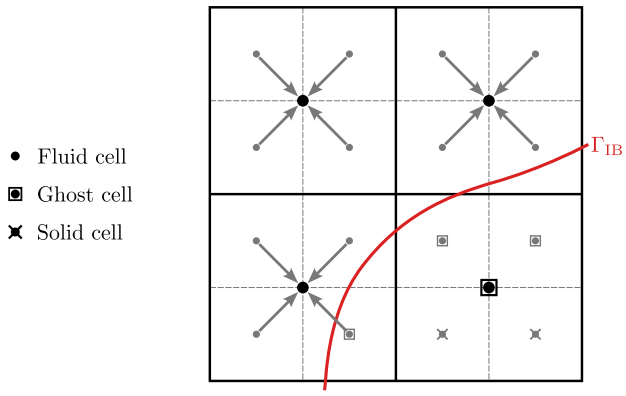


Figure 4.5: Restriction from fine grid (grey) to coarse grid (black) near the immersed boundary. The green arrows indicate the nodes which contribute to the restriction from the fine to the coarse grid.

Potential approaches include modifying the restriction and prolongation operators to account for the distance to the immersed boundary instead of the ghost cell, or setting the ghost cell values to be the average of the fluid points around them. However, this can be quite complicated, particularly in 3D since the interpolation/averaging stencils will be different depending on the configuration of the fluid and ghost cells. In the present implementation, the restriction and prolongation operators are not modified near the immersed boundary, and the zero value of the ghost cell for the pressure, velocity, and residual fields is used. This means the immersed boundary is no longer resolved in the restriction and prolongation, although it is still accounted for in the discrete equations on each coarse level. Nonetheless, this still results in good convergence rates for the multigrid method as demonstrated in Section 4.5. The additional complexity required to more accurately resolve the immersed boundary at these stages is not considered worthwhile; especially since on the coarse grids, the resolution of the immersed boundary is lowered.

4.3.8 General Remarks

For the steady solver used here, the source terms are recalculated and updated at every outer iteration. While updating more frequently can result in a higher convergence rate, it was not found to be necessary in this case, and was done at each outer iteration for simplicity. Also, due to the Picard linearisation of the momentum equations, fluxes corresponding to the faces on the AB that appear in the advection coefficients must also be modified to use the values obtained from the immersed boundary reconstruction.

In Section 4.3.1, it was stated that the order of accuracy of the reconstructions in Eqs. 4.2 and 4.3 are $O(\Delta x\, \delta_{\mathrm{IB}})$. It is generally reasonable to assume that $\delta_{\mathrm{IB}} \sim O(\Delta x)$ [121], and

so the method is second order accurate. Chi et al. [130] however note this is not strictly true. Nonetheless, the convergence rate will be better than first order, and since second order approximations are used at the interior of the domain, the solution will still converge at second order [44]. This is confirmed with convergence testing in Section 4.5.1. In general, an appropriate Δx near the IB should be determined through grid refinement studies, with the aim of getting asymptotic convergence of the solution. The precise value of Δx is highly problem dependent, however the general standard principles regarding mesh quality and the refinement of meshes around regions of large gradients still apply.

While more involved than standard ghost cell methods, the method presented here solves many of their associated issues and is still very simple to implement in an existing solver. The above calculations can all be done independent of the existing solver method. The determination of the source terms is purely geometrical, and requires little information about the discretisation. This makes it easy to convert a solver that cannot handle non-rectangular geometries into one that can. This is particularly true for coupled solvers.

4.4 Implementation Details

All geometry handling is done using CGAL (Computational Geometry Algorithms Library) [132]. With CGAL, simple geometry meshes can be created directly (such as for spheres and prisms), or more complex geometries can be read in as STL files. The Axis-Aligned Bounding Box (AABB) tree component of the library is used to perform intersection and distance queries. This includes determining if a point is inside or outside the geometry (for tagging ghost and solid cells), and calculating the distance between a given point and the surface of the geometry.

4.5 Numerical Tests

Verification using the case of a sphere inside a 3D lid driven cavity is carried out to show the 2nd order accuracy of the method. Scaling of solver time with number of cells is also analysed to demonstrate the performance of the multigrid method. Two steady validation cases are presented. Firstly, the external flow over a sphere, where drag coefficients are compared to experiments and surface pressure coefficients are compared to validated numerical data. Secondly, the flow over a 45° angled block in a channel is compared to the SIMPLE scheme in OpenFOAM using an unstructured mesh.

4.5.1 Sphere in Lid Driven Cavity

The case of a sphere inside a 3D lid driven cavity is used to verify the 2nd order accuracy of the scheme and the scaling of solver time with grid size, which should be linear for an ideally performing multigrid method. The cavity is a cube with side length $4D$, where D is the diameter of the sphere centred within the domain (Figure 4.6). The Reynolds number is based on the cavity length and driven lid velocity. Results were obtained on uniform mesh of size 20^3, 40^3, 80^3, 160^3 and 320^3.

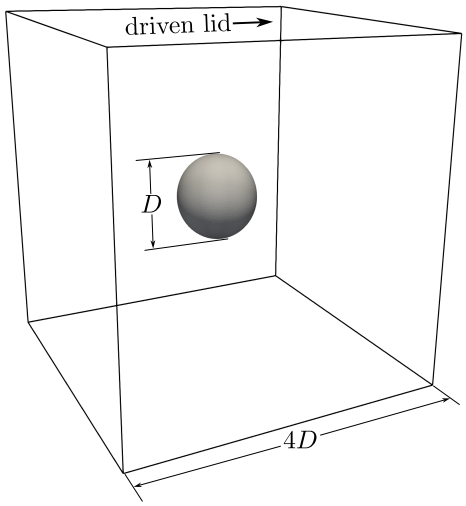
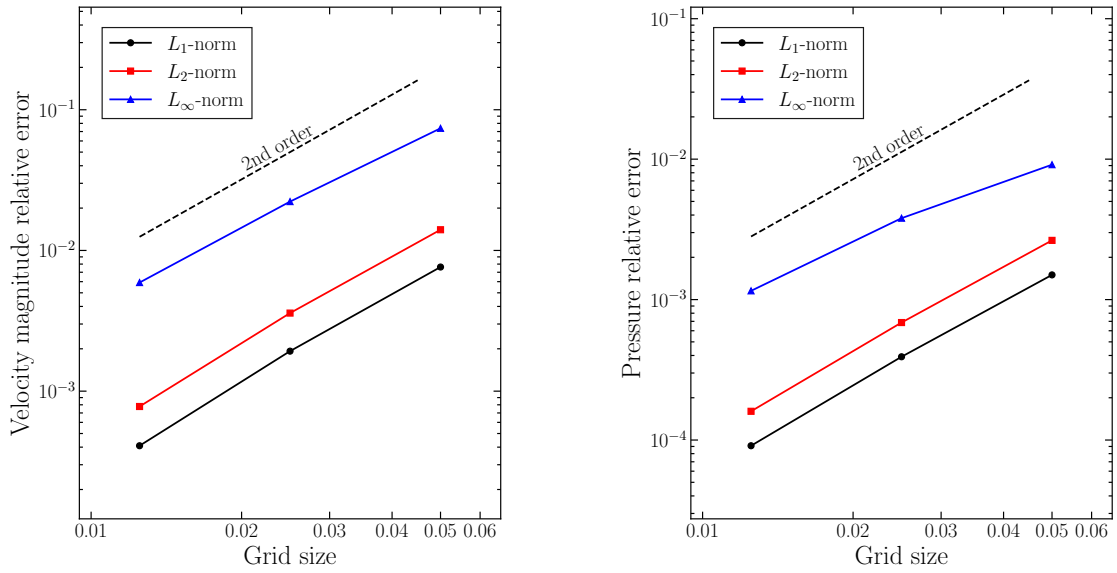


Figure 4.6: Sphere centred in lid driven cavity case used to verify order of accuracy and multigrid scaling.

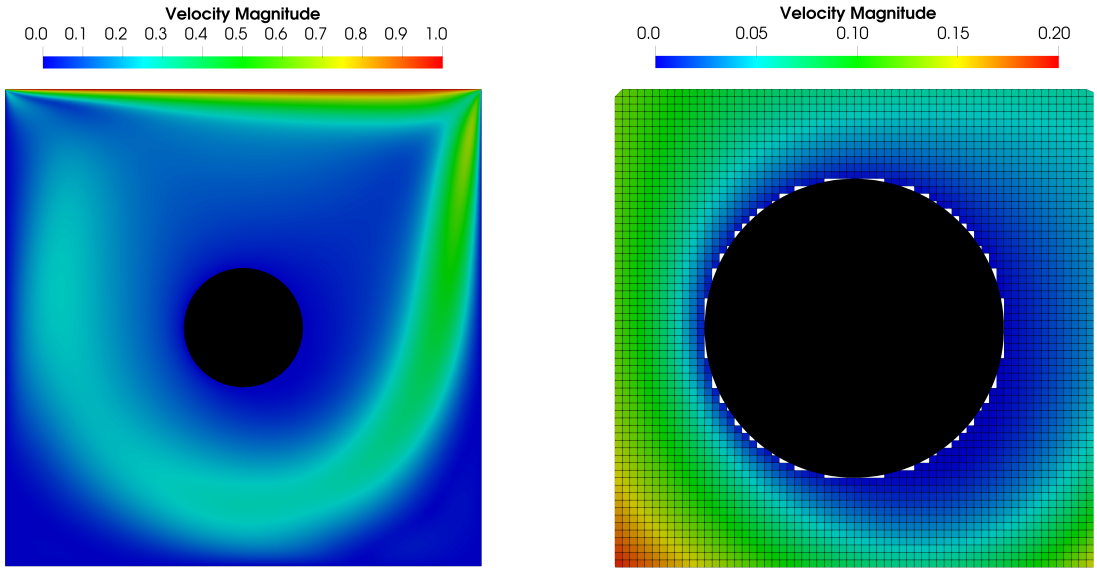
The finest grid (320^3) was taken as a reference solution and the order of accuracy was verified by calculating the L_1, L_2, and L_∞ norms of the difference between the velocity magnitude and the pressure on each coarser grid and the reference grid. The first layer of cells on the coarsest grid and the corresponding regions on the finer grids were excluded from the calculation of the norms to eliminate the effect of any interpolation/extrapolation that may be required near the boundary. Since a reference solution was used, the error will approach zero as the grid approaches the reference grid, so the grid size 160^3 was excluded from the calculation since it would approach zero error faster than the convergence rate. In other words, we must ensure the fine grid is sufficiently finer than the coarse grids used to determine the order of accuracy. Results for $Re = 200$ are shown in Figure 4.7 which show that second order accuracy is achieved under all norms for both velocity magnitude and pressure on the 40^3 ($\Delta x = 0.025$) and 80^3 ($\Delta x = 0.0125$) grids. Velocity contours of the solution at the centre plane for the case with $Re = 1000$ are shown in Figure 4.8.

Solver time for each of the grid sizes tested for a range of Reynolds numbers is shown in Figure 4.9 and Table 4.1. For the coarsest grid (20^3), the problem was solved on a single grid



(a) **(b)**

Figure 4.7: Velocity magnitude (a) and pressure (b) error norms relative to solution computed on a $320 \times 320 \times 320$ grid ($\Delta x = 0.003125$) for the sphere inside a lid driven cavity with $Re = 200$. White cells are ghost cells.



(a) **(b)**

Figure 4.8: Velocity magnitude contours for 3D sphere in lid driven cavity at centre plane for $Re = 1000$ of the complete flow field (a) and a close up of the sphere with cells shown for 160^3 mesh (b).

level, that is, multigrid was not used. The next finest grid was solved using 1 multigrid level, and 1 grid level was added to each subsequent grid, so the finest grid was solved using 4 grid levels. For the $Re = 600$ and $Re = 1000$ cases, under relaxation of 0.98 was applied to the outer iterations at each grid level to ensure a stable solution was reached. No under relaxation was

used for the other Reynolds numbers.

Certainly for the 3 finest meshes the solver time scales linearly with the number of cells. On the two coarsest meshes, the scaling is not linear, with a "kink" at the second coarsest grid level that is more pronounced with higher Reynolds numbers. This is not a cause of concern since at these coarse grid levels, overhead due to other solver operations can be quite significant relative to solution smoothing. Furthermore, at these grid sizes the smoothing qualities of a multigrid method are not fully utilised since the mesh is already coarse, so its performance is not expected to be optimal.

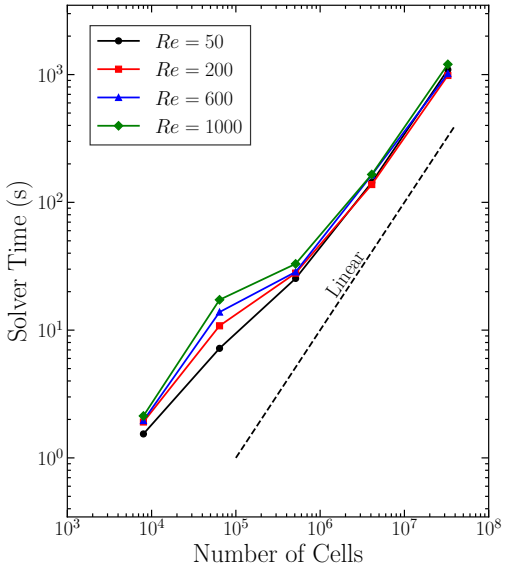


Figure 4.9: Solver CPU time with grid size for different Reynolds numbers for the sphere inside the 3D lid driven cavity.

As well as verifying the implementation of the multigrid solver, observing linear scaling with problem size indicates that the restriction/prolongation method around the immersed boundary described in Section 4.3.7 is effective and likely has little negative effect on the multigrid convergence, despite the fact that it does not fully account for the presence of the immersed boundary.

4.5.2 Sphere in Free Stream

Drag coefficients and surface pressure coefficients for the flow over a smooth sphere have been computed as validation cases for the accuracy of the immersed boundary method. Steady solutions on a non-uniform grid were computed for a range of Reynolds numbers. The domain size used was $26D \times 12D \times 12D$, where D is the sphere diameter (Figure 4.10). The sphere was placed at a distance of $9D$ from the inflow and centred in the other directions. Symmetry

Table 4.1: Raw solver CPU times with grid size for different Reynolds numbers for the sphere inside the 3D lid driven cavity as shown in Figure 4.9.

	Solver Time (s)			
Grid Size	$\boldsymbol{Re = 50}$	$\boldsymbol{Re = 200}$	$\boldsymbol{Re = 600}$	$\boldsymbol{Re = 1000}$
20^3	1.54	1.91	1.96	2.13
40^3	7.20	10.81	13.84	17.28
80^3	25.39	27.94	28.55	33.01
160^3	143.22	137.87	163.71	165.33
320^3	1092.75	984.27	1018.07	1203.56

boundary conditions were used on the boundaries parallel to the flow direction. The inflow boundary conditions were a fixed uniform velocity with zero normal pressure gradient, while the outflow was a zero normal gradient on the velocities and a fixed uniform gauge pressure.

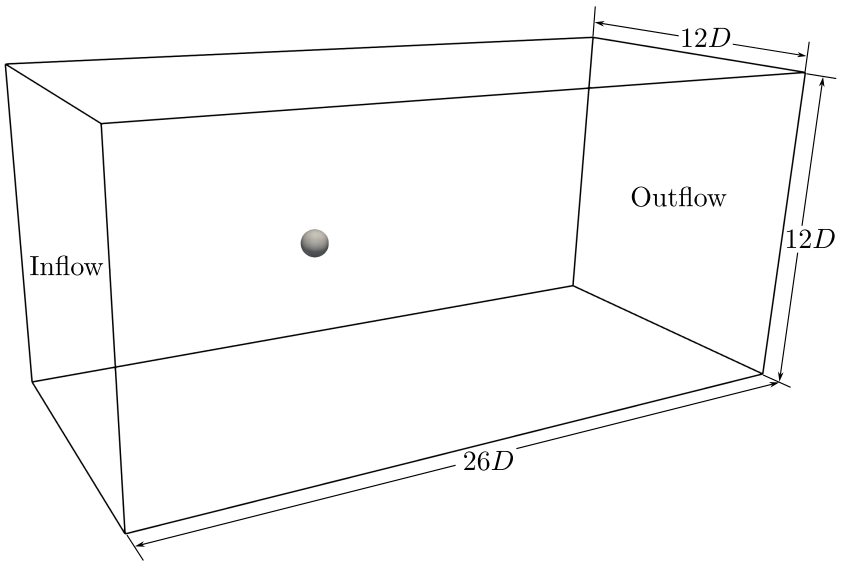


Figure 4.10: Flow over a sphere case used for method validation. All boundary conditions except the inflow and outflow were symmetry conditions, and the sphere has diameter D.

To obtain an expression for the drag force over a body in the flow, we integrate the momentum equation (Eq. 2.1) over a volume Ω which encloses the body and add a force $\boldsymbol{f}$, which is the force exerted by the fluid on the body

$$\int_\Omega \nabla \cdot (\boldsymbol{u}\boldsymbol{u}^T)\, dV = -\int_\Omega \frac{1}{\rho}\nabla p\, dV + \int_\Omega \nu \nabla^2 \boldsymbol{u}\, dV + \int_\Omega \boldsymbol{f}\, dV. \tag{4.19}$$

Using Gauss' divergence theorem, the volume integrals can be re-written as surface integrals and re-arranged for the total body force

$$\boldsymbol{F} = -\int_{\partial\Omega} \boldsymbol{u}(\boldsymbol{u}\cdot\hat{\boldsymbol{n}})\, dA - \frac{1}{\rho}\int_{\partial\Omega} p\,\hat{\boldsymbol{e}}\cdot\hat{\boldsymbol{n}}\, dA + \nu\int_{\partial\Omega}(\nabla\boldsymbol{u})\cdot\hat{\boldsymbol{n}}\, dA. \tag{4.20}$$

where $\hat{\boldsymbol{e}} = \{\hat{i}, \hat{j}, \hat{k}\}$ is the vector made of basis vectors and $\hat{\boldsymbol{n}}$ is the outward pointing unit normal vector to the surface. The first term in Eq. 4.20 is the contribution of momentum entering the domain, the second term is due to the pressure forces acting on the body, and the last term are the viscous forces. The integral can be computed on any domain that encompasses the solid, however the most convenient is to compute it around the AB. Values for the velocity and pressure on the cell faces are already available from the reconstruction, while derivatives are calculated with a single sided approximation over a half cell using the cell face value and the cell centre value immediately adjacent to the face outside Ω. The drag force is taken to be the component of $\boldsymbol{F}$ in the direction of the inflow.

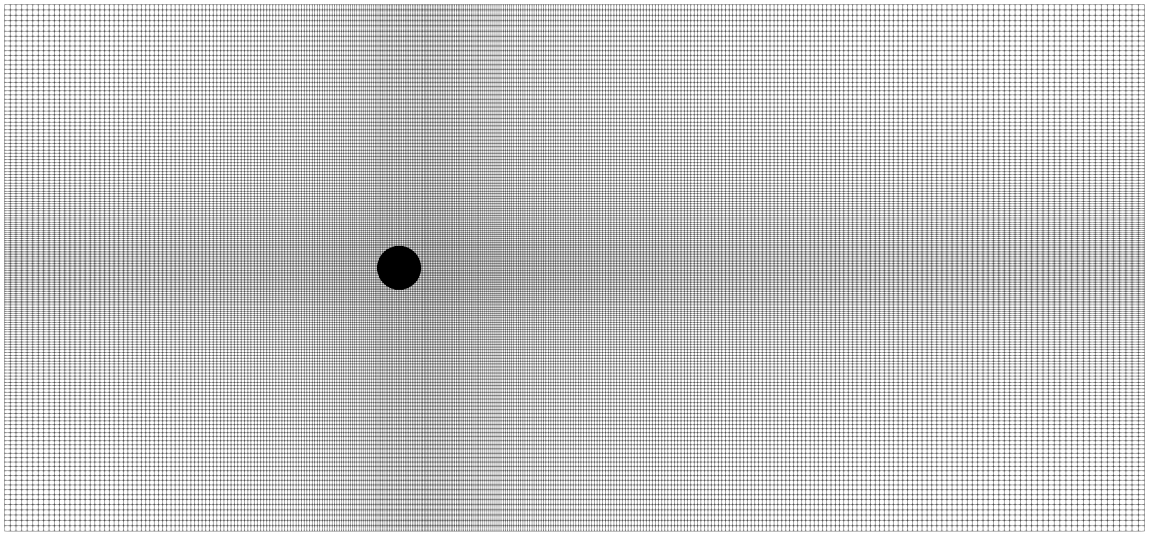
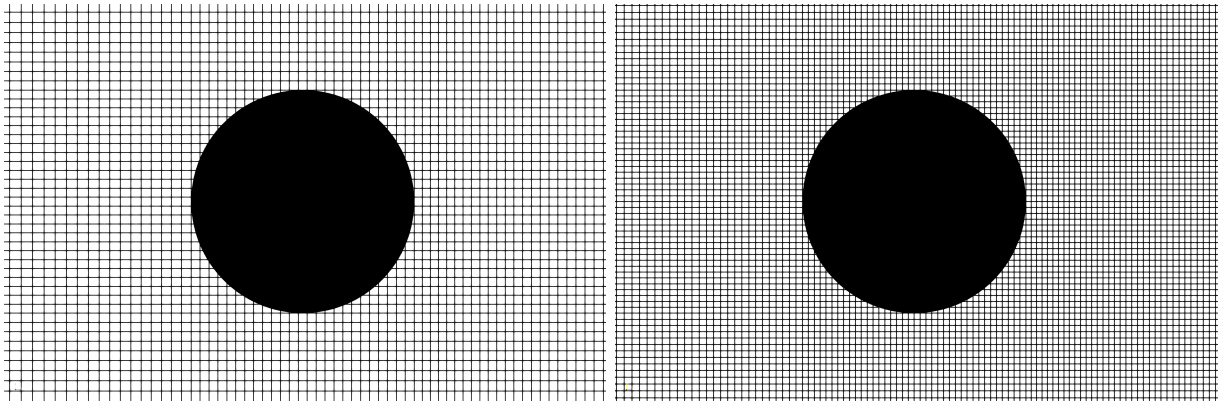


Figure 4.11: Cross-section of complete $343 \times 171 \times 171$ non-uniform mesh used to calculate drag coefficient. View is such that the inflow is on the left. The maximum cell growth ratios in each direction are 1.11301, 1.03324, and 1.03324 in the streamwise and both cross-stream directions respectively. The mesh used for the surface pressure coefficients has the same growth ratios, with the number of cells increased in each dimension.

The incompressible Navier-Stokes equations were solved using the present coupled method on a a $343 \times 171 \times 171$ non-uniform grid (Figs. 4.11 and 4.12a), and to a residual of 10^{-6}. The computed drag coefficients for Reynolds numbers from 5 to 200 are shown in Figure 4.13. Also shown are the experimental measurements of Roos and Willmarth [1]. Agreement is excellent, particularly up to $Re \sim 100$ where the flow is still steady. Beyond this, the flow is known to become unsteady and using a steady scheme to predict the drag coefficient may result in deviations from the experimental measurements. Despite this, at these higher Reynolds numbers, the agreement is still very good.

The cell centre pressure field along the centre plane around the sphere for $Re = 100$ on a finer mesh (Figure 4.12b) is shown in Figure 4.15a for the present method. Small spikes in pressure can be found in cells that lie on an internal corner of the AB. These spikes were not



(a) **(b)**

Figure 4.12: Computational mesh used to calculate sphere drag coefficient (a) and surface pressure coefficient (b). Note the full computational domain is not show here.

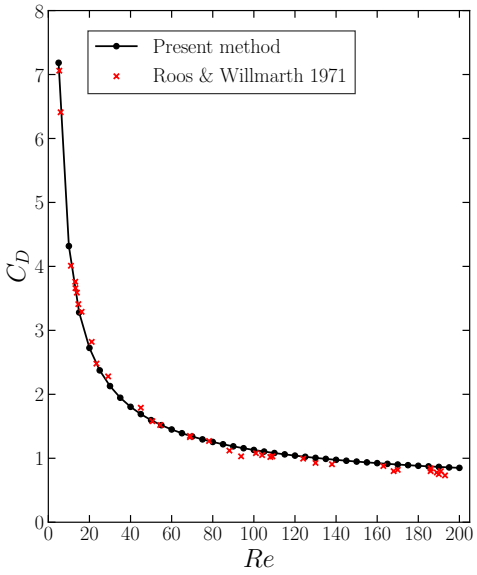
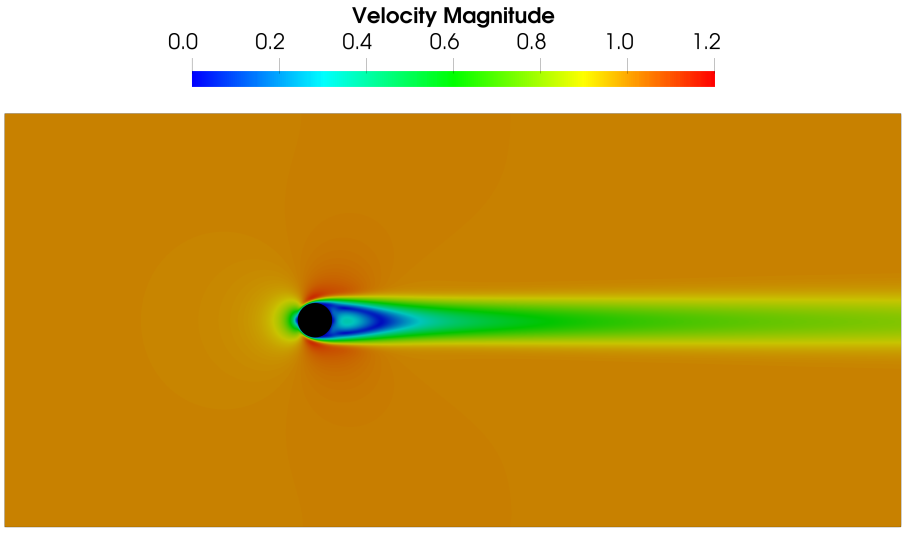


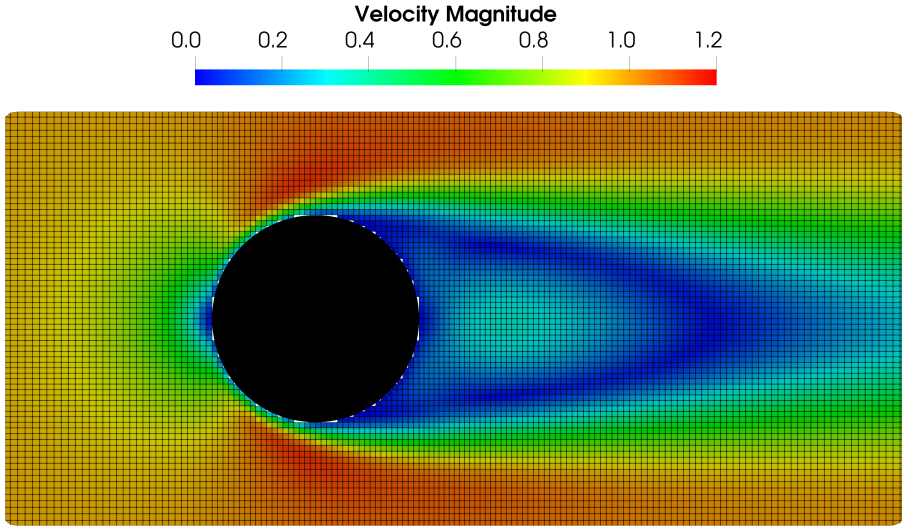
Figure 4.13: Computed drag coefficient compared with experimental data of Roos and Willmarth [1].

observed in the velocity field, which is smooth near the immersed boundary surface, as shown in Figure 4.14. Testing showed that refinement of the mesh, or removal of the mass flux correction did not eliminate the spikes or even decrease them.

If the geometry is approximated such that it conforms to the grid lines – a "staircase" approximation – these spikes still appear due to the presence of internal corners. Ghost cell immersed boundary methods implicitly create a staircase boundary, called the AB in this work (Figure 4.1). The difference being that the flux across the AB is not zero in the immersed boundary method, but set to some value that will more accurately predict the velocity around the solid boundary. For this reason, similar spikes will be observed. The authors also suspect that these spikes may arise from the stability condition discussed in Section 4.3.1. Chi et



(a)

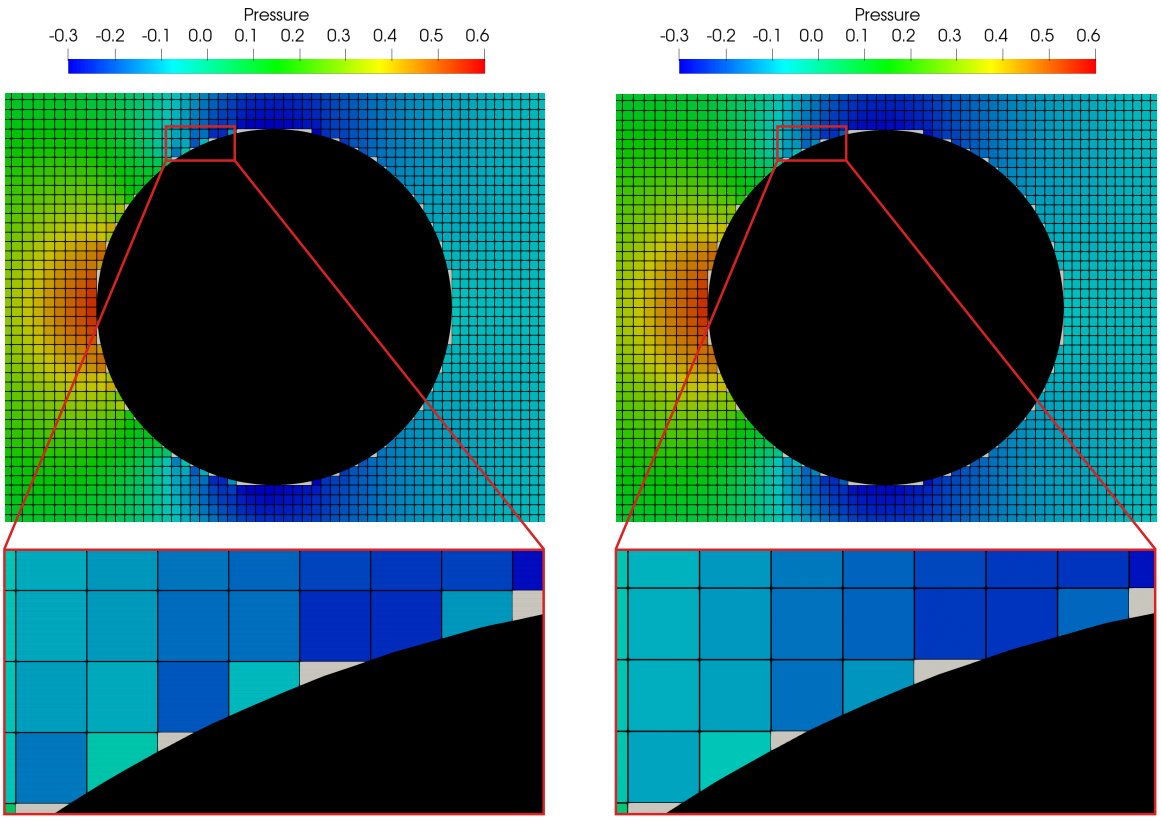


(b)

Figure 4.14: Velocity magnitude contours for flow over sphere for $Re = 100$ of the complete flow field using the reconstruction from Eq. 4.3 (a) and a close up of the sphere with cells shown (b). The white cells immediately around the sphere are ghost cells.

al. [130] who used a directional method with the same stability condition did not show pressure distributions or note the presence of these spikes. To the authors knowledge, the presence of these spikes has not been reported or discussed in the literature, but are likely present in most direct forcing immersed boundary methods. Nonetheless, the verification and validation testing in this section confirms the predicted velocity distributions are accurate, and the presence of these spikes is not necessarily a cause for concern.

It was found that the presence of the spikes could be minimised (but not completely eliminated) by an alternative reconstruction scheme. Instead of switching to Eq. 4.3 for the condition in Figure 4.3b, the velocity is simply set to zero, i.e. $\phi_e = 0$. This reconstruction is not unreasonable



(a) **(b)**

Figure 4.15: Pressure distribution around sphere for $Re = 100$ using the reconstruction from Eq. 4.3 (a) and an alternative reconstruction where $\phi_e = 0$ if the AB face lies within the solid (b) with a close up of pressure spikes near the boundary for both. The white cells immediately around the sphere are ghost cells.

since in the case of Figure 4.3b, the face centre is inside the immersed boundary, so physically, the velocity is zero. The pressure distribution for this reconstruction around the sphere surface is shown in Figure 4.15b. Although the spikes remain, the pressure field is smoother. This alternative reconstruction essentially mimics a staircase approximation for the case when the AB face lies within the solid, meaning that the reconstruction of the velocity field may be less accurate. This reconstruction is also used for the calculation of sphere surface pressure coefficients below for comparison.

A polar distribution of the surface pressure coefficient along the centreline of the sphere has been calculated for both reconstruction schemes. The surface pressure coefficient is defined as

$$C_p = \frac{p - p_\infty}{\frac{1}{2}\rho u_\infty^2}, \tag{4.21}$$

where p_∞ is taken to be the outflow gauge pressure, and u_∞ is the inflow velocity. In an immersed boundary method, the pressure p on the surface of the sphere is not directly available, so an extrapolation using the cell centre data from the fluid domain was used to obtain values on the sphere surface. This was done with a Radial Basis Function (RBF) interpolator using linear

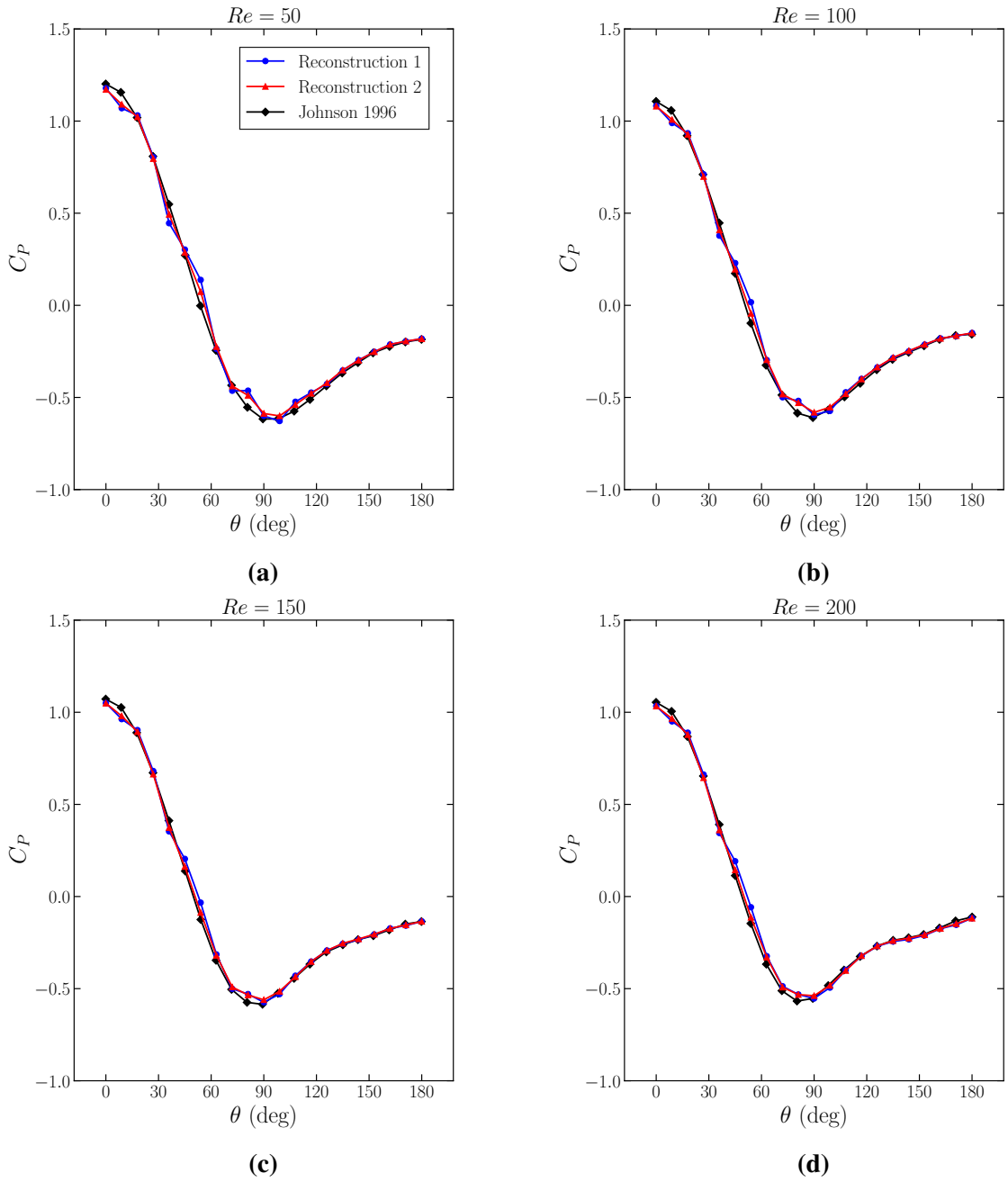


Figure 4.16: Extrapolated sphere surface pressure coefficient for a range of Reynolds numbers for two different reconstructions. Validation data is from Johnson [2]. Reconstruction 1 uses Eq. 4.3 from Section 4.3.1, while reconstruction 2 is the case where $\phi_e = 0$ if the AB face lies within the solid.

radial basis functions and the 5 nearest fluid cell centres, using the SciPy Python library [133]. Pressure coefficients were calculated on a finer $515 \times 258 \times 258$ ($\sim 33.6 \times 10^6$ cells) non uniform mesh, which is shown in Figure 4.12b for the region around the sphere. A finer mesh was used to obtain a smoother distribution for the pressure coefficient. But the prior results demonstrate that accurate drag coefficients can be obtained with a coarser mesh. Plots of the pressure coefficients are given in Figure 4.16 for a range of Reynolds numbers, and compared with the data given in

Johnson [2].

Due to the RBF extrapolation onto the surface which used the 5 nearest neighbours, nearby corner pressure spikes influenced the pressure coefficient, which can be seen in Figure 4.16. In general, agreement is good, particularly on the lee side of the sphere. As the Reynolds number increases, the spikes become smaller in magnitude. Using the alternative reconstruction method does mitigate the spikes, but does not eliminate them completely. As discussed before, the spikes originate due to internal corners in the AB, which are independent of the reconstruction method.

Table 4.2 shows the L_1-norm of the C_P error against the validation data for both reconstructions both with and without the mass flux correction. The percentage improvement is also indicated. For the original reconstructions, an improvement is only gained when the Reynolds number is greater than 200. For the second reconstruction, the correction always resulted in an improvement. This is reasonable to expect since the second reconstruction does not represent the solid boundary as faithfully as the method described in Section 4.3.1, so the resulting mass flux error may be larger. Addition of the correction has a more significant effect in this case.

Table 4.2: L_1-norm of C_P error for present method against validation data of Johnson [2] for reconstruction 1 (a) and reconstruction 2 (b), with and without the mass flux correction. A positive percentage improvement indicates that the solution is more accurate with the mass flux correction.

(a)

	L_1-norm of C_P error		
Re	**Without Correction**	**With Correction**	**% Improvement**
50	0.03241988285	0.03268878422	-0.8294334762
100	0.02745154669	0.02745804099	-0.0236573133
150	0.02467949377	0.02408928746	2.3914847001
200	0.02728493566	0.02700711850	1.0182071031

(b)

	L_1-norm of C_P error		
Re	**Without Correction**	**With Correction**	**% Improvement**
50	0.02494613792	0.02450334450	1.7749978783
100	0.02124424978	0.02119032868	0.2538150554
150	0.01757969153	0.01712275835	2.5992104541
200	0.01838705571	0.01783115112	3.0233475056

4.5.3 45° Angled Surface Mounted Cube in Channel

Finally, we consider a surface mounted cube in an open channel angled at 45° relative to the flow direction. The domain used had dimensions $10H \times 4H \times 3H$, where H is the side length of the cube. The top boundary condition was a symmetry condition and the other boundaries parallel to the flow direction were solid non-slip walls (Figure 4.17). The same inflow and outflow conditions as in the previous section were used. The problem was solved using the present method on two different non-uniform meshes: a $275 \times 109 \times 116$ fine mesh ($\sim 3.5 \times 10^6$ cells), and a $125 \times 50 \times 53$ coarse mesh ($\sim 3.3 \times 10^5$ cells). The coarse mesh is shown in Figure 4.18, with the fine mesh having the same growth rates in each direction, with the number of cells increased. These meshes in the vicinity of the immersed boundary are shown in Figure 4.19.

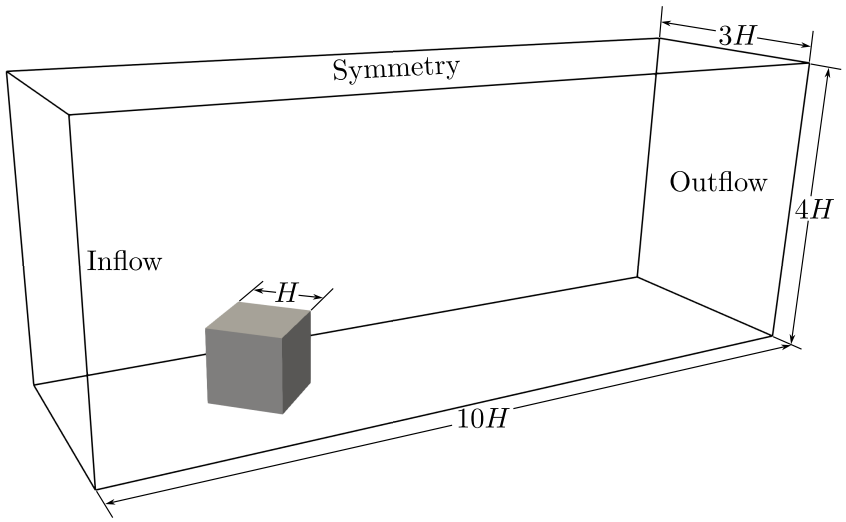
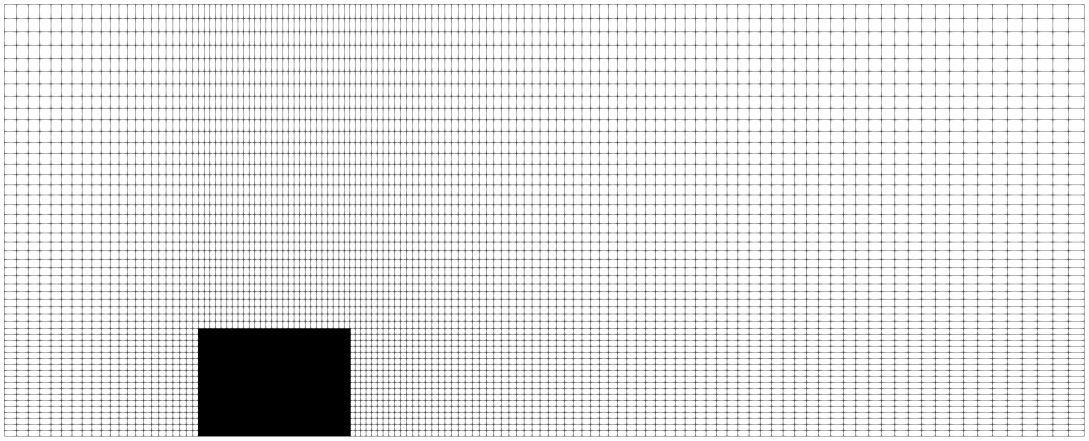


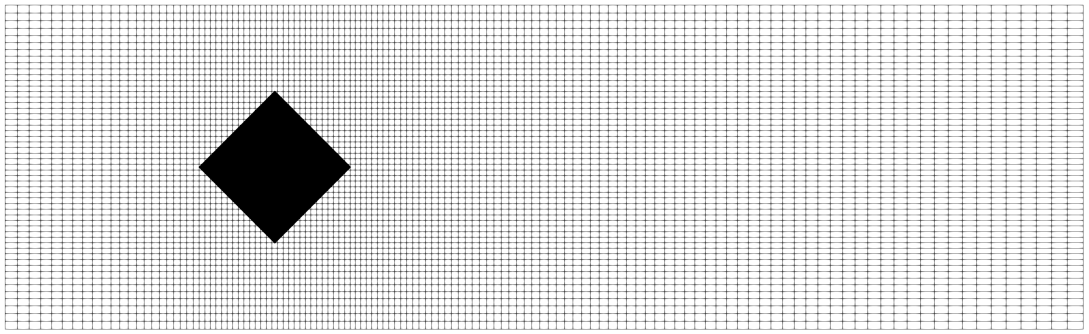
Figure 4.17: Flow over 45° angled cube in channel. All boundary conditions except ones marked are walls.

The same problem was also solved with OpenFOAM v9 [94] using the SIMPLE scheme on an unstructured body fitted mesh created using the "snappyHexMesh" utility with $\sim 3.5 \times 10^6$ cells. The same second order discretisation was used for the OpenFOAM solution, except a second order upwind scheme was used for the advection instead of QUICK, which was used in the present method.

Comparison of the stream-wise velocity between the three solutions along the centreline and off-centre are shown in Figure 4.20. The plots show excellent agreement with the OpenFOAM reference solution, even when using the coarse mesh. By interpolating the reference solution onto the fine and coarse grids the L_1-norm of the relative velocity magnitude error normalised by the inflow velocity was computed both with and without the mass flux correction. These errors can be found in Table 4.3. On the coarse mesh, an accuracy improvement of $\sim 2\%$ was observed, while $\sim 0.4\%$ was observed on the fine mesh. As the mesh is refined and the AB becomes a

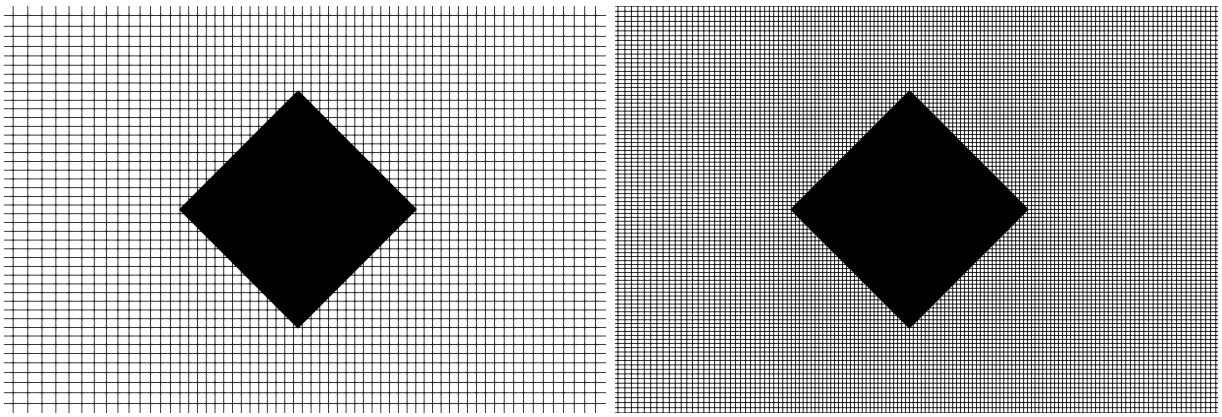


(a)



(b)

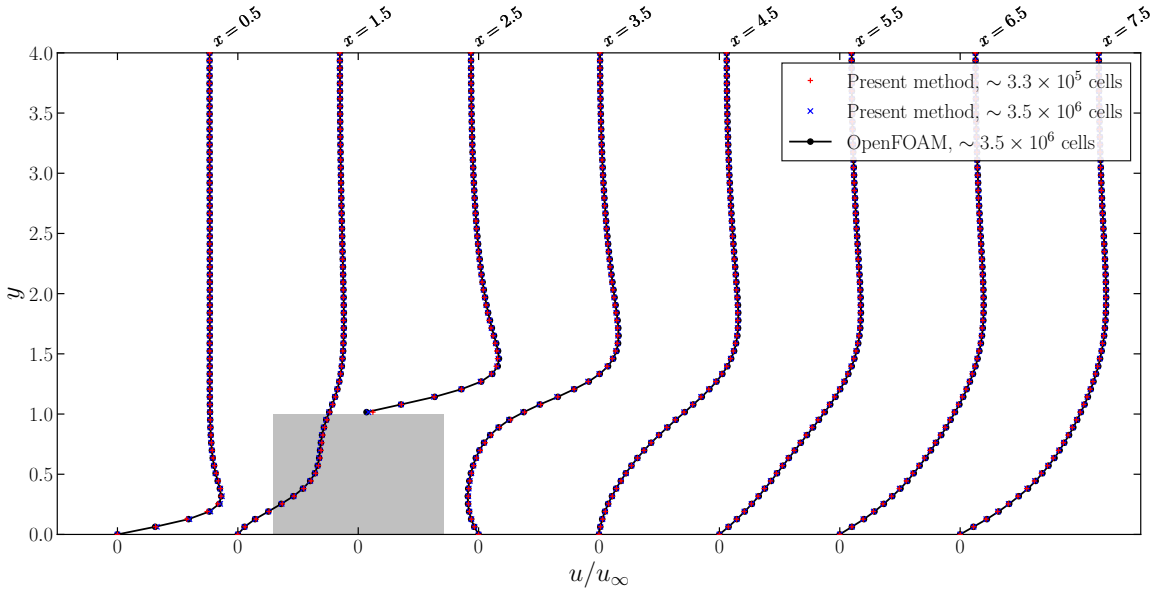
Figure 4.18: Cross-sectional view from the side (a) and top (b) of complete $125 \times 50 \times 53$ coarse coarse mesh used for the 45° angled cube case. The maximum cell growth ratios are 1.09167, 1.16819, and 1.04023 in the streamwise, vertical, and cross-stream directions respectively. The fine mesh has the same growth ratios, with the number of cells increased in each dimension.



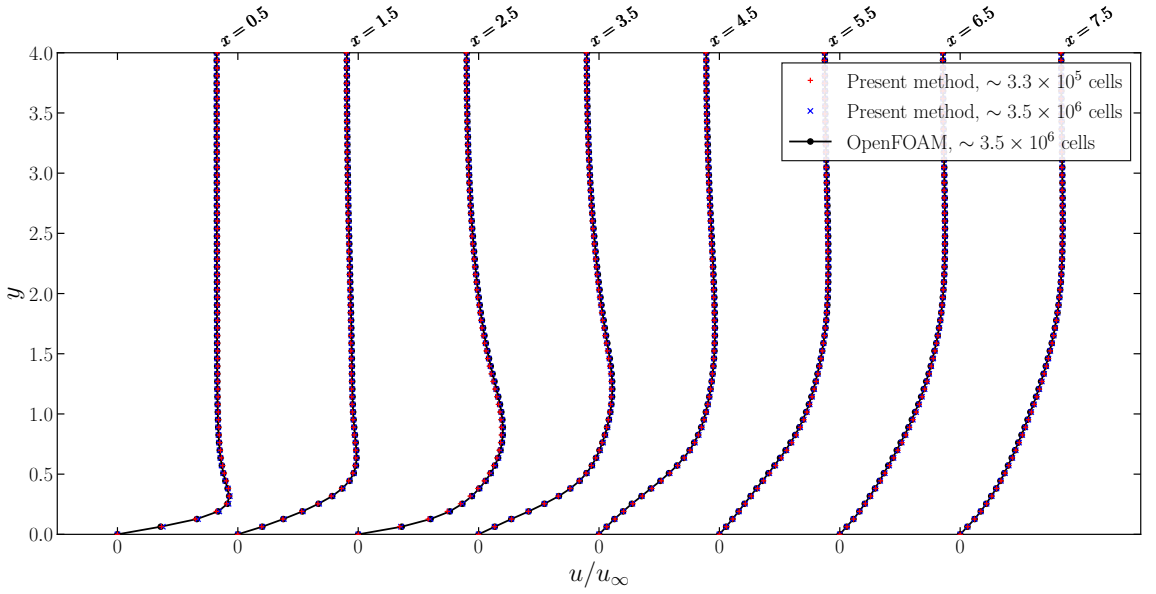
(a) **(b)**

Figure 4.19: Top view of coarse mesh with $\sim 3.3 \times 10^5$ cells (a) fine mesh with $\sim 3.5 \times 10^6$ cells (b) near the solid cube. Note the full computational domain is not show here.

better approximation of the solid boundary, the reconstructed velocity will approach the velocity of the solid surface (which is zero). This means the mass flux error approaches to zero as Δx



(a)



(b)

Figure 4.20: Velocity profile comparison of present method for a fine and coarse grid with a fine unstructured grid in OpenFOAM along the centreline of the domain (a) and off-centre, halfway between the block and wall boundary (b).

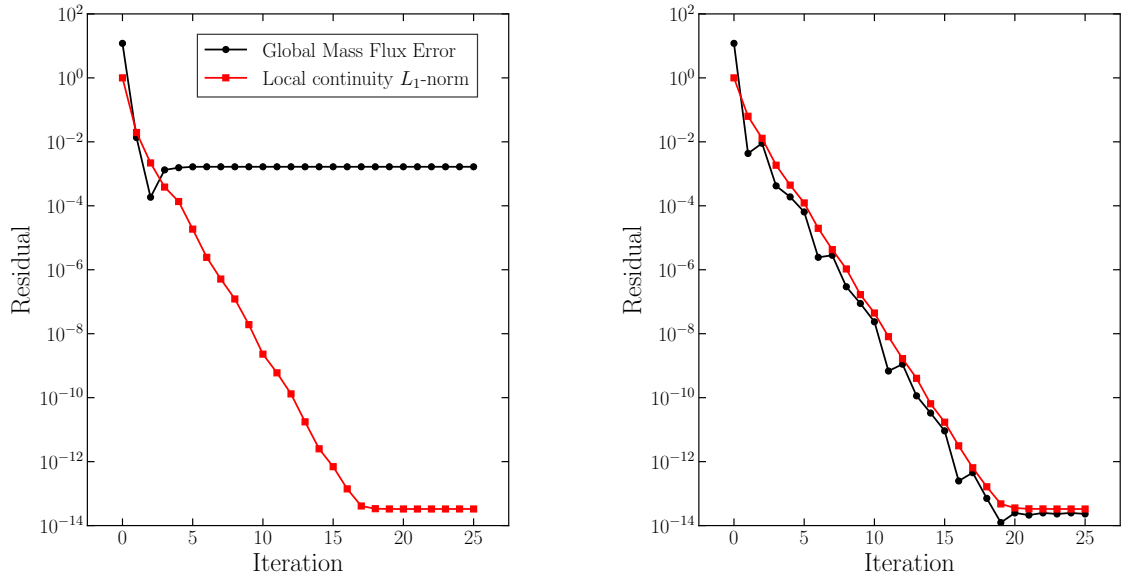
approaches zero, even without the correction, which explains why the improvement is smaller on the finer grid. The results in Table 4.3 indicate that the use of the mass flux correction allows one to gain better solution accuracy when using coarser grids.

All problems were solved on the same computer in serial i.e. using a single CPU core (Intel i9-11900F). The present method solved to a residual of 10^{-6} in 10.5 s with 4 multigrid iterations on the coarse grid, and 129 s with 4 multigrid iterations on the fine grid. OpenFOAM on the

other hand took 4499 s to compute the solution to the same residual in 942 outer iterations, not including meshing time. A major contributor to the significant performance advantage is the fully coupled smoother with FAS multigrid. Alongside this, the immersed boundary method on a rectilinear grid means the code can use simple array data structures that are very efficient to access. Furthermore, the extra cost associated with the computation of the immersed boundary source terms is very small. All immersed boundary operations (reconstruction, flux correction, ghost cell determination, and source term calculation) made up only 0.3% of the solver time for the fine grid, and 0.6% on the coarse grid.

Table 4.3: L_1-norm of velocity magnitude error for present method on the coarse and fine grid against fine unstructured grid in OpenFOAM both with and without the mass flux correction. A positive percentage improvement indicates that the solution is more accurate with the mass flux correction.

	L_1-norm of velocity magnitude error		
Grid	**Without Correction**	**With Correction**	**% Improvement**
coarse	0.004138123810	0.004052356962	2.0726022630
fine	0.003028628018	0.003017623572	0.3633475340



(a) **(b)**

Figure 4.21: Local and global mass continuity convergence history for angled block case with $Re = 50$ on the fine mesh without (a) and with (b) mass flux correction.

Figure 4.21 shows the local and global continuity convergence history for the angled block problem on the fine grid both with and without the mass flux correction (Section 4.3.2). The

global mass flux error is calculated by integrating the mass flux through the domain boundaries (excluding the immersed boundary). In both cases, local continuity for the fluid cells is satisfied and the residual goes to machine zero. However, when the mass flux correction is not applied, there is a constant global mass flux error that remains in the solution which arises from the reconstruction. When the correction is applied, the global mass flux error goes to machine zero as the solution converges. From Figure 4.21 it is evident that the application of the correction has little effect on the convergence rate of the local continuity error.

4.6 Summary

The immersed boundary method was chosen for its simple and lightweight implementation, computational efficiency, and ease of meshing when dealing with complex geometries. This chapter built upon the basic smoother method developed in Chapter 3. The coupled nature of the method and the fact that a collocated grid introduced some difficulties in implementation regarding global mass conservation, and the treatment of MWI that had not been previously addressed in literature. The present approach of using a directional ghost cell method with a mass flux correction based on the global mass error solved these issues with little additional complexity. Verification and validation testing demonstrated that the method is second order accurate, conserves mass both locally and globally, and is still very computationally efficient. The present chapter along with Chapter 3 form a complete foundation for an efficient steady solver method that can deal with complex geometries. The only remaining element is the addition of a turbulence model to allow for the solution of the RANS equations. This is the subject of the following chapter.

CHAPTER 5

Turbulence Modelling

5.1 Introduction

Steady-state solutions to turbulent flows are commonly obtained by solving the RANS equations. In this case, the solution is actually a time averaged one. The Reynolds stresses which appear as unknowns in the RANS equations need to be determined from a turbulence model. These turbulence models can be transport equations for each of the Reynolds stresses, or come from the use of an eddy viscosity model. As has been the case throughout this thesis, fast simulation time is the primary aim, so the simplicity and convenience of an eddy viscosity model make it the natural choice. The eddy viscosity will be obtained using a zero-equation, or algebraic turbulence model. This is done for two reasons. Firstly, the simplicity of zero-equation models naturally lend themselves to being very computationally efficient. Secondly, using a simple zero-equation model is the logical first step in validating and testing the performance of the algorithm when solving the RANS equations before potentially introducing more complex models.

Arguably, the $k-\varepsilon$ model [134] is the most widely adopted turbulence model. It is available in most commercial CFD packages, and has been applied to a wide range of problems [135–137] including urban flows [138, 139]. The $k-\varepsilon$ model is known to perform reasonably well for shear flows with small streamline curvature and mean pressure gradients but its accuracy can suffer in cases with strong pressure gradients and flow separation [140]. This means for urban flows, it struggles to predict many of the recirculating flow features that appear around buildings [141–143]. Attempts to improve the standard $k-\varepsilon$ model have been made through modifications such as the RNG $k-\varepsilon$ model [144], the $k-\varepsilon$ with Launder and Kato modification [145], and the Realizable $k-\varepsilon$ model [146]. While it is often reported that these models outperform the standard $k-\varepsilon$ [147, 148], there is some uncertainty in the literature, with some authors reporting better performance from the standard model for some urban flow cases [149, 150].

The $k-\omega$ model [151], and perhaps more commonly the Shear Stress Transport (SST) $k-\omega$ model [152] are other models that have been applied to a wide range of flows. For flows with strong pressure gradients, the $k-\omega$ model is typically superior to $k-\varepsilon$ models. A number of authors have tested the model for urban flows [153–155] with results being satisfactory at best, and the advantage over the $k-\varepsilon$ model being not so apparent.

Given all this, it is evident that in its current state, turbulence modelling (at least as far as eddy viscosity models are concerned) still fails to give consistently accurate predictions of velocity fields in complex urban-like environments. While some of these inaccuracies come from the particular turbulence model itself, they also stem from the fundamental assumptions of the eddy viscosity hypothesis [140]. However, this is not to say that their predictions are not useful. When their relative cost compared to LES for example is taken into account, they are a very attractive – and sometimes the only – option. In any case, this naturally raises the question: if they fail to produce accurate predictions in the first place, why incur their additional computational expense at all? For this reason, some authors have looked towards simpler turbulence models which may still provide useful results, at only a fraction of the computational time.

Zero-equation turbulence models have found use in a range of engineering applications for their efficiency, including ship hydrodynamics [156, 157], supercritical fluid flows [158], two-phase flows [159], Heating, Ventilation, and Air-Conditioning (HVAC) [160, 161], and turbo machinery [162, 163] to name a few. In relation to urban flows, Gowardhan [41] and Gowardhan et al. [42] have made use of a simple mixing length zero-equation model in the QUIC-CFD code for fast response urban applications. They demonstrated that the cost savings that come from these models make them valuable in situations where rapid simulation time is a priority. Since then, specialised turbulence models for outdoor urban environments have been developed. These came as developments to the model initially introduced by Chen and Xu [160] for indoor airflow. Validation tests for indoor environments with natural and forced convection have shown reasonable predictions, especially when the accuracy of the $k-\varepsilon$ model is taken into consideration as well [164]. The attractive feature of this model was its computational cost with Chen and Xu noting that the zero-equation model was an order of magnitude faster than the $k-\varepsilon$ model they tested. Following this, models such as that by Qian and Srebric [165] were based on the original Chen and Xu model with adjusted constants for flow characteristics found in outdoor flows based on experiments for the flow over an isolated building. With the aid of more extensive experimental data, further improvements were made to these zero-equation models by authors such as Davidovic et al. [166, 167], Liu et al. [168], and Li et al. [169]. These more sophisticated (but still simple) models typically contain problem dependent parameters

that depend on the particular geometry, such as the average building height and width and roughness height for example and are relatively easy to estimate based on available geometrical and problem information. Liu et al. [170] conducted an extensive comparison between a range of zero-equation turbulence models, including the ones mentioned above with a standard $k-\varepsilon$ model and an LES model for three different urban-like flow configurations. They conclude that while zero-equation models are not as consistently accurate to LES and wind tunnel experiments, their significantly faster run time make them a promising and valuable tool in engineering practise, especially when compared to other RANS models such as the $k-\varepsilon$ model. For this reason, there has been an ongoing effort by researchers to create and optimize simple RANS turbulence models using a range of approaches from traditional optimization techniques [171] to machine learning methods [172]. While these approaches have so far proven to be quite promising, exploring them is not within the scope of this work, and readers are referred to the recent review by Liu et al. [173] for a comprehensive overview.

Given that more sophisticated turbulence models (such as the $k-\varepsilon$ model) also fail to give accurate predictions and require significantly more computational effort, their usefulness in applications where simulation time is a priority becomes questionable. The aim of this work is to produce fast running models, and so zero-equation models are a natural choice. In this chapter, the zero-equation model of Liu et al. [168] is implemented into the solver developed in Chapters 2 - 4. The eddy viscosity model is introduced in Section 5.2, and its discretisation is given in Section 5.3. Details of the model used and its implementation are given in Secs. 5.4 and 5.5. The turbulence model is validated against three urban flow cases which are part of the University of Hamburg CEDVAL experimental reference data set [63]: the flow over a 40° angled cube (CEDVAL A1-7), the flow over an array of rectangular blocks (CEDVAL B1-1), and the flow over an intersection (CEDVAL B1-5). The accuracy and performance of the method is also compared to the standard $k-\varepsilon$ model in OpenFOAM [94] (using the SIMPLE scheme) and ANSYS Fluent [93] (using the Coupled solver).

5.2 The Eddy Viscosity Model

Performing Reynolds averaging on the incompressible Navier-Stokes equations gives the RANS equations [140]

$$\nabla \cdot (\overline{\boldsymbol{u}} \otimes \overline{\boldsymbol{u}}) = -\frac{1}{\rho}\nabla \overline{p} + \nabla \cdot \left(\nu \left\{ \nabla \overline{\boldsymbol{u}} + (\nabla \overline{\boldsymbol{u}})^T \right\} - \overline{\boldsymbol{u}'\boldsymbol{u}'} \right) \tag{5.1}$$

$$\nabla \cdot \overline{\boldsymbol{u}} = 0, \tag{5.2}$$

where $\overline{\square}$ indicates a time averaged quantity and $\overline{\boldsymbol{u}'\boldsymbol{u}'}$ is the Reynolds stress tensor. The complete viscous stress tensor $\nu\left\{\nabla\overline{\boldsymbol{u}} + (\nabla\overline{\boldsymbol{u}})^T\right\}$ has been included since it simplifies the formulation of the eddy viscosity model, which will become more apparent below. In the case of an incompressible fluid with constant viscosity, the second term in the viscous stress tensor $(\nabla\overline{\boldsymbol{u}})^T$ is zero and can be left out, which is what was done up until this point. The Reynolds stresses are determined by a turbulence model. Here, this is done via the eddy viscosity model. The eddy viscosity model models the Reynolds stresses using a relationship that is directly analogous to the viscous stresses of a Newtonian fluid. For an incompressible flow, it is given by:

$$\overline{\boldsymbol{u}'\boldsymbol{u}'} - \frac{2}{3}k\boldsymbol{I} = \nu_T\left\{\nabla\overline{\boldsymbol{u}} + (\nabla\overline{\boldsymbol{u}})^T\right\}, \tag{5.3}$$

where $k = \frac{1}{2}\overline{\boldsymbol{u}'\cdot\boldsymbol{u}'}$ is the turbulent kinetic energy, ν^T is the turbulent or eddy viscosity, and $\boldsymbol{I}$ is the identity matrix. The second term on the left hand side $-\frac{2}{3}k\boldsymbol{I}$ is combined with the pressure gradient term to form the turbulence pressure $\overline{p}$ as

$$\overline{p} \leftarrow \overline{p} + \frac{2}{3}\rho k. \tag{5.4}$$

From here, the overbar is dropped from the averaged quantities to simplify notation, it is assumed all quantities are time averaged. Using Eqs. 5.3 and 5.4, the RANS equations with an eddy viscosity model become

$$\nabla\cdot(\boldsymbol{u}\otimes\boldsymbol{u}) = -\frac{1}{\rho}\nabla p + \nabla\cdot\left(\nu_{\text{eff}}\left\{\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right\}\right) \tag{5.5}$$

$$\nabla\cdot\boldsymbol{u} = 0, \tag{5.6}$$

where $\nu_{\text{eff}} = \nu + \nu_T$ is the effective viscosity. Note that unlike the laminar equations, the term $(\nabla\boldsymbol{u})^T$ must be retained since ν_{eff} can be spatially varying. With this, the problem of calculating the Reynolds stresses is now a matter of calculating the turbulent viscosity ν_T and using it to solve Eqs. 5.5 and 5.6. The models used to calculate the turbulent viscosity are given in Section 5.4.

5.3 Discretisation of The Eddy Viscosity Model

As was done in Chapter 2 for the discretisation of the complete Navier-Stokes equations, we take the volume integral of of the equations and use Gauss' divergence theorem to obtain the integral form of the equations. Taking the entire viscous term in Eq. 5.5

$$\int_\Omega \nabla\cdot\left(\nu_{\text{eff}}\left\{\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right\}\right)dV = \int_{\partial\Omega}\nu_{\text{eff}}\left\{\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right\}\cdot\hat{\boldsymbol{n}}\,dS. \tag{5.7}$$

Rewriting the discrete integrals as summations over the cell faces, the viscous term becomes

$$\int_{\partial\Omega} \nu_{\text{eff}} \left\{\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right\} \cdot \hat{\boldsymbol{n}}\, dS \approx \sum_f \left(\nu_{\text{eff}} \left\{\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right\} \cdot \hat{\boldsymbol{n}}\right)_f A_f. \tag{5.8}$$

Equation 5.8 is broken up into two parts, an explicit part, and an implicit part:

$$\sum_f \left(\nu_{\text{eff}} \left\{\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T\right\} \cdot \hat{\boldsymbol{n}}\right)_f A_f = \underbrace{\sum_f \left(\nu_{\text{eff}} \nabla \boldsymbol{u} \cdot \hat{\boldsymbol{n}}\right)_f A_f}_{\text{implicit}} + \underbrace{\sum_f \left(\nu_{\text{eff}} (\nabla \boldsymbol{u})^T \cdot \hat{\boldsymbol{n}}\right)_f A_f}_{\text{explicit}}. \tag{5.9}$$

The implicit part is treated as part of the finite volume stencil, while the explicit part is included in the source terms of the discrete equations and updated every outer iteration. Note that the implicit part is the term in common with the laminar equations (with the exception of an effective viscosity being used), while the explicit part is an additional term that arises from the eddy viscosity model and the fact that the effective viscosity is spatially varying.

The finite volume discretisation in Eq. 5.9 indicates that the effective viscosity is required at cell faces. Since a collocated grid arrangement is used, the turbulent viscosity ν_T is calculated and stored at the cell centres. While it is possible in principle to calculate and store this at cell faces, particularly for zero-equation turbulence models, cell centres is the natural choice. In order to be consistent with the physics of diffusion according to Fick's law, harmonic interpolation from the cell centres should be used [64, 174], which is given by

$$\nu_e = \frac{\nu_P \nu_E}{(1-\lambda_e)\nu_P + \lambda_e \nu_E}. \tag{5.10}$$

Discretisation of the implicit term in Eq. 5.9 is identical to what was shown in Chapter 2.4.1, with the difference being that the viscosity is no longer constant, and kept inside the summation. So for a given velocity component ϕ, in 1D, the stencil is given by

$$\sum_f (\nu_{\text{eff}} \nabla \phi \cdot \hat{\boldsymbol{n}})_f A_f = \left(\nu_{\text{eff}} \frac{\partial \phi}{\partial x}\bigg|_e A_e - \nu_{\text{eff}} \frac{\partial \phi}{\partial x}\bigg|_w A_w\right) \tag{5.11}$$

$$\approx \left(\frac{(\nu_{\text{eff}})_e A_e}{\Delta x_e}\phi_E - \left(\frac{(\nu_{\text{eff}})_e A_e}{\Delta x_e} + \frac{(\nu_{\text{eff}})_w A_w}{\Delta x_w}\right)\phi_P + \frac{(\nu_{\text{eff}})_w A_w}{\Delta x_w}\phi_W\right). \tag{5.12}$$

We now consider the explicit term in Eq. 5.9. We will draw our attention to the term inside the brackets, which can be expressed more explicitly as

$$(\nabla \boldsymbol{u})^T \cdot \hat{\boldsymbol{n}} = \begin{pmatrix} \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} & \frac{\partial w}{\partial x} \end{pmatrix}^T \cdot \hat{\boldsymbol{n}} \\ \begin{pmatrix} \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} & \frac{\partial w}{\partial y} \end{pmatrix}^T \cdot \hat{\boldsymbol{n}} \\ \begin{pmatrix} \frac{\partial u}{\partial z} & \frac{\partial v}{\partial z} & \frac{\partial w}{\partial z} \end{pmatrix}^T \cdot \hat{\boldsymbol{n}} \end{pmatrix} \tag{5.13}$$

where each row corresponds to each of the x, y and z momentum equations. Note that this term is evaluated on the cell faces. In contrast to the term $\nabla \boldsymbol{u} \cdot \hat{\boldsymbol{n}}$ where only derivatives in the normal direction to each cell face are required, $(\nabla \boldsymbol{u})^T \cdot \hat{\boldsymbol{n}}$ requires that derivatives in all directions be calculated on the cell faces. These gradients are calculated in two different ways. Terms which come from the diagonal elements of $(\nabla \boldsymbol{u})^T$ are derivatives in the cell face normal direction, and identical to those in $\nabla \boldsymbol{u}$, so they are calculated in the same way using the central difference approximation given by Eq. 2.9. Terms that come from the off-diagonal elements of $(\nabla \boldsymbol{u})^T$ on the other hand are derivatives taken in the in-plane direction of the cell face. These derivatives are obtained by interpolating cell centre derivatives onto cell faces. Taking the e face as an example, the derivative of a quantity ϕ in the y-direction would be approximated as

$$\left.\frac{\partial \phi}{\partial y}\right|_e \approx (1-\lambda_e)\left.\frac{\partial \phi}{\partial y}\right|_P + \lambda_e \left.\frac{\partial \phi}{\partial y}\right|_E. \tag{5.14}$$

Cell centred derivatives are then calculated using a central difference scheme with linear interpolation:

$$\left.\frac{\partial \phi}{\partial y}\right|_P \approx \frac{\phi_n - \phi_s}{\Delta y} \tag{5.15}$$

$$= \frac{\lambda_n \phi_N + (1-\lambda_n-\lambda_s)\phi_P - (1-\lambda_s)\phi_S}{\Delta y}. \tag{5.16}$$

For simplicity, at domain boundaries, cell face derivatives are approximated by taking the cell centre derivative of the adjacent cell. This is also done for the cell faces on the AB near the immersed boundary.

5.4 The Zero-Equation Model

In this work, the zero-equation model of Liu et al. [168] is implemented into the present method. Compared to the other models tested by Liu et al. in their comparison [170], this model along with the model by Li et al. [169] gave the best performance, with no clear advantage of one over the other in terms of accuracy. The model by Liu et al. was preferred however for two reasons. Firstly, the model by Li et al. required many more problem dependent constants that are obtained from scaled wind tunnel experiments for a particular city. This makes the model less general and more difficult to apply to different scenarios more accurately. Secondly, in the authors experience, Li et al.'s model had poor robustness properties, and for many of the flow cases tested, a converged solution could not be obtained. This was not much of an issue with Liu

et al.'s model. The expression for the eddy viscosity is given by

$$\begin{aligned} \nu_T &= \max\left\{\nu_{\text{in}}, \nu_{\text{out}}\right\} \\ \nu_{\text{in}} &= \alpha z_h \exp\left(-b\frac{l}{H}\right)\left(\frac{l}{H}\right)^2 |\boldsymbol{u}| \\ \nu_{\text{out}} &= 0.16 U_H \frac{z+z_0}{\ln\left(\frac{z+z_0}{z_0}\right)} \end{aligned} \tag{5.17}$$

where ν_{in} and ν_{out} are the eddy viscosity for the inner and outer layers, l is the distance to the nearest wall, $|\boldsymbol{u}|$ is the local velocity magnitude, and z is the height of the particular point from the ground. The model uses a number of constants which are set according to the problem: αz_h is the roughness height parameter, the constant b is is the urban morphological parameter, H is the average building height, U_H is the inflow velocity magnitude at the average building height, and z_0 is the aerodynamic roughness length in the atmospheric boundary layer. When $z = 0$, a division by zero can result in the calculation of ν_{out}. In this case, we simply set $\nu_{\text{out}} = 0$. The eddy viscosity calculation for this model is straightforward and computationally inexpensive especially for problems with non-moving boundaries where the wall distance l only needs to be calculated once at the start of the simulation.

At solid walls, it is required that the eddy viscosity is zero. This is dealt with in a natural manner by setting the eddy viscosity to zero inside the solid, or in the case of the domain boundaries, the ghost cell value is set to zero. Then by the harmonic interpolation in Eq. 5.10, the cell face eddy viscosity will also be zero. This means that the eddy viscosity on the AB (See Chapter 4.3) of the immersed boundary will be set to zero, which is not strictly true, since the velocity on the AB is not zero. A more accurate treatment would set the eddy viscosity according to the interpolation in Eq. 5.10, it will then be zero on the immersed boundary surface, and non-zero on the AB. This approximation is made on the grounds that the simplicity of zero-equation turbulence model does not justify the additional complexity introduced by the more accurate treatment at the boundaries. At non-wall boundary conditions (such as inflows, outflows, and symmetry conditions) the eddy viscosity is calculated in the ghost cells directly using Eq. 5.17. All quantities required in Eq. 5.17 are available at the ghost cells from the given boundary condition. The distance to the nearest solid wall is also calculated in the usual way, assuming the physical location of the ghost cell.

5.5 Implementation Details

Due to the simplicity of the zero-equation model, the implementation into an existing CFD code is relatively straightforward, the eddy viscosity simply needs to be recalculated at each outer

iteration, and the diffusive terms updated. The wall distance l may be the distance to the solid geometry or any wall boundary. In the present method, the wall distance is initially calculated to the solid geometry. Then it is checked if there is a domain boundary wall that is closer than this wall distance. The wall distance to the solid geometry is calculated using the AABB tree component of the CGAL C++ library [132], as was done for all immersed boundary method calculations in Chapter 4. While the wall distance calculated can be expensive, especially for large geometries with many triangles, it only needs to be done once at the start of the simulation.

5.6 Numerical Tests

Validation and performance tests were conducted for three different turbulent flow cases using the University of Hamburg CEDVAL experimental reference data set [63]. The three cases are the flow over a 40° angled cube (CEDVAL A1-7), the flow over an array of rectangular blocks (CEDVAL B1-1), and the flow over an intersection (CEDVAL B1-5). The accuracy and performance of the present method with the zero-equation turbulence model introduced in Section 5.4 is also compared the to coupled scheme in ANSYS Fluent with the $k-\varepsilon$ model [93] and the SIMPLE scheme in OpenFOAM with the $k-\varepsilon$ model [94], both using body fitted meshes. Body-fitted grids were used for the $k-\varepsilon$ models to ensure the accurate and consistent application of wall functions and near-wall behaviour, in line with the model formulation. Furthermore, as body-fitted grids remain the dominant approach among practitioners for this class of problems, comparison against them provides a meaningful benchmark relative to current standard practice.

The dataset provides inflow profiles for average velocity and turbulence intensity. Since no additional transport equations are solved with the zero-equation turbulence model, only the average velocity at the inflow is required. The velocity profile is interpolated using linear interpolation onto the mesh. For the ANSYS Fluent and OpenFOAM models, which use the $k-\varepsilon$ model, k and ε need to be specified at the inflow. For k, a profile is created using the following relation [140]

$$k = \frac{3}{2}(I|\boldsymbol{u}|)^2 \tag{5.18}$$

where I is the turbulence intensity. Both I and $|\boldsymbol{u}|$ are available from the measured inflow profile in the dataset. Equation 5.18 is used to create a profile, which is interpolated onto the mesh. Similarly, a profile for ε is created using the definition of the turbulence length scale [140]

$$\varepsilon = C_\mu^{3/4}\frac{k^{3/2}}{L}, \tag{5.19}$$

where L is the turbulence length scale and $C_\mu = 0.09$ is a model constant. The turbulence length scale is estimated from the problem geometry, and is taken as the maximum height of the buildings being simulated.

The model parameters in Eq. 5.17 used in the present method for each test case are given in Table 5.1. The average building height H, the inflow velocity at this height U_H, and the roughness length z_0 were obtained from the dataset. The morphological parameter b and the roughness height parameter αz_H were taken to be the same values used by Liu et al. [170] since no other guidance was provided on how these can be selected or determined.

Table 5.1: Model parameters used for zero-equation model (Eq. 5.17) for each test case.

	Model Parameter				
Test Case	αz_h (m)	z_0 (m)	b	H (m)	U_H (m/s)
CEDVAL A1-7	$0.35 \cdot 0.1H$	0.0004	1.75	0.125	3.43
CEDVAL B1-1	$0.35 \cdot 0.1H$	0.0007	1.75	0.125	4.51
CEDVAL B1-5	$0.35 \cdot 0.1H$	0.0004	1.75	0.06	3.66

For all codes tested, a first order upwind scheme was used for the advective term. This was done for a number of reasons. Firstly, the rationale here is that the zero-equation model in Eq. 5.17 is such a simplistic model that the errors it introduces into the solution are likely much larger than the discretisation errors from the first order scheme. This was confirmed by running the tests for all models with a refined mesh, which showed no effective difference in the errors from the experiment. This indicates that discretisation errors were insignificant relative to modelling errors, and hence using a more accurate advection scheme will not improve accuracy. Secondly, it was found that when higher order advection schemes (such as QUICK or second order upwind) with the OpenFOAM SIMPLE scheme and to a lesser extent the ANSYS Fluent coupled solver and the present method, it was difficult to obtain converged solutions, even with significant amounts of under-relaxation. The additional numerical diffusion from the first order upwind scheme can help to stabilise the solution. After all, the aim here is a fast running model, even if this means making some simplifying approximations which further allow us to achieve this goal. Default model constants and parameters are used for both $k - \varepsilon$ models. Regarding wall functions, it was found that refining the mesh in the wake region to adequately capture the flow resulted in wall y^+ of boundary cells to be less than 30, placing them in the buffer region. Standard wall functions cannot accurately model the flow in this region. To deal with this, wall functions with exponential blending between the viscous sublayer and log-law region were

used [175]. In ANSYS Fluent, this is termed "Enhanced Wall Treatment", and in OpenFOAM this can be set with a blending parameter for the $k - \varepsilon$ wall function.

The problem setup is identical for all three models tested: the domain size, boundary conditions, and geometry. The width and height of the domain was chosen to match the dimensions of the wind tunnel, which is the same for all test cases. The distance upstream and downstream of the geometry was set to be large enough so that the solution is independent of it. This was done through trial and error, and selected such that the resulting errors from the experimental data are independent of the distances. The same is true for the convergence residuals where a residual of 10^{-6} was found to be sufficient. For all cases and models, the geometry was specified by importing an STL file into the code. Both ANSYS Fluent and OpenFOAM models were run using the exact same unstructured mesh. Therefore, any discrepancy in results observed will be due to differences in implementation of the numerical schemes and turbulence models. This mesh was first created in the ANSYS Fluent meshing tool, then exported for use in OpenFOAM. The mesh in the present method and the $k - \varepsilon$ models were chosen to have approximately the same number of total cells and mesh size in the vicinity of the geometry so a fair comparison of simulation time could be made. To compare simulation results with experimental data points, the solution from the simulation is interpolated onto the location of the measured data. For the rectilinear grid used in the present method, trilinear interpolation was used. For the unstructured meshes, interpolation was performed with a Radial Basis Function (RBF) interpolator using linear radial basis functions and the 8 nearest fluid cell centres, using the SciPy Python library [133].

Solver settings for the ANSYS Fluent coupled solver and the OpenFOAM SIMPLE scheme are identical to those used in Chapter 3, but they will be repeated here for completeness. Default settings were used in ANSYS Fluent, and the coupled solver uses Algebraic Multigrid (AMG) to solve the coupled linearised momentum and pressure equations. In the OpenFOAM SIMPLE scheme, Geometric Agglomerated Algebraic Multigrid (GAMG) with a Gauss-Seidel solver was used for the pressure equation, and Preconditioned Bi-Conjugate Gradient Stabilised with a diagonal based incomplete lower upper factorization preconditioner was used for the momentum equations. Only a small number of linear inner iterations were performed at each outer iteration, typically 1 to 3, as this was found to give the best performance. For the present method, F-cycles were used in the FAS scheme, although W-cycles were found to give essentially the same performance. In all cases, 2 pre and post-smoothing outer iterations were performed at each grid level, and 5 fine grid iterations were performed. A residual tolerance of 10^{-10} was used on the coarsest grid, with a maximum number of 100 iterations. The number of coarse grid levels was

chosen to be the largest number of levels that would allow for a stable solution. This was problem dependent and is given individually for each problem. The same is true for the under-relaxation factors used for all schemes tested here. These were determined through trial and error.

All tests were run on System 1 in Table 3.4 using 8 processors to ensure a fair comparison of run time. Since multiple processors were used, differences in performance observed come from a range of factors including: convergence rate of the solver, the efficiency of the implementation, and the parallel efficiency of the method.

5.6.1 40° Angled cube (CEDVAL A1-7)

The first case considered is a surface mounted cube angled at 40° relative to the inflow direction (CEDVAL A1-7). The domain setup is shown in Figure 5.1. The non-uniform rectilinear mesh used for the present method is shown in Figure 5.2. 3 coarse levels were used in the FAS scheme (meaning 4 grid levels total). Under-relaxation factors of 0.85 on the momentum equations only were used for the present method, while 0.7 was used in the ANSYS Fluent coupled solver. No under-relaxation was used on the k and ε equations. For the OpenFOAM SIMPLE scheme, under-relaxation of 0.2 was used for pressure and 0.5 for the momentum, k, and ε equations. For both the present method and the $k-\varepsilon$ models, a mesh of $\sim 1.06 \times 10^6$ cells was used. For the $k-\varepsilon$ models, an unstructured tetrahedral mesh with two inflation layers on all solid walls is used. The mesh was refined in the region near the block.

Comparison of the three models tested with the wind tunnel data of the x velocity profiles along a vertical and horizontal plane are shown in Figure 5.3. Qualitatively, agreement of the present method with experiment is good. The approaching flow and the flow past the block is well modelled, with the present method marginally over predicting the velocity, while the $k-\varepsilon$ models slightly under predict it. This is also true in the free stream in the region directly above the block. Along the top face of the block adjacent to the wall, it is clear that the flow is not well predicted by the zero-equation model. This is somewhat expected since no wall functions were used. On the other hand, the $k-\varepsilon$ models do a reasonable job of predicting the velocity in these regions due to the use of wall functions. In the wake region, the vertical profiles (Figure 5.3a) indicate that both models struggle to predict the recirculation zone, with the $k-\varepsilon$ models doing a better job. Interestingly however, the off–centre profiles in Figure 5.3b show better agreement with the zero-equation model.

Velocity magnitude contour plots for all three models are shown in Figure 5.4. In general the same differences described above between the two models can be seen. Above the block in the free stream, the velocity has a greater magnitude in the zero-equation model compared to

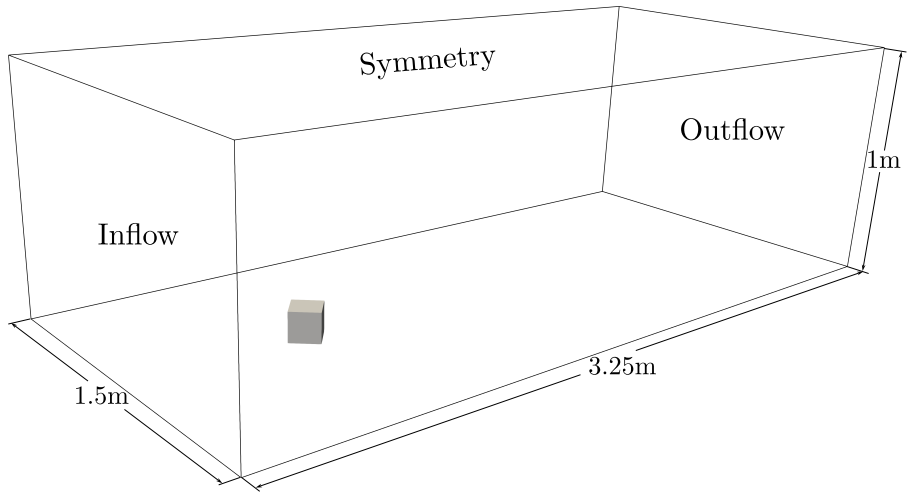
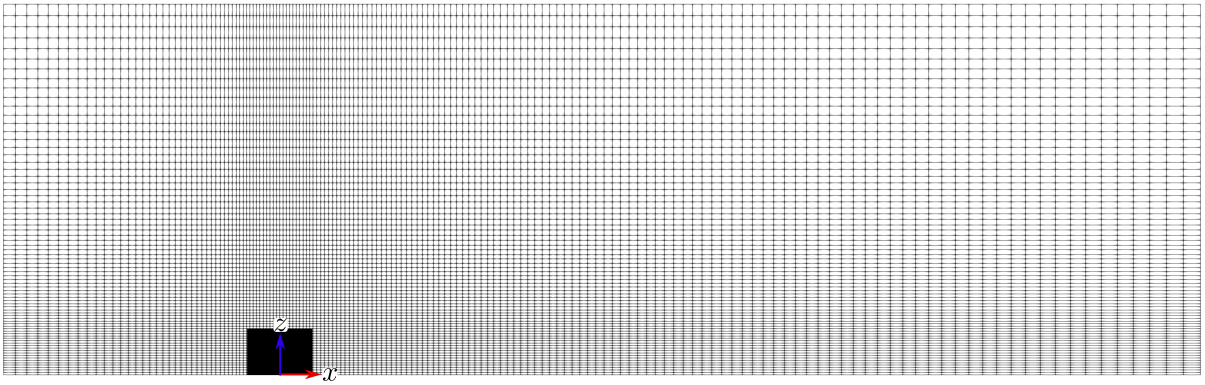
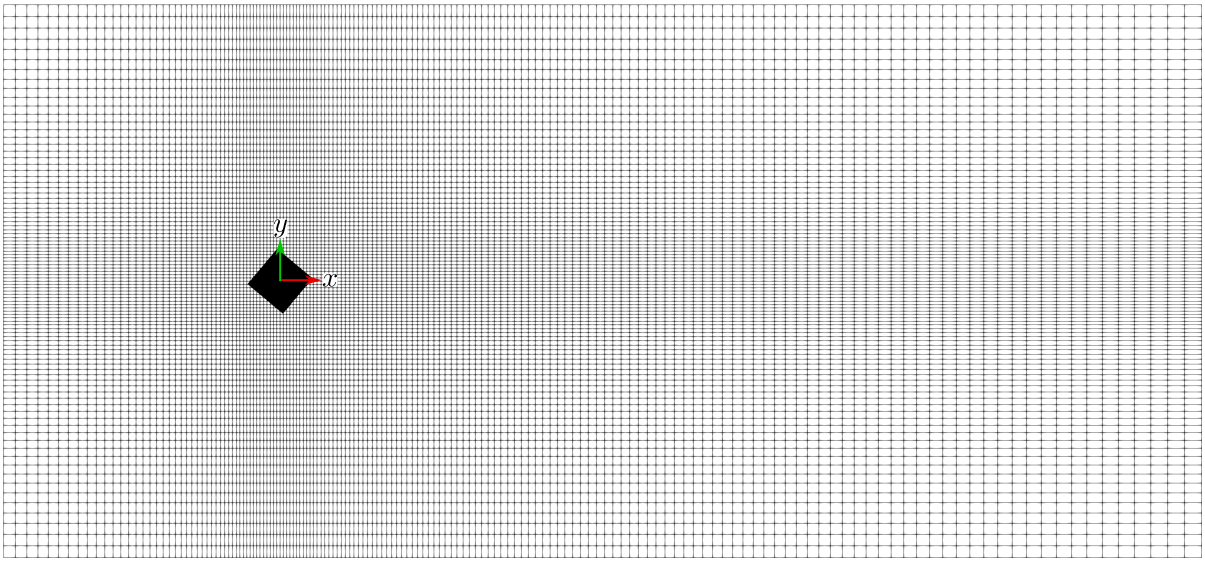


Figure 5.1: Problem setup for wind tunnel 40° angled cube case (CEDVAL A1-7). The cube has side length 0.125 m, and all domain boundaries except for the inflow, symmetry, and outflow as marked are walls.



(a)



(b)

Figure 5.2: Side (a) and top (b) view of computational mesh used for 40° angled cube case (CEDVAL A1-7) in present method with $\sim 1.06 \times 10^6$ cells.

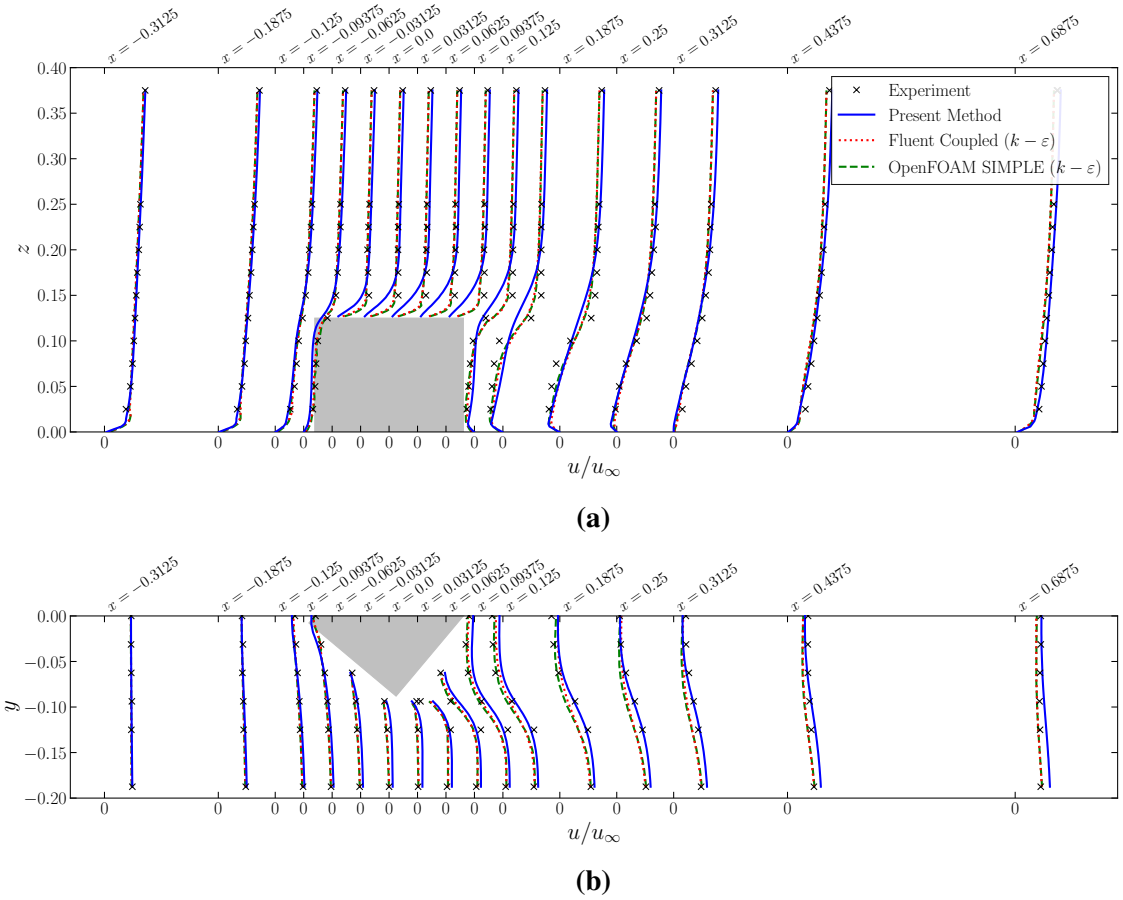


Figure 5.3: Comparison of velocity profiles with experiment for the $40°$ angled cube case (CEDVAL A1-7) in the $x - z$ plane with $y = 0$ m (a) and $x - y$ plane with $z = 0.05$ m (b).

the $k - \varepsilon$ model. This is consistent with the previous observation that the zero-equation model over predicted the free stream velocity compared to the experiment, while the $k - \varepsilon$ model under predicted it. The same can be seen regarding the boundary layers near the walls of the block in the zero-equation model, which are not well resolved. The contours also show that the recirculation length is slightly larger for the zero-equation model. This is in contradiction with the results by Liu et al. [170] which showed that the $k - \varepsilon$ model predicted a significantly larger recirculation length from a cube that was not at an angle to the flow, as is the case here. Pressure contour plots for the same plane are shown for all three models are Figure 5.5. Unfortunately, no pressure data is available from dataset. Both the $k - \varepsilon$ models have very similar pressure fields around the block, but differ greatly from the present method. Qualitatively, they share similar features, with a high pressure region ahead of the block, and lower pressure above the block and downstream in the recirculation region. However the pressure range and magnitude varies greatly, despite the same boundary conditions being used in all models – zero gradient for the pressure on all boundaries except the outlet, which was fixed at zero.

Comparison of all experimental data points with each simulation is shown in Figure 5.6. In

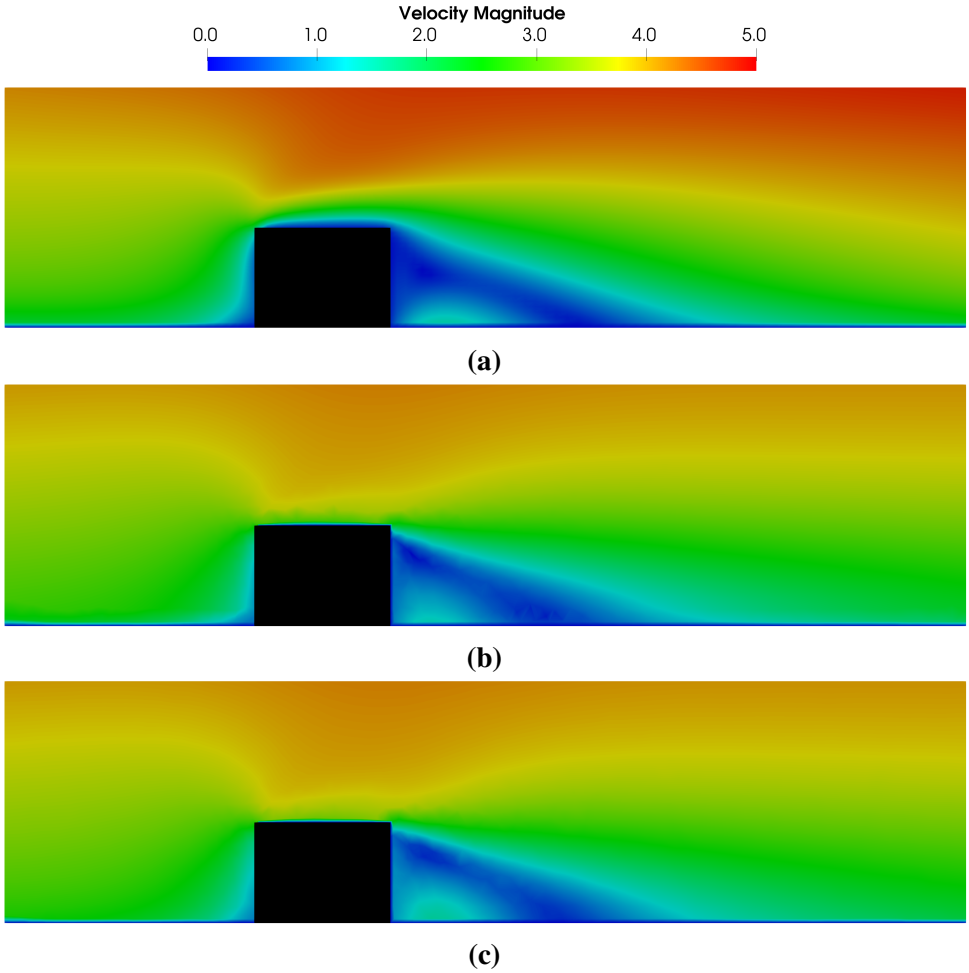


Figure 5.4: Velocity magnitude in the $x-z$ plane for the 40° angled cube case (CEDVAL A1-7) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k-\varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k-\varepsilon$) (c).

the figure, points which lie on the diagonal are in perfect agreement with experiment. Also shown are the bounds for 10% and 30% error. For all models tested, it appears that the majority of points lie within 30% error. A very good result, especially for the zero-equation model. In general from Figure 5.6, it appears that the zero-equation model is over predicting the velocity, while the $k-\varepsilon$ models are under predicting the velocity. This is in line with the observations above. Interestingly, there is a region in Figure 5.6 where $|u|_{\text{experiment}} \gtrsim 3.5$ where all models under predict the velocity. These points correspond to the region immediately above the block, but away from the centre line at $y = 0$. Table 5.2 shows the average errors of all the data points for each of the models, as well as the solver run time to calculate the solution. Note that for the present method, the sweep directions are chosen such that the optimal convergence rate is given. This will be discussed in more depth below. The results in Table 5.2 show that in terms of accuracy, the zero-equation model is competitive with the $k-\varepsilon$ for this problem, especially considering that no wall functions are used. Importantly, the present method with the

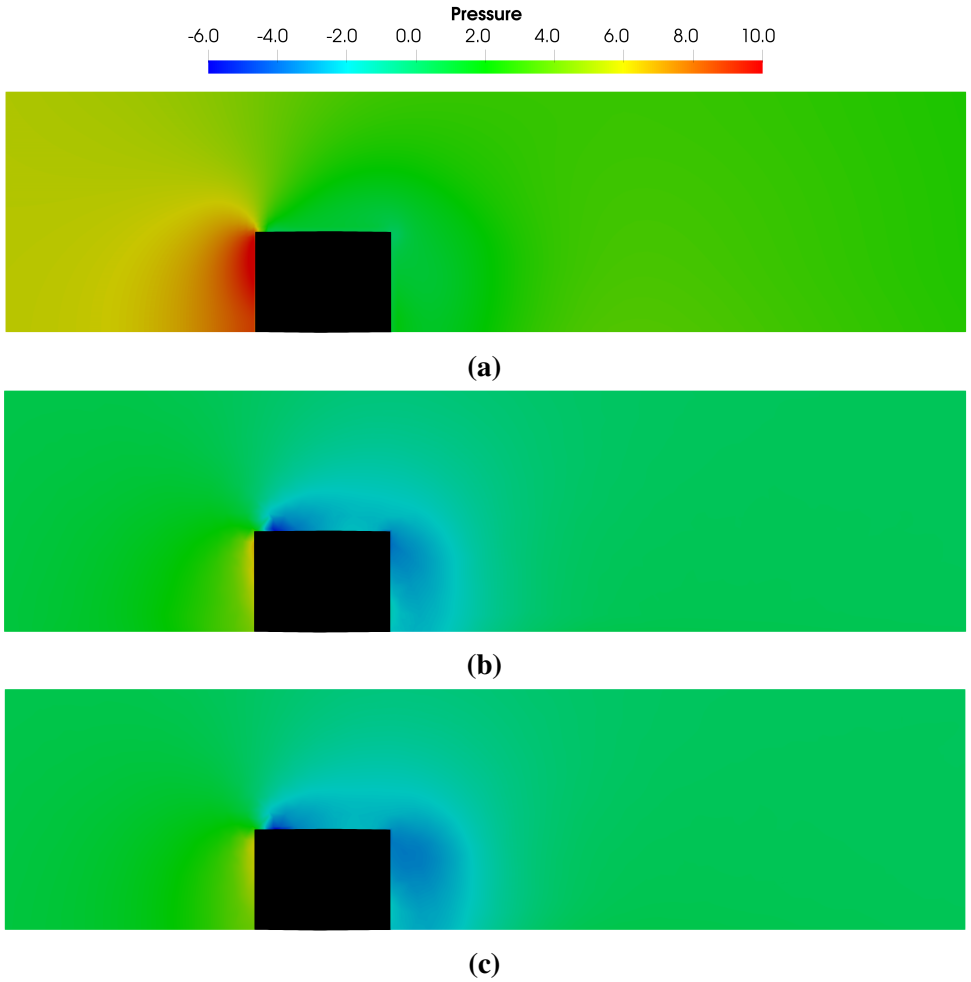


Figure 5.5: Pressure in the $x - z$ plane for the $40°$ angled cube case (CEDVAL A1-7) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c).

zero-equation model solves the problem significantly faster that the $k - \varepsilon$ models, being about $72\times$ faster than the coupled solver in ANSYS Fluent, and about $86\times$ faster than the SIMPLE scheme in OpenFOAM.

Also to note in the results so far is the difference in results between the $k - \varepsilon$ models in the ANSYS Fluent Coupled scheme and the OpenFOAM SIMPLE scheme. In principle, both are solving the same set of equations, however small differences in implementation will result in different solutions. These differences likely lie within the formulation of the turbulence models used and the wall functions employed. In any case, it is not within the scope of this work to assess the differences in implementation of the different codes.

As was done in Chapter 3, the effect of different line and plane sweeping directions on convergence rate is examined here. The solution obtained by each sweeping direction was identical, and the results above were obtained using Plane z, Line y sweeping direction. The problem was solved for each possible direction to a residual of 10^{-10}. The number of multigrid

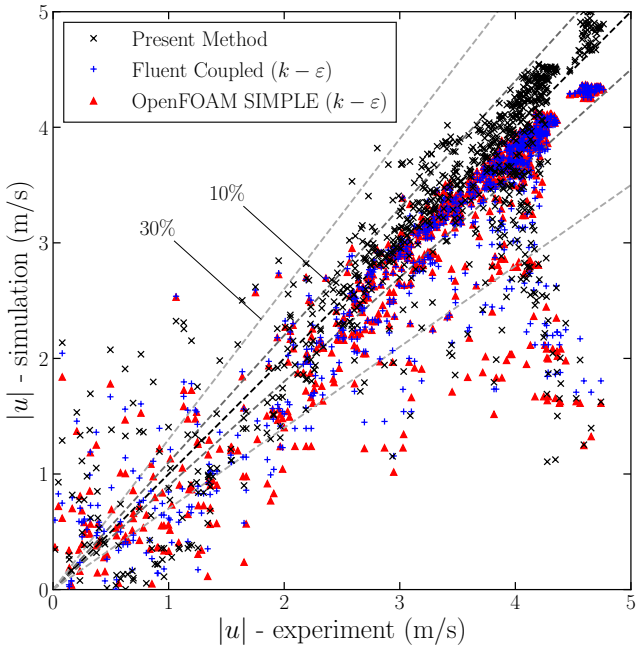


Figure 5.6: Comparison of all experimental data points with methods tested for the 40° angled cube case (CEDVAL A1-7). Points with perfect agreement with experiment lie on the diagonal line. Also indicated lines which show the boundary of 10% and 30% error.

Table 5.2: Average error from all experimental data points of each method tested and solver time to reach a residual of 10^{-6} with $\sim 1.06 \times 10^6$ cells for the 40° angled cube case (CEDVAL A1-7).

Method	**Average error (%)**	**Solver time (s)**
Present Method	29.60	19.81
ANSYS Fluent Coupled ($k-\varepsilon$)	28.68	1419.15
OpenFOAM SIMPLE ($k-\varepsilon$)	25.61	1616.19

cycles to reach the target residual and the total solver times are shown in Table 5.3. In Chapter 3 it was shown that for the 3D laminar lid driven cavity and the 3D backwards facing step, the choice of sweep direction had little effect on convergence rate. Furthermore, for the 3D backwards facing step, it was found that some sweeping directions were unstable. For the turbulent flow case here, the results are much the same, with 4 of the 6 sweeping directions essentially giving the same convergence rate. One sweeping direction (Plane x, Line y) took significantly more cycles to converge however, while another sweeping direction (Plane y, Line z) stalled at a residual of 10^{-2}. This residual is too high to provide an accurate steady solution. In terms of the geometry of the problem and the dominant flow direction, there does not seem to be an intuitive reasoning why the particular directions observed to give poor convergence are what they are.

This was also found to be the case for the 3D backwards facing step. Furthermore, while the plane y, line z direction was found to be non-convergent for both the 3D backwards facing step and the 40° angled cube, the other non-convergent direction for the 3D backwards facing step (Plane z, Line x) converged well in this case.

Table 5.3: Number of multigrid cycles and solver time for present method to reach a residual of 10^{-10} for the 40° angled cube case (CEDVAL A1-7) with each plane and line update direction.

Sweep Direction	Number of Cycles	Solver time (s)
Plane x, Line y	29	73.04
Plane x, Line z	15	36.50
Plane y, Line x	15	36.71
Plane y, Line z	Stalled at residual $\sim 10^{-2}$	
Plane z, Line x	14	34.37
Plane z, Line y	15	36.21

5.6.2 Array of Rectangular Blocks (CEDVAL B1-1)

The second case considered is a regular array of surface mounted rectangular buildings (CEDVAL B1-1). The domain setup is shown in Figure 5.7, with the non-uniform rectilinear mesh used for the present method shown in Figure 5.8. For the present method 2 coarse grid levels were used in the FAS scheme (meaning 3 grid levels total). Using any more coarse levels led to instability in the solution. Under-relaxation factors of 0.8 were used on the momentum equations only for the present method, while 0.7 was used for the ANSYS Fluent coupled solver for the momentum equations, and no under-relaxation for other variables. For the OpenFOAM SIMPLE scheme, under relaxation of 0.2 was used on the pressure, and 0.5 for the momentum, k, and ε equations. All meshes contained $\sim 2.06 \times 10^6$ cells, while for the $k-\varepsilon$ models, an unstructured tetrahedral mesh with two inflation layers on all solid walls was used.

Velocity profiles along an off-centre vertical plane and horizontal plane are shown in Figure 5.9. Starting with the vertical plane (Figure 5.9a), both $k-\varepsilon$ models do a good job of predicting the velocity profiles above the blocks, with the tendency to under-predict the velocity, particularly for the OpenFOAM model. The zero-equation model on the other hand slightly over-predicts the velocity in this region, which was also the case in 40° cube case. Adjacent to the wall on top of the blocks, the $k-\varepsilon$ models do a better job of predicting the flow, likely due to the presence of wall functions. Unlike the 40° cube case where the present method grossly under predicted these regions, here they do a reasonable job, especially above the block located

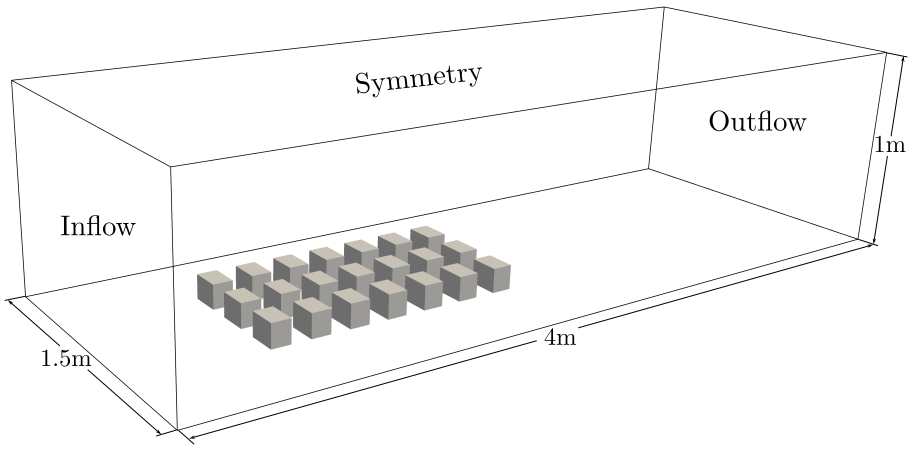
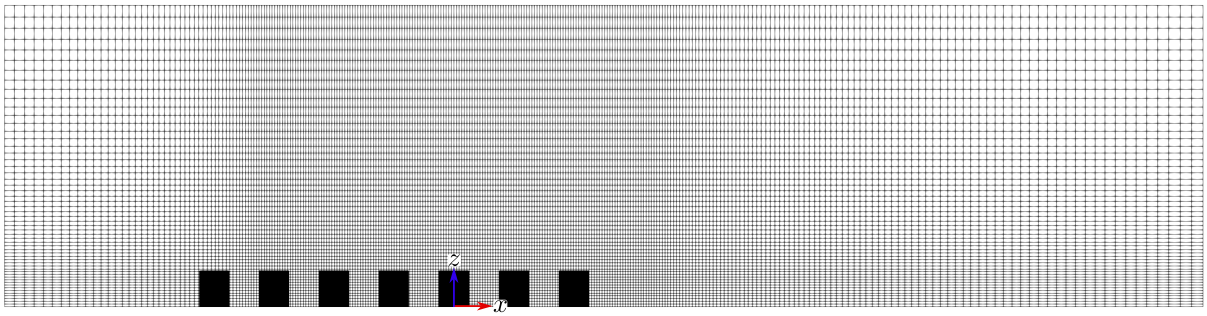
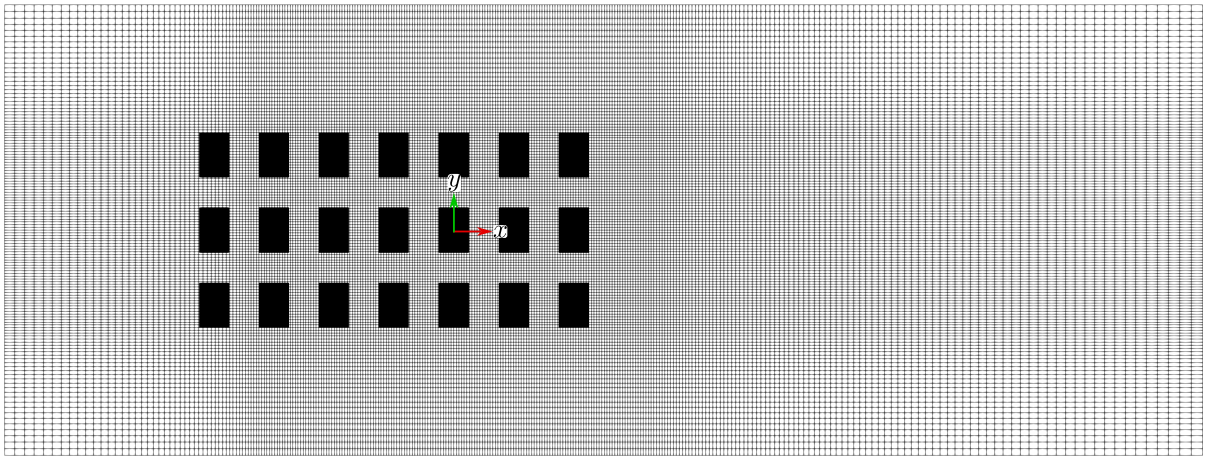


Figure 5.7: Problem setup for wind tunnel array of rectangular blocks case (CEDVAL B1-1). Each block has dimensions 0.1 m $\times$ 0.15 m $\times$ 0.125 m, and all domain boundaries except for the inflow, symmetry, and outflow as marked are walls.

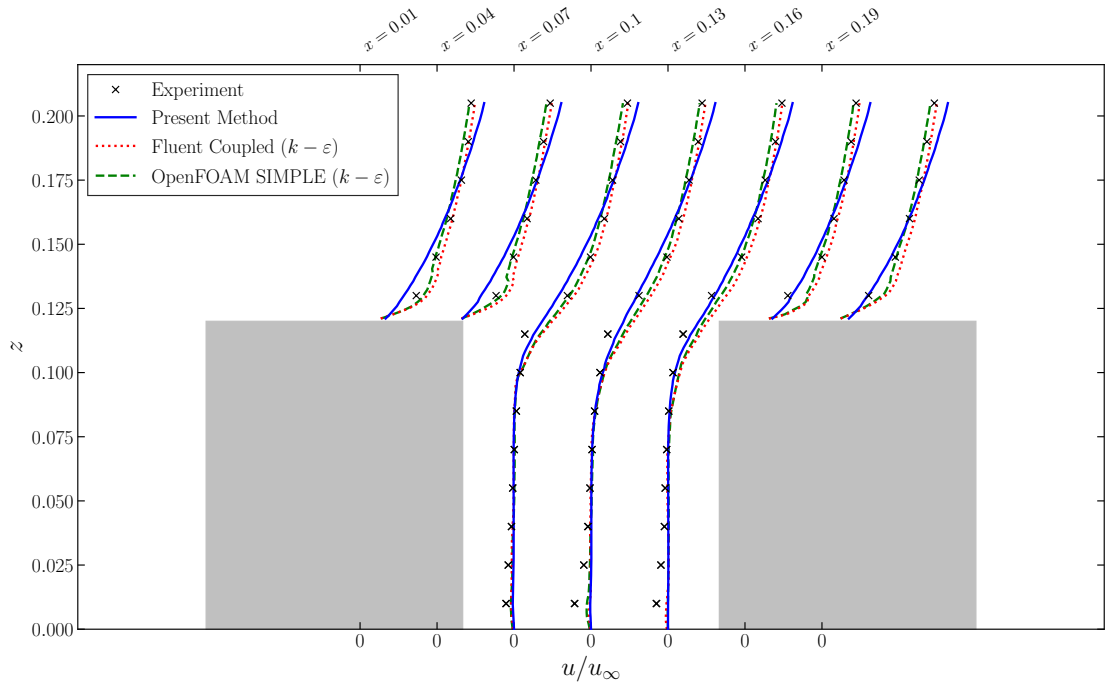


(a)

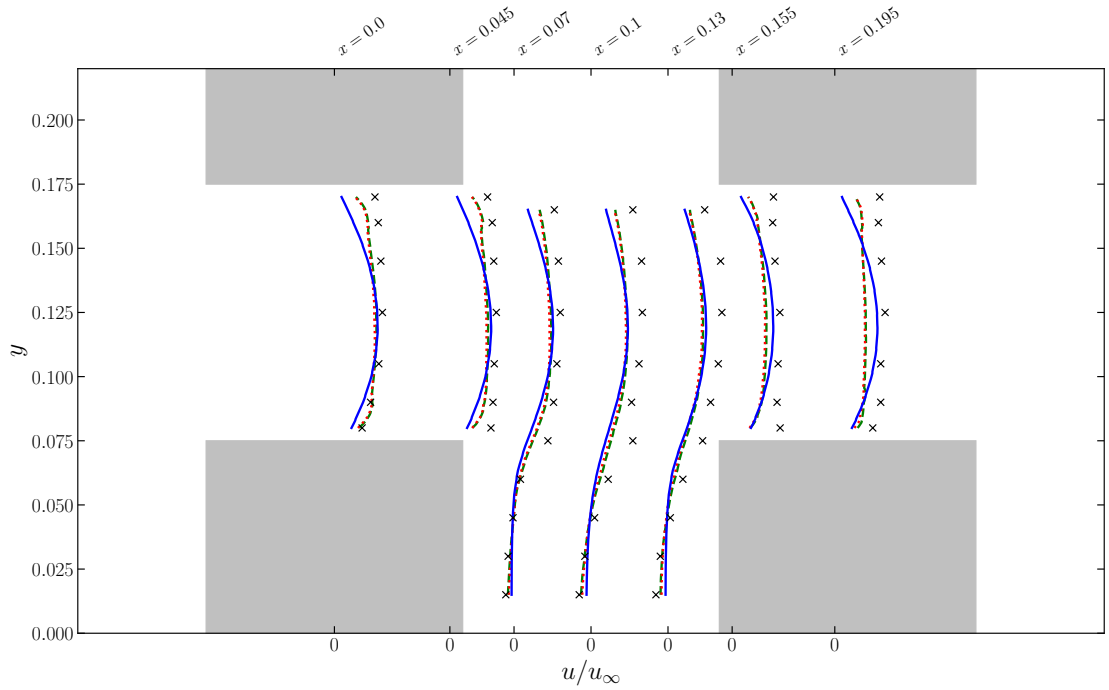


(b)

Figure 5.8: Side (a) and top (b) view of computational mesh used for the array of rectangular blocks case (CEDVAL B1-1) in present method with $\sim 2.06 \times 10^6$ cells.



(a)



(b)

Figure 5.9: Comparison of velocity profiles with experiment for the array of rectangular blocks case (CEDVAL B1-1) in the $x - z$ plane with $y = -0.05$ m (a) and $x - y$ plane with $z = 0.0625$ m (b).

downstream. In between the blocks, it seems that the zero-equation model has trouble capturing the recirculation, and while the $k-\varepsilon$ models do a better job of this, the velocity magnitude is still under-predicted, particularly towards the bottom of the cavity.

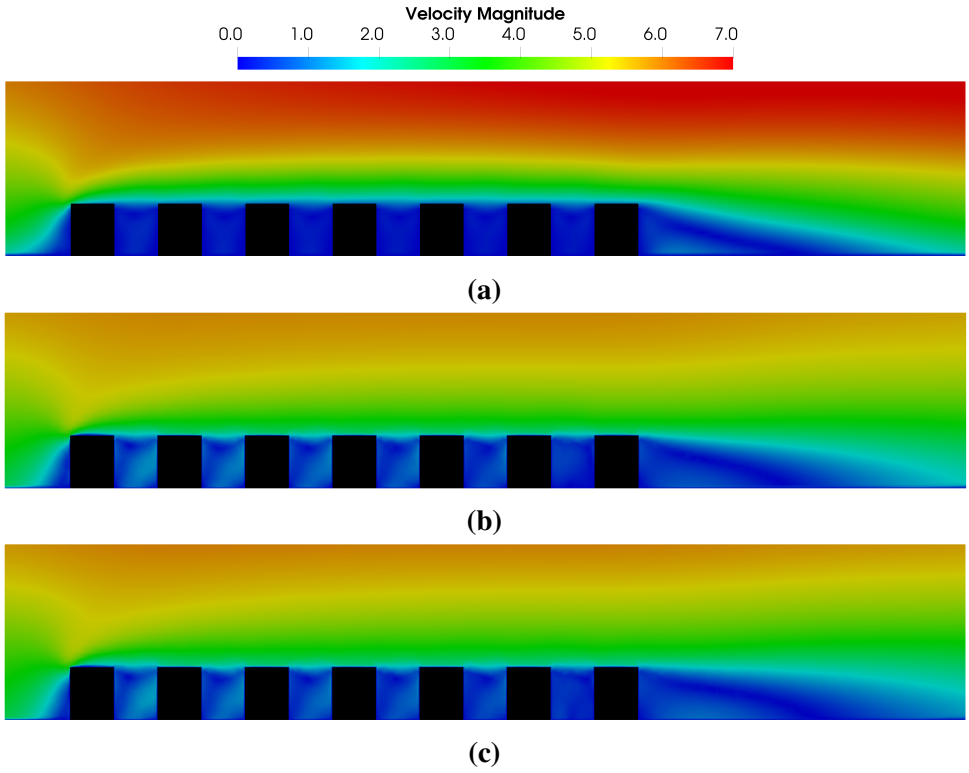


Figure 5.10: Velocity magnitude in the $x-z$ plane for the array of rectangular blocks case (CEDVAL B1-1) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k-\varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k-\varepsilon$) (c).

For the horizontal plane in Figure 5.9b, the predictions of the zero-equation model are on par with the $k-\varepsilon$ models and in some cases better. However near the walls, the zero-equation model fails to predict the steep velocity gradients. While the $k-\varepsilon$ models don't necessarily do a great job of this either, they are closer to the experimental data, again likely due to the presence of wall functions.

The velocity contours are shown in Figure 5.10. Consistent with the velocity profiles, the zero-equation model predicts a greater velocity magnitude above the blocks in the far field compared to the $k-\varepsilon$ models. Furthermore, it can be seen that the zero-equation model does not effectively capture the recirculated flow between the blocks - the flow is relatively stagnant. It also appears that behind the last block, the shape of the recirculation zone predicted by the zero-equation model is qualitatively different to the $k-\varepsilon$ models. Unfortunately, no experimental measurements in this region are available within the dataset to make a valid assessment of which is more accurate. The pressure fields shown in Figure 5.11 show qualitatively the same relative behaviour between models that was seen for the 40° angled cube case in Section 5.6.1.

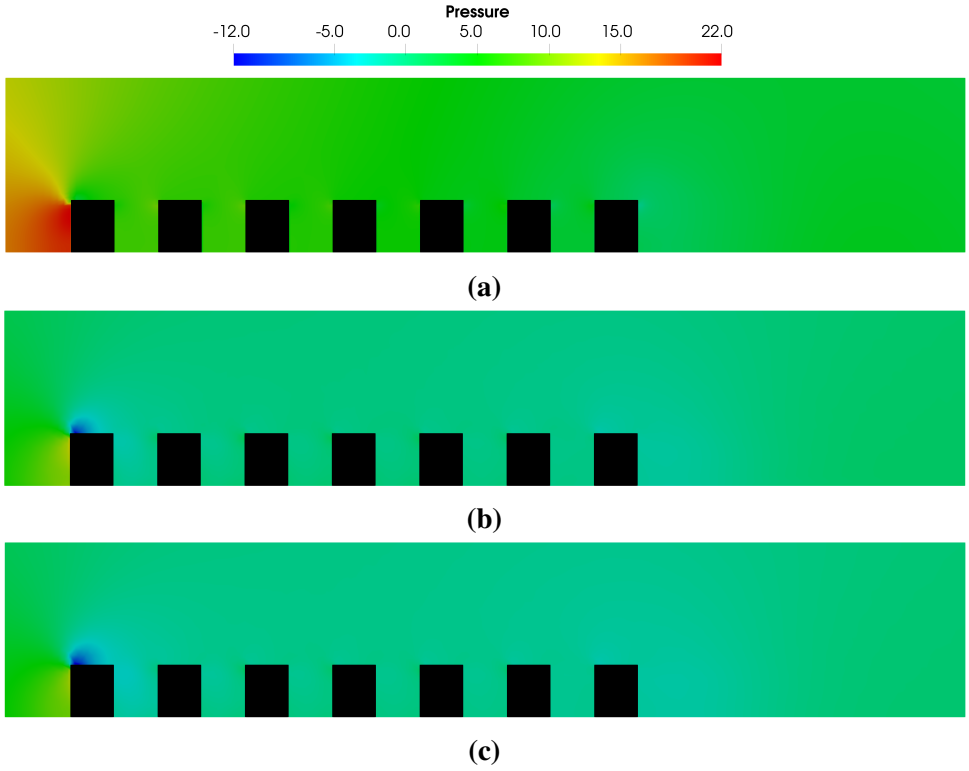


Figure 5.11: Pressure in the $x - z$ plane for the array of rectangular blocks case (CEDVAL B1-1) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c).

The overall comparison with the experimental data in Figure 5.12 and the average errors in Table 5.2 indicate very similar results to the 40° cube case, with the average errors of the zero equation model only being a few percent greater than the $k - \varepsilon$ models. For this problem, the speed-up of the present method is even more significant, about $126\times$ faster than the ANSYS Fluent Coupled method, and about $145\times$ faster than the OpenFOAM SIMPLE scheme. The speed-up of the coupled solvers against the segregated SIMPLE solver is an expected result that was explored in Chapter 3. However of note is that the relative speed-up of the present method for this case is even more significant than what was observed for the $40°$ angled cube case in Section 5.6.1. This is largely attributed to the better scaling with problem size of the FAS scheme. The number of cells used in this problem is double what was used in the $40°$ angled cube case in Section 5.6.1. For the present method, the solver time also roughly doubled – consistent with linear scaling with problem size that was demonstrated in Chapters 3 and 4. However, the solver time for the other two methods was more than double.

The sweep direction results in Table 5.2 show the same qualitative behaviour that was observed for the 40° angled cube case: The plane x, line y direction shows a much slower convergence rate, while the plane y, line z direction stalled. In this case though, the convergence stalled at a much lower residual of 10^{-6} – small enough to deem acceptable for a steady solution.

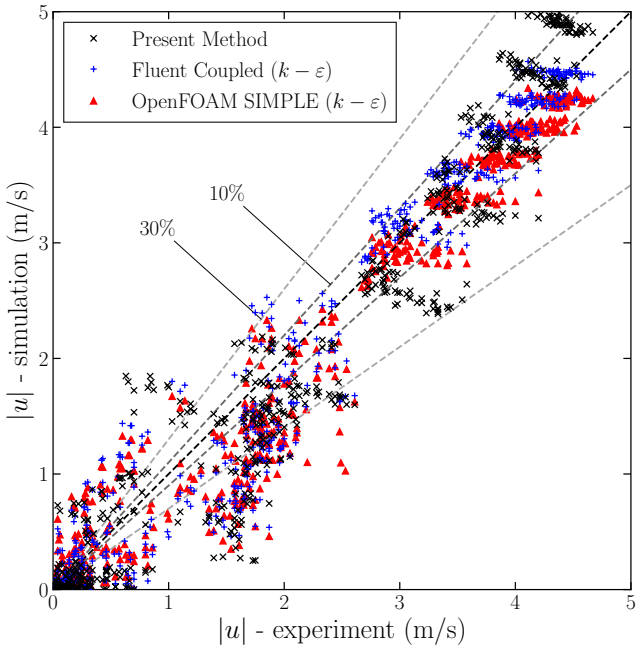


Figure 5.12: Comparison of all experimental data points with methods tested for the array of rectangular blocks case (CEDVAL B1-1). Points with perfect agreement with experiment lie on the diagonal line. Also indicated lines which show the boundary of 10% and 30% error.

Table 5.4: Average error from all experimental data points of each method tested and solver time to reach a residual of 10^{-6} with $\sim 2.06 \times 10^6$ cells for the array of rectangular blocks case (CEDVAL B1-1).

Method	Average error (%)	Solver time (s)
Present Method	40.51	40.93
ANSYS Fluent Coupled ($k-\varepsilon$)	38.29	5152.87
OpenFOAM SIMPLE ($k-\varepsilon$)	39.65	5943.43

5.6.3 Flow Across an Intersection (CEDVAL B1-5)

The final test case is the flow over an intersection (CEDVAL B1-5). The problem geometry and domain setup is shown in Figure 5.13. The geometry is an intersection made up of four buildings which are square "rings" with a combination of flat and pitched roofs. The buildings are rotated at an angle of 29° relative to the inflow direction. The non-uniform rectilinear mesh used for the present method is shown in Figure 5.14. In the $x-y$ plane, two coordinate axis are considered. The x', y' is the coordinate axis aligned with the building geometry and is used by the dataset. These are the axis along which experimental measurements were taken, and will be used here for visualising results. The x, y axis is the coordinate axis aligned with the inflow and mesh, and is what was used in the solver. The results given for the effect of sweep direction are given in terms

Table 5.5: Number of multigrid cycles and solver time for present method to reach a residual of 10^{-10} for the array of rectangular blocks case (CEDVAL B1-1) with each plane and line update direction.

Sweep Direction	Number of Cycles	Solver time (s)
Plane x, Line y	40	284.31
Plane x, Line z	12	94.47
Plane y, Line x	13	98.72
Plane y, Line z	Stalled at residual $\sim 10^{-6}$	
Plane z, Line x	15	100.16
Plane z, Line y	15	105.17

of the x, y plane. For the present method, 2 coarse grid levels were used in the FAS scheme (meaning 3 grid levels total). Under-relaxation of 0.9 was used on the momentum equations only for the present method, while 0.7 was used only on the momentum equations for the ANSYS Fluent coupled scheme, and for the SIMPLE scheme, 0.2 was used for the pressure, and 0.5 for the momentum, k and ε equations. For all methods, a mesh with 3.14×10^6 cells was used.

Velocity profiles for two vertical planes (in the x', y' axis) are given in Figure 5.15. Immediately, it is obvious that for this more complicated case, both models struggle to make predictions at the same level of accuracy that was observed in the previous two cases. Often, both models completely mispredict the velocity profile at a qualitative level, such as for example, between the buildings at $x' = 0$. Overall however, it is clear that the $k - \varepsilon$ models are generally able to do a better job. In fact, there are regions where the two models predict very different velocity distributions, particularly in Figure 5.15b where the flow is recirculating. Many times, the zero-equation model fails to capture the recirculating flow that appears in building canyons (near $x' = 0$ in Figure 5.15) and in "courtyard" regions of the buildings ($x' > 0.09$ and $x' < -0.09$ in Figure 5.15).

The velocity profiles on two horizontal planes (in the x', y' axis) are shown in Figure 5.16. In these profiles, good agreement is observed with the $k - \varepsilon$ model, however the zero-equation model struggles to predict the channelling in the street canyons, where the velocity is under predicted. Though, outside the street canyons, around $y' = -0.3$ and $y' = 0.3$, both models do a reasonable job.

The velocity contours in Figure 5.17 and pressure contours in Figure 5.18 show much of the same differences between the two turbulence models that were observed in the previous two tests cases. In part3cular, the zero-equation model predicts a higher velocity magnitude above the buildings near the free stream than the $k - \varepsilon$ model. For the previous two cases, the

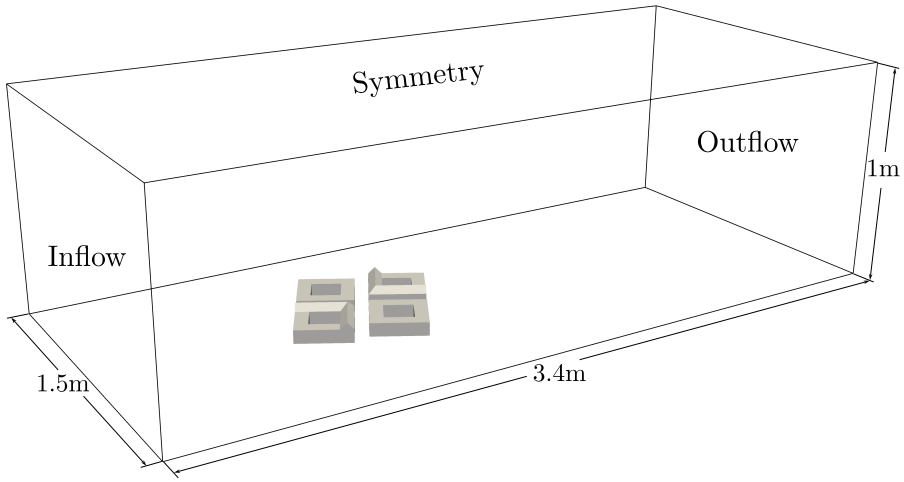
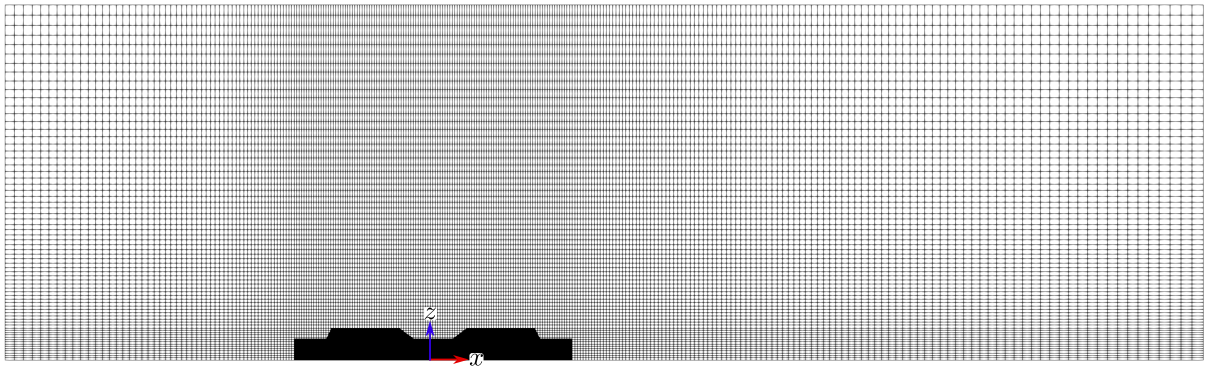
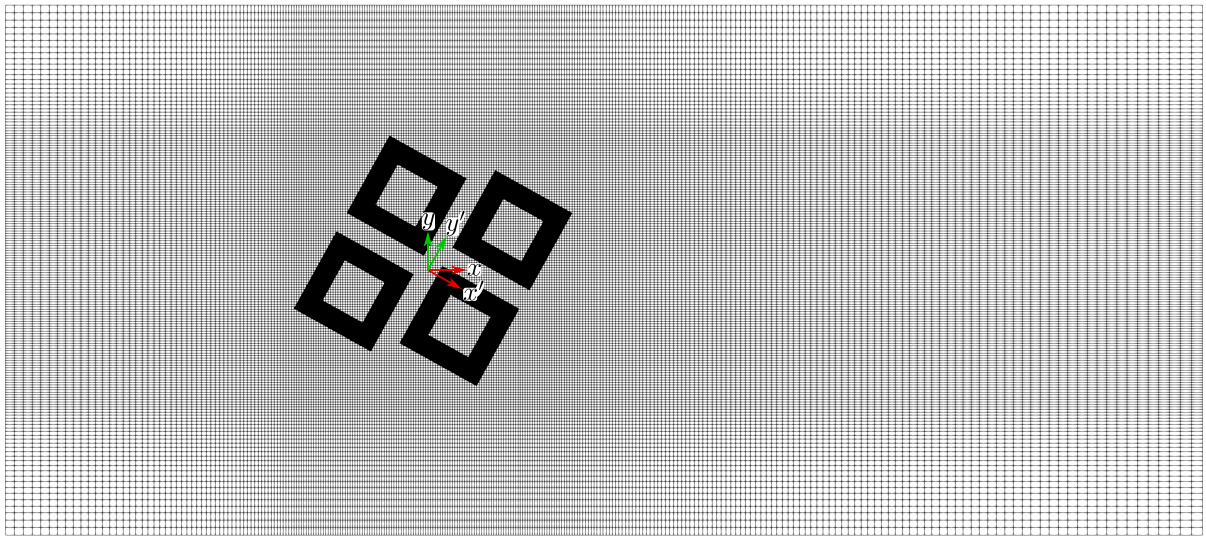


Figure 5.13: Problem setup for wind tunnel flow over an intersection case (CEDVAL B1-7). The buildings have a height of 0.06 m without the pitched roof, and 0.09 m with the pitched roof, and all domain boundaries except for the inflow, symmetry, and outflow as marked are walls. The buildings are rotated 29° relative to the inflow direction.



(a)



(b)

Figure 5.14: Side (a) and top (b) view of computational mesh used for the flow over an intersection case (CEDVAL B1-5) in present method with $\sim 3.15 \times 10^6$ cells. The (x, y) axis is used in the simulation and is aligned with the inflow direction, while the (x', y') axis is aligned with the geometry.

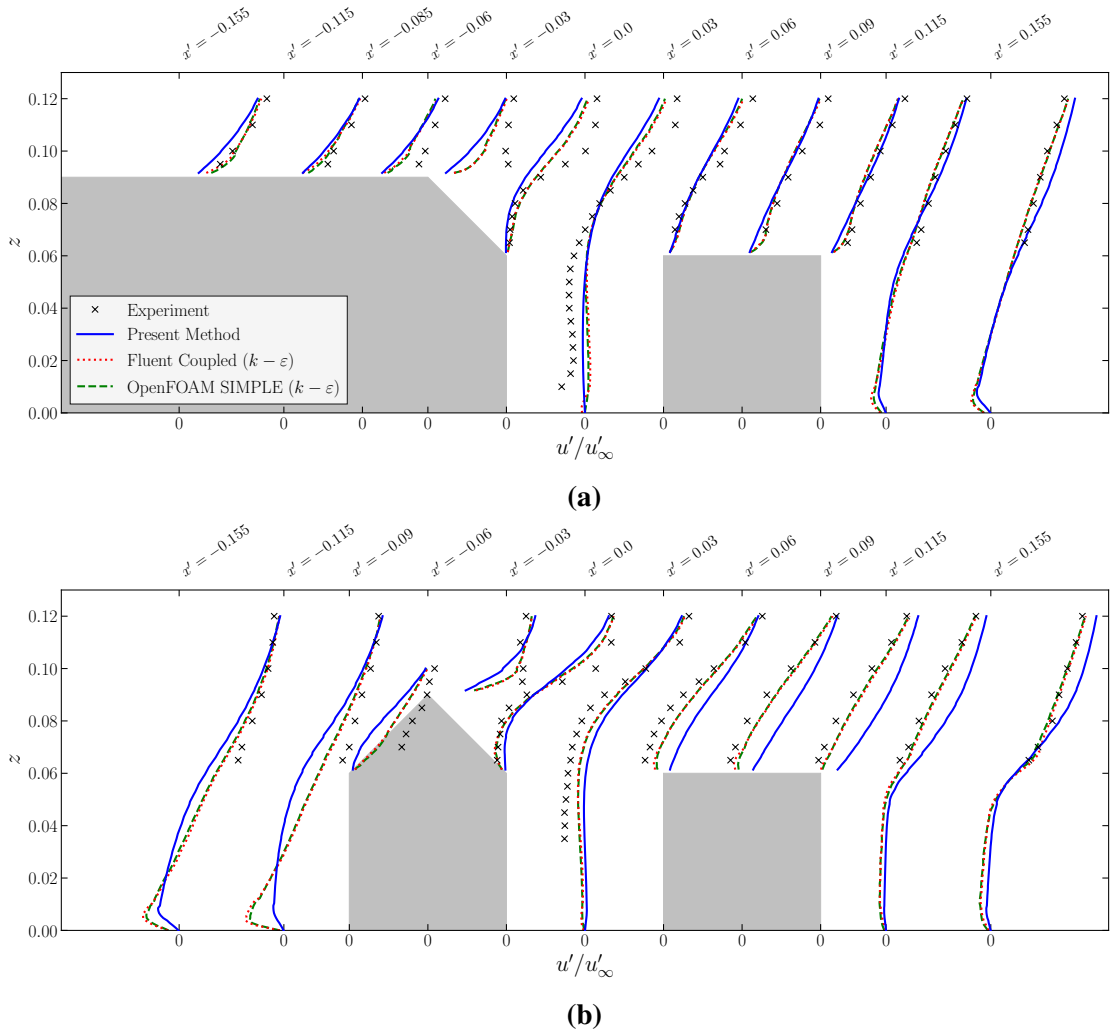


Figure 5.15: Comparison of velocity profiles with experiment for the flow over an intersection case (CEDVAL B1-5)in the $x' - z$ plane with $y' = -0.06$ m (a) $y' = -0.155$ m (b). u' is the velocity component in the x' direction.

zero-equation model slightly over predicted this velocity, while the $k - \varepsilon$ under predicted it in both cases, especially in the array of blocks case where it was significantly under predicted (Figure 5.9). Unfortunately, experimental measurements for this region of the flow are not available in the dataset, so it cannot be concluded which model is more accurate in this region. The velocity magnitudes in the regions amongst the buildings further affirm the results found in Figure 5.15 and 5.16: the zero-equation model under predicts the velocity magnitude, and much of the recirculation between the buildings is not correctly captured.

The comparison with all experimental data points in Figure 5.19 shows that most zero-equation model predictions have a greater than 30% error, with most of the velocities being significantly under-predicted. This is in contrast with the previous two test cases were the zero-equation model had the tendency to over predict velocities. This may be due to most of the experimental measurements being between the buildings, and none above the buildings near the

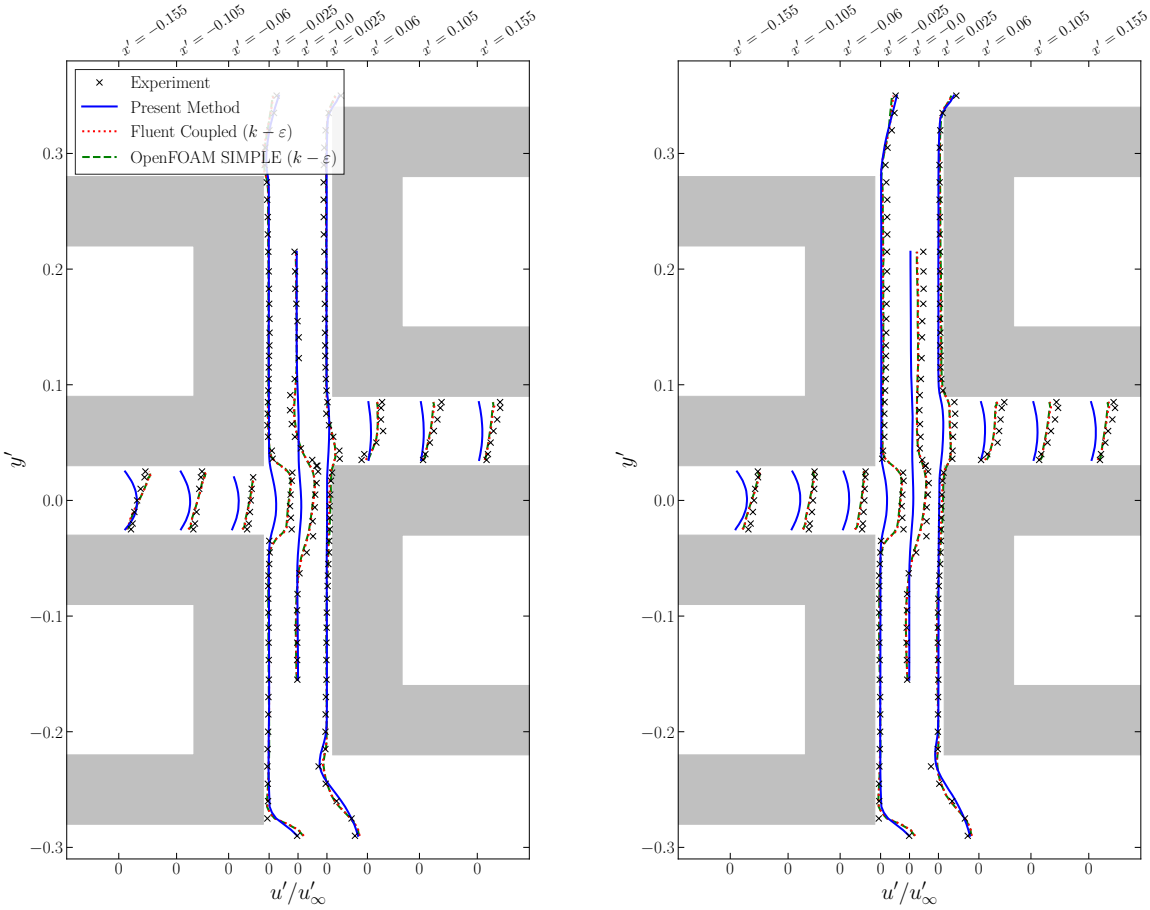


Figure 5.16: Comparison of velocity profiles with experiment for the flow over an intersection case (CEDVAL B1-5) in the $x' - y'$ plane with $z = 0.03$ m (a) $z = 0.05$ m (b). u' is the velocity components in the x' direction.

freestream. On the other hand, the results for the $k - \varepsilon$ models seems to relatively consistent with what was observed previously, with a significant fraction of the data being within 30% error. The average error of the zero-equation model from the experimental data was calculated to be approximately 25% greater than the $k - \varepsilon$ models (Table 5.6). While this is quite significant, the time to calculate the solution must be taken into account. The present method with the zero-equation model was was about $205\times$ faster than the coupled solver in ANSYS Fluent, and about $315\times$ faster than the SIMPLE scheme in OpenFOAM. The loss in accuracy of the model is the trade-off that is made to obtain such fast solutions.

Finally, the effect of sweeping direction is given in Table 5.7. There are some differences to the previous cases here that should be noted. Firstly, the plane x, line y direction stalled at a residual of 10^{-6}, whereas in other cases tested, this sweeping direction reached a residual of 10^{-10}, albeit at a much slower convergence rate. Also, unlike the previous cases, the plane y, line x had a much slower convergence rate. As before however, the plane y, line z direction also

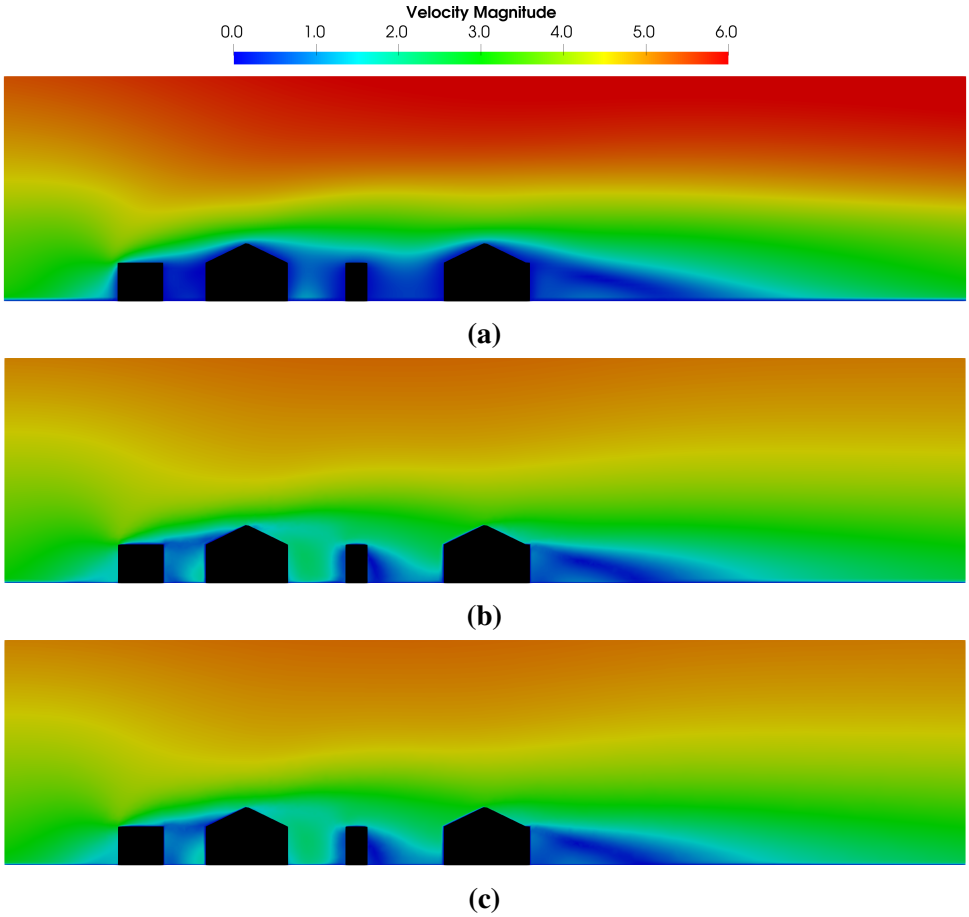


Figure 5.17: Velocity magnitude in the $x - z$ plane for the flow over an intersection case (CEDVAL B1-5) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c).

Table 5.6: Average error from all experimental data points of each method tested and solver time to reach a residual of 10^{-6} with $\sim 3.15 \times 10^6$ cells for the flow over an intersection case (CEDVAL B1-7).

Method	Average error (%)	Solver time (s)
Present Method	74.89	41.88
ANSYS Fluent Coupled ($k - \varepsilon$)	48.21	8600.53
OpenFOAM SIMPLE ($k - \varepsilon$)	50.11	13193.90

stalled, but at a relativity smaller residual of 10^{-8} – small enough to be considered converged. As has been the cases throughout this chapter and Chapter 3, it is difficult to put together an argument for what influences the convergence with different sweep directions, let alone make predictions before calculating the solution on which sweep direction will provide optimal convergence.

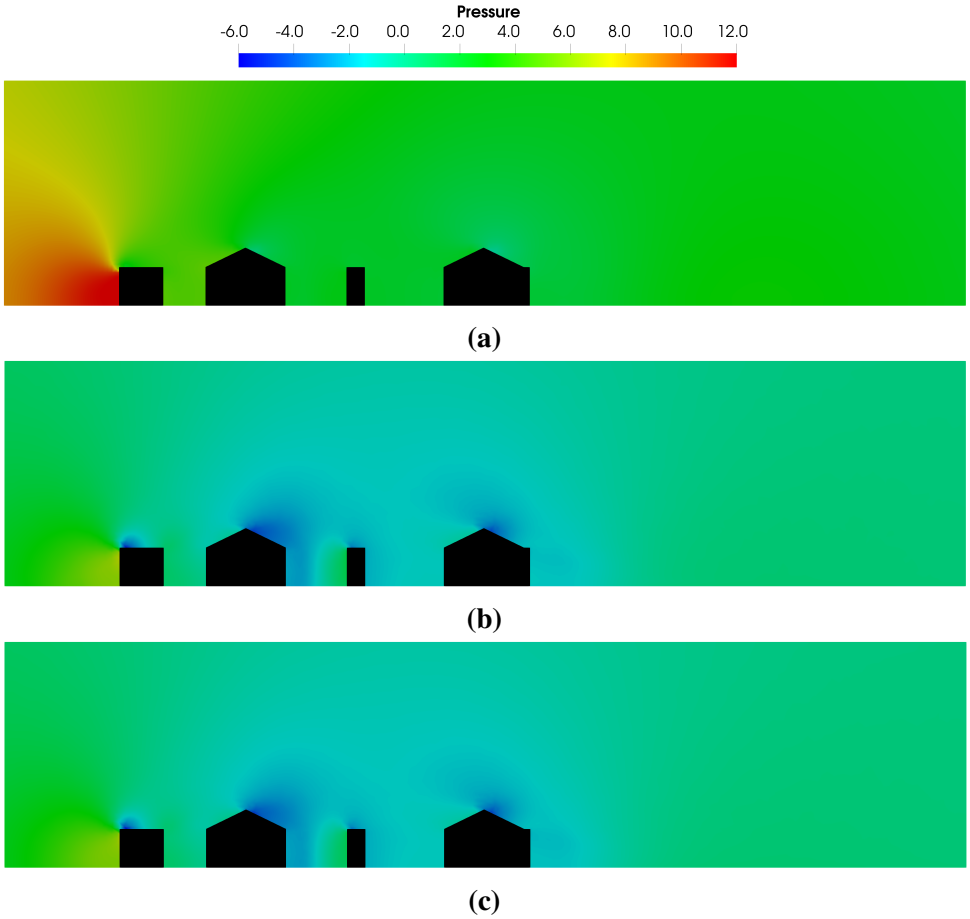


Figure 5.18: Pressure in the $x - z$ plane for the flow over an intersection case (CEDVAL B1-5) obtained from the present method (a), the ANSYS Fluent Coupled scheme ($k - \varepsilon$) (b), and the OpenFOAM SIMPLE scheme ($k - \varepsilon$) (c).

Table 5.7: Number of multigrid cycles and solver time for present method to reach a residual of 10^{-10} for the flow over an intersection case (CEDVAL B1-7) with each plane and line update direction.

Sweep Direction	**Number of Cycles**	**Solver time (s)**
Plane x, Line y	Stalled at residual $\sim 10^{-6}$	
Plane x, Line z	13	115.16
Plane y, Line x	31	292.93
Plane y, Line z	Stalled at residual $\sim 10^{-8}$	
Plane z, Line x	12	93.20
Plane z, Line y	12	92.09

5.6.4 Discussion

The main takeaway from the results above is that the coupled solver method introduced in Chapter 3 with the immersed boundary method from Chapter 4 and the zero-equation turbulence

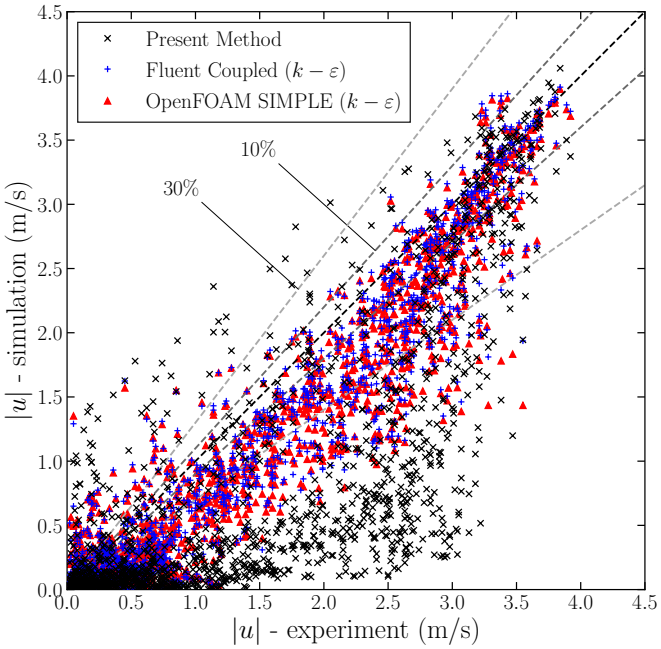


Figure 5.19: Comparison of all experimental data points with methods tested for the flow over an intersection case (CEDVAL B1-5). Points with perfect agreement with experiment lie on the diagonal line. Also indicated lines which show the boundary of 10% and 30% error.

model is in many cases two orders of magnitude faster than standard commercial CFD solvers using the $k-\varepsilon$ model. This significant speed-up comes at the expense of accuracy. The accuracy will be discussed shortly, however we divert our attention to the source of this significant speed-up. In Chapter 3, it was shown that the basic smoother was already significantly faster than the SIMPLE and coupled schemes tested here for laminar steady problems. This demonstrates the faster convergence rate of the smoother. Furthermore, the simplicity of the immersed boundary method along with the fact that rectilinear grids are used means that computationally, the cost of an iteration for the present method is comparability cheap. This is one half of the reason. The second comes from the zero-equation turbulence model used. The $k-\varepsilon$ introduces an additional two transport equations that need to be solved in addition to the Navier-Stokes equations. The direct cost that comes from needing to solve these transport equations is significant, however the additional stiffness in the governing equations that arises from the use of these models is a bigger contributor. The additional stiffness results in a lower convergence rate and hence more iterations for the solver to converge. This is evident in the results above where the $k-\varepsilon$ models required more under-relaxation in order to obtain a stable solution. This is another benefit of the zero-equation model: the better conditioning of the governing equations make it easier for the practitioner to obtain stable solutions to the problem.

Regarding accuracy for the predicted velocity fields, the results in this chapter demonstrate

that the present method is up to 25% less accurate on average in the worst case, and in some simpler geometries, it is competitive with the $k-\varepsilon$ model. The validation tests in Chapters 3 and 4 show very good agreement for laminar flow problems, so one can conclude that the largest source of error is the treatment of turbulence. It seems that for simple rectangular prisms (the cases in Secs. 5.6.1 and 5.6.2), the zero-equation model works reasonably well considering its simplicity and is competitive with the $k-\varepsilon$ model. Indeed, the model was derived using wind tunnel data from such simple geometries [168], so one would expect this result. However, for more complicated geometries, such as the one considered in Section 5.6.3, the model does not seem to generalise as well as the $k-\varepsilon$ model. The zero-equation model predicted significantly different pressure fields in terms of magnitude than the $k-\varepsilon$ models, though qualitatively speaking, they both contained the same salient features such as a high pressure region immediately upstream of the blocks and low pressure regions above and downstream of the block. Unfortunately, pressure measurements is not available from the experimental data. For the present application, the discrepancy in the pressure field is not too concerning since it is not used in the calculation of the dispersion of a gas in the flow field. Nonetheless, the errors in pressure fields can be largely attributed to the turbulence model, since the pressure coefficients calculated for the cases in Chapter 4.5 showed good agreement with experimental and numerical data.

Beyond the issue of determining the correct problem specific parameters in the model which often requires a level of guesswork (Table 5.1), the form of the zero-equation model itself was obtained to fit the flow field of a particular urban flow wind tunnel experiment. The assumption is then that it can generalise well to other urban flow problems with different geometries. Perhaps most important is the fundamental limitation that algebraic models (at least the ones considered here) are local in space and time. Models such as the $k-\varepsilon$ model take into account the history and transport effects of turbulence which the zero-equation models simply cannot do. Zero-equation models only really have validity under the assumption of local equilibrium: where the turbulence at a given point is dissipated at the same rate as it is produced, thereby exerting no influence on the flow field at other points. For the flows considered here, this is not the case [140, 151]. Practically, what this means is that the verified range of applicability for this model is relatively narrow – it is restricted to geometries which represent built environments.

Beyond this fundamental limitation of the zero-equation model, the $k-\varepsilon$ models tested here both make use of wall functions for near wall treatment. In some cases, the use of wall functions in the $k-\varepsilon$ models resulted in noticeably better prediction of the flow in the near wall region, as was shown in Figure 5.3a for the 40° cube case for example. However, in the cases with the more complex geometries, where there are regions of impinging flow and significant separation, the

benefit of wall functions is not so clear. Despite these shortcomings, so long as these algebraic models can provide solutions at a significantly reduced computational cost, they will remain to be genuinely useful for applications where results are needed in a short amount of time.

It should not be forgotten that the primary application of this fast running CFD model is to provide a wind field for the calculation of pollutant dispersion in urban environments. Hence the importance of the errors discussed above depend on their effect on the accuracy of the dispersion calculation. It may be possible that the pollutant dispersion calculations are not highly sensitive to the inaccuracies observed in the present method. On the other hand, the effect of these inaccuracies could be amplified in the dispersion calculation, leading to even more erroneous predictions. This should be explored through the implementation and validation testing of a dispersion model within the code. However, this is out of the scope of this work, and is left for future work.

At the start of this section, it was mentioned that for the present method, the number of coarse levels used was the maximum that would allow for a stable solution. For the case of the isolated block, a stable solution could be achieved as long as there was at least 1 - 2 cells between the solid block and the domain boundary. Choosing the coarsest grid level to be this coarse was not necessary, and using beyond 3 coarse levels did not provide any benefit in terms of performance. For the array of blocks (Section 5.6.2) and the intersection (Section 5.6.3) cases, the coarse level had to be chosen so that at least 2 cells were present between solid walls to achieve a stable solution. Otherwise upon coarsening, the solid would be "merged" at these regions, and no coarse grid correction would be applied there, giving an unstable solution. For the meshes used here, this put a limit of 2 coarse levels. While good performance was still achieved here, it is not difficult to imagine scenarios in complex urban-like geometries where the finest mesh would not allow for any coarsening unless it was refined excessively, which would generally be counterproductive. This could be addressed in a number of different ways, including adaptive coarsening [176, 177], or coarsening the grid by a factor of less than 2 in each direction [178]. This is a fundamental issue to the multigrid concept which requires some careful thought and is an important point for further research.

5.7 Summary

In this chapter, a zero-equation turbulence model was added to the steady solver developed in Chapters 3 and 4 to allow for the solution of turbulent flows. The zero-equation turbulence models selected were formulated specifically for flows in built environments. While it is understood that

the range of applicability for zero-equation models is particularly small, especially for complex recirculating flows, they were chosen here on the grounds that savings in computation time would make up for the loss in accuracy. Indeed, compared to the $k - \varepsilon$ model with the SIMPLE scheme in OpenFOAM and the coupled solver in ANSYS Fluent, speed-ups of around two orders of magnitude on the same hardware were observed. For a simple isolated block, and regular array of rectangular blocks the accuracy of the zero-equation model was competitive with the $k - \varepsilon$ model. For the more complicated case for the flow over an intersection, the zero-equation model was about 25% less accurate than the $k - \varepsilon$ model. This drop in accuracy is expected for the zero-equation model for cases that have less similarity to the case that it was created with. The tests were run on computer which is representative of high end consumer grade hardware at the time of writing. For the present method, all cases were solved in less than one minute, while the $k - \varepsilon$ models took over one hour or more for the same mesh size. These results demonstrate that the present approach is capable of calculating flow field solutions within a time frame that sufficient for emergency response scenarios and rapid turnaround time.

CHAPTER 6

Conclusions and Further Research

6.1 Summary of current research

This work set out to develop a complete solver methodology which could provide rapid wind field solutions to the flow in urban-like geometries. The aim was to solve problems significantly faster than current CFD packages, without resorting to diagnostic approaches such as the mass-conservative method of Röckle [23], which lack the physics of the momentum equations. In particular, the usage strategy required a model which could run useful problems on desktop hardware without having to resort to HPC resources which may not always be readily available. The fact that the resulting code is not massively parallel is not a weakness, but a conscious choice made that was driven by the use case. This was achieved by taking the approach of developing a method to solve the steady-state incompressible Navier-Stokes equations which prioritised simplicity, robustness, and efficiency. The solver development was broken up into three fundamental stages: the basic solver method, the treatment of complex geometries, and turbulence modelling.

The solver was based on the finite volume method, with collocated grids being used due to their simplicity in implementation. A fully coupled approach was taken to solve the steady Navier-Stokes equations. Unlike segregated methods, the more implicit pressure velocity coupling during the solution procedure results in a more efficient and robust method. In Chapter 3, a matrix free coupled solver method for collocated grids was developed, that is analogous to the Vanka smoother on staggered grids. To the authors knowledge, there has not been a matrix free coupled method like the Vanka smoother applied on collocated grids up until the present work. Coupled methods for collocated grids generally make use of a "black-box" sparse linear solver package to solve the coupled equations, and while this can be effective, a matrix free method is worthwhile to have because of its simplicity and efficiency. The coupled solver was

implemented within an FAS multigrid method. Tests for the 3D steady laminar lid driven cavity and the 3D steady laminar backwards facing step were performed to verify and validate the basic method. Performance was compared to the SIMPLE scheme in OpenFOAM and the coupled solver in ANSYS Fluent (both of which are collocated grid solvers). The present method showed speed-ups of 1 - 2 orders of magnitude in terms of solver time compared to these methods. Even without FAS multigrid, the base coupled smoother showed excellent performance, up to an order of magnitude speed-up in most cases. Furthermore, the multigrid implementation showed the optimal linear scaling with problem size. The present method was implemented in parallel using OpenMP for shared memory CPUs. Using multi-colour ordering, the method was parallelised in a number of ways: by points, lines and planes, with the idea being that plane parallelisation would be more efficient in terms of cache locality for a shared memory implementation. Scaling tests showed that this was indeed the case, with solver times being significantly better for the plane parallel, followed by lines, then points. The method described in Chapter 3 formed the foundation for a highly efficient steady solver upon which the rest of the solver was to be built on.

To allow for the flow over complex urban-like geometries, the immersed boundary method was used for its simplicity, efficiency, and ease of meshing – which can be a very time consuming process in itself. Despite these features, ghost cell immersed boundary methods in their basic form do not conserve mass globally, even in a mass-conservative finite volume framework where local mass conservation is satisfied in the fluid domain. Also, when used with coupled solvers on collocated grids, correct implementation of MWI at the boundaries is not straightforward. Finally, solid boundaries with corners are also difficult to treat naturally with traditional ghost cell methods. Addressing these issues was the subject of Chapter 4. An immersed boundary method which reconstructs field variables onto cell faces instead of ghost cells directly was introduced. The reconstructed face values where added to the discrete equations through source terms, which worked in analogy to ghost cells. This approach meant that ghost cells could take on multiple values. This fixed the issue of MWI and corners could be accounted for naturally. Mass-conservation was then enforced by applying a correction to the face fluxes at the boundary that was based on the global mass conservation error. This mass flux correction was very simple and efficient to calculate. The immersed boundary method was used within the present FAS scheme, where the immersed boundary was applied on all grid levels. Validation and verification testing for a range of steady problems showed the method was second order accurate and had good agreement with experiments and unstructured mesh solvers. Furthermore, the FAS scheme showed linear scaling with problem size even with the immersed boundary method, demonstrating an efficient multigrid implementation.

Finally, in Chapter 5 an eddy viscosity turbulence model was included so that turbulent flows could be solved. This was the final element required to create a complete package which could obtain fast, useful results for the wind field in an urban geometry. In line with this aim of producing rapid solutions, a zero-equation turbulence model designed for the flow in outdoor environments was used. Although zero-equation turbulence models do not include transport effects, which can limit their range of applicability, the significant savings in computational time make them attractive compared to one and two-equations models if they can be shown to be sufficiently accurate. Three urban test cases from the University of Hamburg CEDVAL experimental reference data set [63] were carried out, and the present method was compared to the ANSYS Fluent coupled solver with the $k-\varepsilon$ model, and the OpenFOAM SIMPLE scheme with the $k-\varepsilon$ model. For the isolated angled cube case, the average error of the present method from experiment was less than 30%, with the $k-\varepsilon$ models performing only a few percent better. For the array of rectangular blocks, the zero equation model had average errors of about 40%, with the $k-\varepsilon$ models again only performing a few percent better on average. For these two cases, it was believed that the lack of wall functions was the biggest drawback in terms of accuracy compared to the $k-\varepsilon$ models. For the flow across an intersection case, the superiority of the $k-\varepsilon$ model in terms of accuracy was more apparent, with the zero-equation model having an average error of about 75%, compared to the $k-\varepsilon$ models, which were on the order of 50%. The $k-\varepsilon$ model generalises to complex flows better than the zero-equation model – an expected result. This is all put into perspective however when the solver time is considered. All these cases were solved in less than one minute on a modern desktop computer using the present method. Using the same number of cells and the same hardware, the $k-\varepsilon$ models took over 25 minutes to solve the isolated cube case, while they took over 1 hour (and in most cases closer to 3 hours) to solve the array of blocks and the intersection cases. That is a speed-up of over two orders of magnitude in most cases. This speed-up comes partly from the efficiency of the basic solver method, as was demonstrated in Chapters 3 and 4, but also from the simplicity and robustness (numerically speaking) of the zero-equation model. The zero-equation model does not result in a stiff set of equations in the same way the $k-\varepsilon$ model does, so the convergence rate is significantly faster.

The basic solver elements that were introduced in this thesis – the coupled solver, the immersed boundary method, and the use of the zero-equation turbulence model – were all developed with the goal of simplicity and efficiency in mind, particularly for steady problems. Together, they form a complete solver framework which is optimised for calculating the steady-state wind field in complex urban like geometries. Regarding the trade-off between accuracy and efficiency, the

present method sits somewhere between conventional RANS solvers which use more complete complete two-equation turbulence models, and diagnostic approaches such as that of Röckle which use empirical parametrisations for the initial wind field. Fundamentally, the capability of the core of the solver is no different to other standard RANS solvers, with the only difference being the use of a simpler algebraic turbulence model. This fact, combined with the efficient solver method developed here makes the present method significantly faster than these RANS models, while retaining benefits of having a method that is based on solving the Navier-Stokes equations.

6.2 Recommendations for future research

As with all research, the ideas and methods presented in this thesis are far from complete, and there are many avenues for further work. Below is a list of a number of areas which in the author's opinion are the natural next steps and important areas for further research and development.

- Add dispersion calculation capability for a passive scalar (airborne contaminant). This is the natural next step in this work as the ultimate goal of the solver developed here is to allow for the fast calculation of the dispersion of airborne contaminants in urban environments. This includes verification and validation tests of the scalar dispersion calculation. These tests will also provide further insight on whether the zero-equation turbulence model and the over all accuracy of the wind field will suffice for the intended application.

- Addition of wall functions. The present method with the zero-equation model did not use wall functions, and in some of the turbulent test cases in Chapter 5 it was believed that the addition of wall functions could improve accuracy near the wall. Adding wall functions is not expected to increase the computational cost significantly, so it is worthwhile to explore. The only potential drawback would be its effect on the convergence of the solver. Furthermore, it is unclear how much the addition of wall functions would be useful in terms of accuracy for the highly separated flows encountered in urban environments, such as the case tested in Section 5.6.3.

- Development of more efficient turbulence modelling techniques. While the results in Chapter 5 demonstrated the utility of the zero-equation model for this application, particularly regarding computational time, it was clear that it did not generalise well to more complex geometries when compared to the $k - \varepsilon$ model. This is to be expected when using such a simple model, and it should be understood that this is the trade-off that

must be made if significant speed-ups are desired. However, this highlights a need for turbulence models which can facilitate fast running solutions by being simple to calculate, and that do not introduce significant stiffness to the equations. Recent research into a range of approaches from traditional optimization to machine learning methods has identified several promising alternatives to traditional turbulence modelling approaches that are worth exploring [173].

- Use of other parallel programming models, in particular GPUs. The present method has only been implemented for shared memory CPUs using OpenMP, however development in general purpose GPU computing has been significant in recent years, and high power desktop grade GPUs have become much more accessible. Due to their massively parallel architecture, GPUs are much better suited for memory bound applications which require high throughput. Indeed, the parallel scaling tests in Chapter 3 show that this is the case. Ideally, parallel portability layers such as RAJA [179] and Kokkos [180] should be used to allow for more seamless targeting of different parallel architectures.

- Overall development of the model pipeline. This work has been focused purely on the methods used for calculating the flow field. It was assumed that model data such as boundary conditions and geometry are readily available. However, this is generally not the case, and obtaining this data from measurements or other models can itself be a challenging and time consuming task. Furthermore, if the incoming model information is not accurate, then the final results of the model will not be reliable. Likewise, the data obtained from the present method must be processed, and useful quantities for the problem at hand need to be calculated rapidly. All these elements should integrate together seamlessly to allow the practitioner to obtain reliable predictions quickly. The development of these methods has broader value beyond fast-response applications in urban environments, as it contributes to making the CFD-based engineering design processes simpler and less time consuming for the practitioner.

References

[1] F. W. Roos, W. W. Willmarth, Some experimental results on sphere and disk drag, AIAA journal 9 (1971) 285–291.

[2] T. A. Johnson, Numerical and Experimental Investigation of Flow past a Sphere up to a Reynolds Number of 300, The University of Iowa, 1996.

[3] M. J. Brown, Urban dispersion—challenges for fast response modeling, Los Alamos Natl. Lab. Rep. LA-UR-04 5129 (2004).

[4] H. Bouchiba, S. Santoso, J.-E. Deschaud, L. Rocha-Da-Silva, F. Goulette, T. Coupez, Computational fluid dynamics on 3d point set surfaces, Journal of Computational Physics: X 7 (2020) 100069. doi:10.1016/j.jcpx.2020.100069.

[5] P. R. Urech, M. O. Mughal, C. Bartesaghi-Koc, A simulation-based design framework to iteratively analyze and shape urban landscapes using point cloud modeling, Computers, Environment and Urban Systems 91 (2022) 101731. doi:10.1016/j.compenvurbsys.2021.101731.

[6] C. Sun, F. Zhang, P. Zhao, X. Zhao, Y. Huang, X. Lu, Automated simulation framework for urban wind environments based on aerial point clouds and deep learning, Remote Sensing 13 (2021). doi:10.3390/rs13122383.

[7] O. Oldrini, P. Armand, C. Duchenne, S. Perdriel, M. Nibart, Accelerated time and high-resolution 3d modeling of the flow and dispersion of noxious substances over a gigantic urban area—the emergencies project, Atmosphere 12 (2021). doi:10.3390/atmos12050640.

[8] N. Onodera, Y. Idomura, Y. Hasegawa, H. Nakayama, T. Shimokawabe, T. Aoki, Real-time tracer dispersion simulations in oklahoma city using the locally mesh-refined lattice boltzmann method, Boundary-Layer Meteorology 179 (2021) 187–208. doi:10.1007/s10546-020-00594-x.

[9] H. Zhou, W. Song, K. Xiao, Simulating flow and hazardous gas dispersion by using wrf-cfd coupled model under different atmospheric stability conditions, Atmosphere 13 (2022). doi:10.3390/atmos13071072.

[10] H. Nakayama, N. Onodera, D. Satoh, H. Nagai, Y. Hasegawa, Y. Idomura, Development of local-scale high-resolution atmospheric dispersion and dose assessment system, Journal of Nuclear Science and Technology 59 (2022) 1314–1329. doi:10.1080/00223131.2022.2038302.

[11] D. J. Sailor, A review of methods for estimating anthropogenic heat and moisture emissions in the urban environment, International journal of climatology 31 (2011) 189–199.

[12] J. Amorim, V. Rodrigues, R. Tavares, J. Valente, C. Borrego, Cfd modelling of the aerodynamic effect of trees on urban air pollution dispersion, Science of the Total Environment 461 (2013) 541–551.

[13] J. Gallagher, C. Lago, How parked cars affect pollutant dispersion at street level in an urban street canyon? a cfd modelling exercise assessing geometrical detailing and pollutant decay rates, Science of the Total Environment 651 (2019) 2410–2418.

[14] B. Maronga, S. Banzhaf, C. Burmeister, T. Esch, R. Forkel, D. Fröhlich, V. Fuka, K. F. Gehrke, J. Geletič, S. Giersch, et al., Overview of the palm model system 6.0, Geoscientific Model Development 13 (2020) 1335–1372.

[15] S. Klasky, H. Abbasi, J. Logan, M. Parashar, K. Schwan, A. Shoshani, M. Wolf, S. Ahern, I. Altintas, W. Bethel, et al., In situ data processing for extreme-scale computing, Proceedings of SciDAC (2011) 1–16.

[16] H. Yi, M. Rasquin, J. Fang, I. A. Bolotnov, In-situ visualization and computational steering for large-scale simulation of turbulent flows in complex geometries, in: 2014 IEEE International Conference on Big Data (Big Data), 2014, pp. 567–572. doi:10.1109/BigData.2014.7004275.

[17] A. Mohamad, Lattice boltzmann method, volume 70, Springer, 2011.

[18] T. Krüger, H. Kusumaatmaja, A. Kuzmin, O. Shardt, G. Silva, E. M. Viggen, The lattice Boltzmann method, volume 10, Springer, 2017.

[19] J. J. Monaghan, Smoothed particle hydrodynamics, Reports on progress in physics 68 (2005) 1703.

[20] S. Li, W. K. Liu, Meshfree and particle methods and their applications, Applied Mechanics Reviews 55 (2002) 1–34.

[21] M. Neophytou, A. Gowardhan, M. Brown, An inter-comparison of three urban wind models using oklahoma city joint urban 2003 wind field measurements, Journal of Wind Engineering and Industrial Aerodynamics 99 (2011) 357–368. doi:10.1016/j.jweia.2011.01.010.

[22] G. Iaccarino, A. Ooi, P. Durbin, M. Behnia, Reynolds averaged simulation of unsteady separated flow, International Journal of Heat and Fluid Flow 24 (2003) 147–156.

[23] R. Röckle, Bestimmung der Strömungsverhältnisse im Bereich komplexer Bebauungsstrukturen, na, 1990.

[24] C. A. Sherman, A mass-consistent model for wind fields over complex terrain, Journal of Applied Meteorology and Climatology 17 (1978) 312 – 319. doi:10.1175/1520-0450(1978)017<0312:AMCMFW>2.0.CO;2.

[25] B. Singh, B. S. Hansen, M. J. Brown, E. R. Pardyjak, Evaluation of the quic-urb fast response urban wind model for a cubical building array and wide building street canyon, Environmental Fluid Mechanics 8 (2008) 281–312. doi:10.1007/s10652-008-9084-5.

[26] A. A. Gowardhan, M. J. Brown, E. R. Pardyjak, Evaluation of a fast response pressure solver for flow around an isolated cube, Environmental Fluid Mechanics 10 (2010) 311–328. doi:10.1007/s10652-009-9152-5.

[27] G. Tinarelli, G. Brusasca, O. Oldrini, D. Anfossi, S. Trini Castelli, J. Moussafir, Micro-Swift-Spray (MSS): A New Modelling System for the Simulation of Dispersion at Microscale. General Description and Validation, volume 17, 2007, pp. 449–458.

[28] Y. Wang, D. R. Miller, D. E. Anderson, M. L. McManus, A lagrangian stochastic model for aerial spray transport above an oak forest, Agricultural and Forest Meteorology 76 (1995) 277–291. doi:https://doi.org/10.1016/0168-1923(95)02221-I.

[29] Y. Wang, C. Williamson, D. Garvey, S. Chang, J. Cogan, Application of a multigrid method to a mass-consistent diagnostic wind model, Journal of Applied Meteorology 44 (2005) 1078–1089. doi:10.1175/jam2262.1.

[30] H. Kaplan, N. Dinar, A lagrangian dispersion model for calculating concentration distribution within a built-up domain, Atmospheric Environment 30 (1996) 4197–4207. doi:10.1016/1352-2310(96)00144-6.

[31] N. Bagal, E. Pardyjak, M. Brown, Improved upwind cavity parameterization for a fast response urban wind model (2003).

[32] N. Bagal, B. Singh, E. Pardyjak, M. Brown, Implementation of rooftop recirculation parameterization into the quic fast response urban wind model, in: 5th Symposium on the Urban Environment, 2004.

[33] M. J. Brown, A. A. Gowardhan, M. Nelson, M. Williams, E. R. Pardyjak, Evaluation of the quic wind and dispersion models using the joint urban 2003 field experiment dataset, 2009.

[34] B. Singh, E. Pardyjak, M. Brown, Testing of a far-wake parameterization for a fast response urban wind model (2006).

[35] S. Pol, N. Bagal, B. Singh, M. Brown, E. Pardyjak, Implementation of a new rooftop recirculation parameterization into the quic fast response urban wind model, 86th AMS Annual Meeting (2006).

[36] A. N. Hayati, R. Stoll, J. J. Kim, T. Harman, M. A. Nelson, M. J. Brown, E. R. Pardyjak, Comprehensive evaluation of fast-response, reynolds-averaged navier–stokes, and large-eddy simulation methods against high-spatial-resolution wind-tunnel data in step-down street canyons, Boundary-Layer Meteorology 164 (2017) 217–247. doi:10.1007/s10546-017-0245-2.

[37] P. S. Arya, Introduction to micrometeorology, Elsevier, 2001.

[38] F. Pasquill, F. Smith, Atmospheric diffusion.: Study of the dispersion of windborne material from industrial and other sources., JOHN WILEY & SONS, 605 THIRD AVE., NEW YORK, NY 10016, USA. 1983. (1983).

[39] M. D. Williams, M. A. Nelson, M. J. Brown, Condensed QUIC-PLUME Theory Guide for QUIC-FST, Report LA-UR-19-24263, Los Alamos National Lab. (LANL), Los Alamos, NM (United States), 2019.

[40] S. Hanna, J. White, J. Trolier, R. Vernot, M. Brown, A. Gowardhan, H. Kaplan, Y. Alexander, J. Moussafir, Y. Wang, C. Williamson, J. Hannan, E. Hendrick, Comparisons of ju2003 observations with four diagnostic urban wind flow and lagrangian particle dispersion models, Atmospheric Environment 45 (2011) 4073–4081. doi:10.1016/j.atmosenv.2011.03.058.

[41] A. A. Gowardhan, Towards understanding complex flow phenomenon in urban areas using numerical tools, Ph.D. thesis, Department of Mechanical Engineering, University of Utah, 2008.

[42] A. A. Gowardhan, E. R. Pardyjak, I. Senocak, M. J. Brown, A cfd-based wind solver for an urban fast response transport and dispersion model, Environmental Fluid Mechanics 11 (2011) 439–464. doi:10.1007/s10652-011-9211-6.

[43] P. Kopka, S. Potempski, A. Kaszko, M. Korycki, Urban dispersion modelling capabilities related to the udinee intensive operating period 4, Boundary-Layer Meteorology 171 (2019) 465–489. doi:10.1007/s10546-018-0399-6.

[44] J. H. Ferziger, M. Perić, R. L. Street, Computational methods for fluid dynamics, springer, 2019.

[45] P. Tamamidis, G. Zhang, D. N. Assanis, Comparison of pressure-based and artificial compressibility methods for solving 3d steady incompressible viscous flows, Journal of Computational Physics 124 (1996) 1–13.

[46] D. Kwak, C. Kiris, J. Dacles-Mariani, An assessment of artificial compressibility and pressure projection methods for incompressible flow simulations, in: Sixteenth International Conference on Numerical Methods in Fluid Dynamics: Proceedings of the Conference Held in Arcachon, France, 6–10 July 1998, Springer, 2007, pp. 177–182.

[47] J. Afful, A review of hpc-accelerated cfd in national security and defense, arXiv preprint arXiv:2504.07837 (2025).

[48] M. Yang, G. Oh, J.-I. Choi, A multi-gpu based les urban wind flow solver for real-time simulation, in: International Conference on Building Energy and Environment, Springer, 2022, pp. 199–202.

[49] I. Senocak, J. C. Thibault, M. Caylor, Rapid-response urban cfd simulations using a gpu computing paradigm on desktop supercomputers (2009).

[50] H. Wang, C. Peng, W. Li, C. Ding, T. Ming, N. Zhou, Porous media: A faster numerical simulation method applicable to real urban communities, Urban Climate 38 (2021) 100865.

[51] Y. Du, B. Blocken, S. Abbasi, S. Pirker, Efficient and high-resolution simulation of pollutant dispersion in complex urban environments by island-based recurrence cfd, Environmental Modelling & Software 145 (2021) 105172.

[52] X. Shao, Z. Liu, S. Zhang, Z. Zhao, C. Hu, Pignn-cfd: A physics-informed graph neural network for rapid predicting urban wind field defined on unstructured mesh, Building and Environment 232 (2023) 110056.

[53] M. Mortezazadeh, J. Zou, M. Hosseini, S. Yang, L. Wang, Estimating urban wind speeds and wind power potentials based on machine learning with city fast fluid dynamics training data, Atmosphere 13 (2022) 214.

[54] S. J. Low, V. S. Raghavan, H. Gopalan, J. C. Wong, J. Yeoh, C. C. Ooi, Fastflow: Ai for fast urban wind velocity prediction, in: 2022 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE, 2022, pp. 147–154.

[55] J. Stam, Real-time fluid dynamics for games, in: Proceedings of the game developer conference, volume 18, 2003, p. 25.

[56] W. Zuo, Advanced simulations of air distributions in buildings (2010) 1–190. URL: https://docs.lib.purdue.edu/dissertations/AAI3413824.

[57] M. Jin, W. Zuo, Q. Chen, Simulating natural ventilation in and around buildings by fast fluid dynamics 64 (2013) 273–289. doi:10.1080/10407782.2013.784131.

[58] S. Zheng, Z. J. Zhai, Y. Wang, Y. Xue, L. Duanmu, W. Liu, Evaluation and comparison of various fast fluid dynamics modeling methods for predicting airflow around buildings 15 (2022) 1083–1095. doi:10.1007/s12273-021-0860-1.

[59] M. Mortezazadeh, L. L. Wang, M. Albettar, S. Yang, Cityffd–city fast fluid dynamics for urban microclimate simulations on graphics processing units, Urban Climate 41 (2022) 101063.

[60] M. A. George, N. Williamson, S. W. Armfield, A coupled block implicit solver for the incompressible navier–stokes equations on collocated grids, Computers & Fluids 284 (2024) 106426.

[61] M. George, N. Williamson, S. W. Armfield, Coupled fas multigrid for the incompressible navier-stokes equations on collocated grids, Available at SSRN 5460400 (Under review at Computers & Fluids) (2025).

[62] M. A. George, N. Williamson, S. W. Armfield, Mass-conserving ghost cell immersed boundary method with multigrid for coupled navier-stokes solvers, Journal of Computational Physics (2025) 114276.

[63] University of Hamburg, Meteorological Institute, CEDVAL: Compilation of Experimental Data for Validation of Microscale Dispersion Models, https://www.mi.uni-hamburg.de/en/arbeitsgruppen/windkanallabor/data-sets.html, 2024.

[64] F. M. L. M. M. Darwish, The finite volume method in computational fluid dynamics, 2016.

[65] P. Bartholomew, F. Denner, M. H. Abdol-Azis, A. Marquis, B. G. M. van Wachem, Unified formulation of the momentum-weighted interpolation for collocated variable arrangements, Journal of Computational Physics 375 (2018) 177–208. doi:10.1016/j.jcp.2018.08.030.

[66] C. M. Klaij, On the stabilization of finite volume methods with co-located variables for incompressible flow, Journal of Computational Physics 297 (2015) 84–89.

[67] ANSYS, Inc., ANSYS Fluent Theory Guide, Release 2021 R1, 2021. Version 2021 R1.

[68] S. Patankar, Numerical heat transfer and fluid flow, Taylor & Francis, 2018.

[69] H. van Santen, D. Lathouwers, C. Kleijn, H. van den Akker, Influence of segregation on the efficiency of finite volume methods for the incompressible Navier-Stokes equations, Numerical Developments in CFD. Proceedings of the Fluids Engineering Division of the Americal Society of Mechanical Engineers (1996) 151–158.

[70] A. Pascau, C. Pérez, F. J. Serón, A comparison of segregated and coupled methods for the solution of the incompressible navier-stokes equations, Communications in numerical methods in engineering 12 (1996) 617–630.

[71] H. Wang, H. Wang, F. Gao, P. Zhou, Z. J. Zhai, Literature review on pressure–velocity decoupling algorithms applied to built-environment cfd simulation, Building and Environment 143 (2018) 671–678.

[72] M. Darwish, I. Sraj, F. Moukalled, A Coupled Incompressible Flow Solver on Structured Grids, Numerical Heat Transfer, Part B: Fundamentals 52 (2007) 353–371. doi:10.1080/10407790701372785, publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10407790701372785.

[73] M. Darwish, I. Sraj, F. Moukalled, A coupled finite volume solver for the solution of incompressible flows on unstructured grids, Journal of Computational Physics 228 (2009) 180–201. doi:10.1016/j.jcp.2008.08.027.

[74] G. B. Deng, J. Piquet, X. Vasseur, M. Visonneau, A new fully coupled method for computing turbulent flows, Computers & Fluids 30 (2001) 445–472. doi:10.1016/S0045-7930(00)00025-6.

[75] I. Ammara, C. Masson, Development of a fully coupled control-volume finite element method for the incompressible Navier–Stokes equations, International Journal for Numerical Methods in Fluids 44 (2004) 621–644. doi:10.1002/fld.662, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.662.

[76] S. P. Vanka, Block-implicit multigrid solution of navier-stokes equations in primitive variables, Journal of Computational Physics 65 (1986) 138–158.

[77] M. Paisley, Multigrid solution of the incompressible Navier–Stokes equations for three-dimensional recirculating flow: coupled and decoupled smoothers compared, International Journal for Numerical Methods in Fluids 30 (1999) 441–459. doi:10.1002/(SICI)1097-0363(19990630)30:4<441::AID-FLD848>3.0.CO;2-K.

[78] V. John, L. Tobiska, Numerical performance of smoothers in coupled multigrid methods for the parallel solution of the incompressible Navier–Stokes equations, International Journal for Numerical Methods in Fluids 33 (2000) 453–473. doi:10.1002/1097-0363(20000630)33:4<453::AID-FLD15>3.0.CO;2-0.

[79] M. Anselmann, M. Bause, Efficiency of local Vanka smoother geometric multigrid preconditioning for space-time finite element methods to the Navier–Stokes equations, PAMM 23 (2023) e202200088. doi:10.1002/pamm.202200088, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pamm.202200088.

[80] P. Bauer, V. Klement, T. Oberhuber, V. Žabka, Implementation of the vanka-type multigrid solver for the finite element approximation of the navier–stokes equations on gpu, Computer Physics Communications 200 (2016) 50–56.

[81] L. Claus, M. Bolten, Nonoverlapping block smoothers for the stokes equations, Numerical Linear Algebra with Applications 28 (2021) e2389.

[82] M. F. Paisley, N. M. Bhatti, Comparison of Multigrid Methods for Neutral and Stably Stratified Flows over Two-Dimensional Obstacles, Journal of Computational Physics 142 (1998) 581–610. doi:10.1006/jcph.1998.5915.

[83] C.-H. Bruneau, K. Khadra, Highly parallel computing of a multigrid solver for 3d navier–stokes equations, Journal of computational science 17 (2016) 35–46.

[84] S. W. Armfield, C. A. J. Fletcher, Numerical simulation of swirling flow in diffusers, International Journal for Numerical Methods in Fluids 6 (1986) 541–556. doi:10.1002/fld.1650060805.

[85] S. W. Armfield, C. A. J. Fletcher, Pressure related instabilities of reduced Navier-Stokes equations for internal flows, Communications in Applied Numerical Methods 2 (1986) 377–383. doi:10.1002/cnm.1630020408.

[86] D. R. Reddy, Global pressure relaxation procedure for laminar and turbulent incompressible flows with strong interaction and separation, University of Cincinnati, 1983.

[87] S. G. Rubin, A Review of Marching Procedures for Parabolized Navier-Stokes Equations, in: T. Cebeci (Ed.), Numerical and Physical Aspects of Aerodynamic Flows, Springer, Berlin, Heidelberg, 1982, pp. 171–185. doi:10.1007/978-3-662-12610-3_11.

[88] D. J. Mavriplis, An assessment of linear versus nonlinear multigrid methods for unstructured mesh solvers, Journal of Computational Physics 175 (2002) 302–325.

[89] F. Lien, M. Leschziner, Multigrid acceleration for recirculating laminar and turbulent flows computed with a non-orthogonal, collocated finite-volume scheme, Computer Methods in Applied Mechanics and Engineering 118 (1994) 351–371. doi:https://doi.org/10.1016/0045-7825(94)90007-8.

[90] S. D. Ryan, R. C. Ripley, F.-S. Lien, F. Zhang, Accelerated convergence for city-scale flow fields using immersed boundaries and coupled multigrid, Journal of Wind Engineering and Industrial Aerodynamics 241 (2023) 105541.

[91] F. X. Trias, O. Lehmkuhl, A. Oliva, C. D. Pérez-Segarra, R. Verstappen, Symmetry-preserving discretization of navier–stokes equations on collocated unstructured grids, Journal of Computational Physics 258 (2014) 246–267.

[92] D. Santos, J. A. Hopman, C. D. Pérez-Segarra, F. Trias, On a symmetry-preserving unconditionally stable projection method on collocated unstructured grids for incompressible flows, Journal of Computational Physics 523 (2025) 113631.

[93] ANSYS, Inc., ANSYS Fluent User's Guide, Release 2021 R1, 2021. URL: https://www.ansys.com/products/fluids/ansys-fluent, version 2021 R1.

[94] OpenFOAM Foundation, OpenFOAM, Version 9, 2021. URL: https://openfoam.org/version/9/.

[95] Z. Zhu, A Novel Computational Method for Complex Three-dimensional Flows, Ph.D. thesis, University of Sydney, 1991.

[96] L. Claus, Multigrid smoothers for saddle point systems, Ph.D. thesis, Universitätsbibliothek Wuppertal Wuppertal, 2019.

[97] R. S. Montero, I. M. Llorente, M. D. Salas, Robust multigrid algorithms for the navier–stokes equations, Journal of Computational Physics 173 (2001) 412–432.

[98] Y. Saad, Iterative methods for sparse linear systems, SIAM, 2003.

[99] V. Henson, et al., Multigrid methods nonlinear problems: an overview, Computational imaging 5016 (2003) 36–48.

[100] G. Guennebaud, B. Jacob, et al., Eigen v3, http://eigen.tuxfamily.org, 2010.

[101] Intel Core i9-11900F Processor (16 M Cache, Up to 5.20 GHz) — Specifications, https://www.intel.com/content/www/us/en/products/sku/212254/intel-core-i911900f-processor-16m-cache-up-to-5-20-ghz/specifications.html, 2025.

[102] GNU Compiler Collection (GCC), Version 12.2.0, GNU Project, 2022. URL: https://gcc.gnu.org/.

[103] Intel Xeon Platinum 8274 Processor (35.75 M Cache, 3.10 GHz) — Specifications, https://www.intel.com/content/www/us/en/products/sku/192487/intel-xeon-platinum-8274-processor-35-75m-cache-3-10-ghz/specifications.html, 2025.

[104] National Computational Infrastructure, Gadi supercomputer, 2020. URL: https://nci.org.au/our-systems/hpc-systems.

[105] C. Lameter, Numa (non-uniform memory access): An overview: Numa becomes more common because memory controllers get close to execution units on microprocessors., Queue 11 (2013) 40–51.

[106] R. van der Pas, Mastering openmp performance, Webinar, OpenMP, 2021. URL: https://www.openmp.org/events/webinar-mastering-openmp-performance/, presented as part of the OpenMP webinar series.

[107] C. S. Peskin, The fluid dynamics of heart valves: experimental, theoretical, and computational methods., Annual review of fluid mechanics 14 (1982) 235–259.

[108] R. Mittal, G. Iaccarino, Immersed boundary methods, Annu. Rev. Fluid Mech. 37 (2005) 239–261.

[109] R. Verzicco, Immersed boundary methods: Historical perspective and future outlook, Annual Review of Fluid Mechanics 55 (2023) 129–155.

[110] R. Ghias, R. Mittal, H. Dong, A sharp interface immersed boundary method for compressible viscous flows, Journal of Computational Physics 225 (2007) 528–553.

[111] Y.-H. Tseng, J. H. Ferziger, A ghost-cell immersed boundary method for flow in complex geometry, Journal of computational physics 192 (2003) 593–623.

[112] E. A. Fadlun, R. Verzicco, P. Orlandi, J. Mohd-Yusof, Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations, Journal of computational physics 161 (2000) 35–60.

[113] J. Lee, J. Kim, H. Choi, K.-S. Yang, Sources of spurious force oscillations from an immersed boundary method for moving-body problems, Journal of computational physics 230 (2011) 2677–2695.

[114] J. Lee, D. You, An implicit ghost-cell immersed boundary method for simulations of moving body problems with control of spurious force oscillations, Journal of Computational Physics 233 (2013) 295–314.

[115] J. Kim, D. Kim, H. Choi, An immersed-boundary finite-volume method for simulations of flow in complex geometries, Journal of Computational Physics 171 (2001) 132–150. doi:https://doi.org/10.1006/jcph.2001.6778.

[116] A. Mark, B. G. van Wachem, Derivation and validation of a novel implicit second-order accurate immersed boundary method, Journal of Computational Physics 227 (2008) 6660–6680. doi:https://doi.org/10.1016/j.jcp.2008.03.031.

[117] A. Shinn, M. Goodwin, S. Vanka, Immersed boundary computations of shear-and buoyancy-driven flows in complex enclosures, International Journal of Heat and Mass Transfer 52 (2009) 4082–4089.

[118] M. H. A. Azis, F. Evrard, B. van Wachem, An immersed boundary method for incompressible flows in complex domains, Journal of Computational Physics 378 (2019) 770–795.

[119] D. Goldstein, R. Handler, L. Sirovich, Modeling a no-slip flow boundary with an external force field, Journal of computational physics 105 (1993) 354–366.

[120] G. Iaccarino, R. Verzicco, Immersed boundary technique for turbulent flow simulations, Appl. Mech. Rev. 56 (2003) 331–347.

[121] R. Mittal, H. Dong, M. Bozkurttas, F. Najjar, A. Vargas, A. Von Loebbecke, A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries, Journal of computational physics 227 (2008) 4825–4852.

[122] D. Pan, A general boundary condition treatment in immersed boundary methods for incompressible navier-stokes equations with heat transfer, Numerical Heat Transfer, Part B: Fundamentals 61 (2012) 279–297.

[123] C. Yan, W.-X. Huang, G.-X. Cui, C. Xu, Z.-S. Zhang, A ghost-cell immersed boundary method for large eddy simulation of flows in complex geometries, International Journal of Computational Fluid Dynamics 29 (2015) 12–25.

[124] S. Kang, G. Iaccarino, F. Ham, P. Moin, Prediction of wall-pressure fluctuation in turbulent flows with an immersed boundary method, Journal of Computational Physics 228 (2009) 6753–6772.

[125] I. N. Yildiran, N. Beratlis, F. Capuano, Y.-H. Loke, K. Squires, E. Balaras, Pressure boundary conditions for immersed-boundary methods, Journal of computational physics 510 (2024) 113057.

[126] P. G. Tucker, Z. Pan, A cartesian cut cell method for incompressible viscous flow, Applied Mathematical Modelling 24 (2000) 591–606.

[127] M. Kirkpatrick, S. Armfield, J. Kent, A representation of curved boundaries for the solution of the navier–stokes equations on a staggered three-dimensional cartesian grid, Journal of Computational Physics 184 (2003) 1–36.

[128] K. Madabushi, S. Ghosh, A mass-conservative immersed boundary method for compressible flows, in: Proc. Twelfth ICCFD Conf, 2024.

[129] F. Moukalled, L. Mangani, M. Darwish, F. Moukalled, L. Mangani, M. Darwish, The finite volume method, Springer, 2016.

[130] C. Chi, A. Abdelsamie, D. Thévenin, A directional ghost-cell immersed boundary method for incompressible flows, Journal of Computational Physics 404 (2020) 109122.

[131] A. Khosronejad, S. Kang, I. Borazjani, F. Sotiropoulos, Curvilinear immersed boundary method for simulating coupled flow and bed morphodynamic interactions due to sediment transport phenomena, Advances in water resources 34 (2011) 829–843.

[132] The CGAL Project, Cgal, computational geometry algorithms library, http://www.cgal.org, 2023. Version 5.6, released 28 July 2023.

[133] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272. doi:10.1038/s41592-019-0686-2.

[134] B. Launder, D. B. Spalding, Mathematical models of turbulence, Academic Press, San Diego, CA, 1972.

[135] D. Wilcox, Turbulence modeling-an overview, in: 39th aerospace sciences meeting and exhibit, 2001, p. 724.

[136] O. Kwon, F. E. Ames, Advanced k-epsilon modeling of Heat Transfer, Technical Report, 1995.

[137] R. Prieler, M. Demuth, D. Spoljaric, C. Hochenauer, Numerical investigation of the steady flamelet approach under different combustion environments, Fuel 140 (2015) 731–743.

[138] C.-H. Chang, R. N. Meroney, Numerical and physical modeling of bluff body flow and dispersion in urban street canyons, Journal of Wind Engineering and Industrial Aerodynamics 89 (2001) 1325–1334.

[139] J. L. Santiago, A. Martilli, Simulation of must experiment using rans k-epsilon model: validation against wind tunnel measurements and analysis of spatial average properties, in:

Proceedings of the 11th International Conference on Harmonization Within Atmospheric Dispersion Modelling for Regulatory Purposes, Cambridge, United Kingdom, 2007.

[140] S. B. Pope, Turbulent flows, Measurement Science and Technology 12 (2001) 2020–2021.

[141] Y. Tominaga, A. Mochida, R. Yoshie, H. Kataoka, T. Nozu, M. Yoshikawa, T. Shirasawa, Aij guidelines for practical applications of cfd to pedestrian wind environment around buildings, Journal of wind engineering and industrial aerodynamics 96 (2008) 1749–1761.

[142] Y. Tominaga, A. Mochida, S. Murakami, S. Sawaki, Comparison of various revised k–ε models and les applied to flow around a high-rise building model with 1: 1: 2 shape placed within the surface boundary layer, Journal of wind engineering and industrial aerodynamics 96 (2008) 389–411.

[143] S. Murakami, R. Ooka, A. Mochida, S. Yoshida, S. Kim, Cfd analysis of wind climate from human scale to urban scale, Journal of wind engineering and industrial aerodynamics 81 (1999) 57–81.

[144] V. Yakhot, S. A. Orszag, S. Thangam, T. Gatski, C. Speziale, Development of turbulence models for shear flows by a double expansion technique, Physics of Fluids A: Fluid Dynamics 4 (1992) 1510–1520.

[145] B. Launder, Modeling flow-induced oscillations in turbulent flow around a square cylinder, in: ASME Fluid Eng. Conference, 1993, pp. 20–24.

[146] T.-H. Shih, W. W. Liou, A. Shabbir, Z. Yang, J. Zhu, A new k-ε eddy viscosity model for high reynolds number turbulent flows, Computers & fluids 24 (1995) 227–238.

[147] Y. Tominaga, T. Stathopoulos, Numerical simulation of dispersion around an isolated cubic building: Comparison of various types of k-ε models, Atmospheric Environment 43 (2009) 3200–3210.

[148] T. L. Chan, G. Dong, C. W. Leung, C. S. Cheung, W. Hung, Validation of a two-dimensional pollutant dispersion model in an isolated street canyon, Atmospheric environment 36 (2002) 861–872.

[149] J. Hang, Z. Luo, M. Sandberg, J. Gong, Natural ventilation assessment in typical open and semi-open urban environments under various wind directions, Building and environment 70 (2013) 318–333.

[150] Y. Miao, S. Liu, Y. Zheng, S. Wang, Y. Li, Numerical study of traffic pollutant dispersion within different street canyon configurations, Advances in meteorology 2014 (2014) 458671.

[151] D. Wilcox, Turbulence Modeling for CFD, DCW Industries, 2004. URL: https://books.google.com.au/books?id=aolIPgAACAAJ.

[152] F. R. Menter, Two-equation eddy-viscosity turbulence models for engineering applications, AIAA journal 32 (1994) 1598–1605.

[153] H. Shen, Q. He, Y. Huang, Y. Liu, Wind around tall building: A comparison between rans and les, in: Proceedings of the 8th International Symposium on Heating, Ventilation and Air Conditioning: Volume 1: Indoor and Outdoor Environment, Springer, 2013, pp. 625–633.

[154] B. Streichenberger, R. Chakir, B. Jouy, J. Waeytens, Simulation and validation of cfd turbulent airflow at pedestrian level using 3d ultrasonic anemometer in the controlled urban area "sense-city", Journal of Wind Engineering and Industrial Aerodynamics 219 (2021) 104801.

[155] M. Shirzadi, Y. Tominaga, P. A. Mirzaei, Experimental and steady-rans cfd modelling of cross-ventilation in moderately-dense urban areas, Sustainable Cities and Society 52 (2020) 101849.

[156] M. Terziev, T. Tezdogan, A. Incecik, Application of eddy-viscosity turbulence models to problems in ship hydrodynamics, Ships and Offshore Structures 15 (2020) 511–534.

[157] N. HIRATA, T. HINO, A comparative study of zero-and one-equation turbulence models for ship flows, in: Journal of the Kansai Society of Naval Architects, Japan 234, The Japan Society of Naval Architects and Ocean Engineers, 2000, pp. 17–24.

[158] M. Bazargan, M. Mohseni, Algebraic zero-equation versus complex two-equation turbulence modeling in supercritical fluid flows, Computers & Fluids 60 (2012) 49–57.

[159] A. Cioncolini, J. R. Thome, Algebraic turbulence modeling in adiabatic and evaporating annular two-phase flow, International Journal of Heat and Fluid Flow 32 (2011) 805–817.

[160] Q. Chen, W. Xu, A zero-equation turbulence model for indoor airflow simulation, Energy and buildings 28 (1998) 137–144.

[161] K. Nering, K. Nering, Validation of modified algebraic model during transitional flow in hvac duct, Energies 14 (2021) 3975.

[162] K. Kirtley, Renormalization group based algebraic turbulence model for three-dimensional turbomachinery flows, AIAA journal 30 (1992) 1500–1506.

[163] W. Dawes, A comparison of zero and one equation turbulence modelling for turbomachinery calculations, volume 79047, American Society of Mechanical Engineers, 1990.

[164] J. Srebric, Q. Chen, L. R. Glicksman, Validation of a zero-equation turbulence model for complex indoor airflow simulation, ASHRAE Transactions 105 (1999) 414.

[165] Y. Qian, J. Srebric, Development and validation of an algebraic turbulence model for outdoor airflow and contaminant simulations around a building, International Journal of Building, Urban, Interior and Landscape Technology 1 (2011) 47–56. URL: https://ph02.tci-thaijo.org/index.php/BUILT/article/view/170316. doi:10.56261/built.v1.170316.

[166] D. Davidovic, Improvements in Numerical Airflow Modeling Around Multiple Buildings, Ph.D. thesis, Pennsylvania State University, 2010.

[167] D. Davidovic, J. Liu, M. Heidarinejad, J. Srebric, Airflow study for a cluster of campus buildings using different turbulence modeling approaches, International Journal of Building, Urban, Interior and Landscape Technology 3 (2014) 33–58.

[168] J. Liu, J. Srebric, N. Yu, A rapid and reliable numerical method for predictions of outdoor thermal environment in actual urban areas, in: Heat Transfer Summer Conference, volume 55492, American Society of Mechanical Engineers, 2013, p. V003T21A008.

[169] C. Li, X. Li, Y. Su, Y. Zhu, A new zero-equation turbulence model for micro-scale climate simulation, Building and Environment 47 (2012) 243–255.

[170] J. Liu, M. Heidarinejad, G. Pitchurov, L. Zhang, J. Srebric, An extensive comparison of modified zero-equation, standard k-ε, and les models in predicting urban airflow, Sustainable cities and society 40 (2018) 28–43.

[171] M. Shirzadi, P. A. Mirzaei, Y. Tominaga, Rans model calibration using stochastic optimization for accuracy improvement of urban airflow cfd modeling, Journal of Building Engineering 32 (2020) 101756.

[172] G. Calzolari, W. Liu, Deep learning to develop zero-equation based turbulence model for cfd simulations of the built environment, in: Building Simulation, volume 17, Springer, 2024, pp. 399–414.

[173] S. Liu, L. Xu, B. Chen, Z. Deng, P. Li, R. Zhao, Q. Chen, Recent progress in optimization of rans turbulence models for accurate urban airflow and contaminant dispersion simulations, Sustainable Cities and Society (2025) 106336.

[174] S. Patankar, Numerical heat transfer and fluid flow, CRC press, 1980.

[175] M. Popovac, K. Hanjalic, Compound wall treatment for rans computation of complex turbulent flows and heat transfer, Flow, turbulence and combustion 78 (2007) 177–202.

[176] M. C. Thompson, J. H. Ferziger, An adaptive multigrid technique for the incompressible navier-stokes equations, Journal of computational Physics 82 (1989) 94–121.

[177] D. Hartmann, M. Meinke, W. Schröder, An adaptive multilevel multigrid formulation for cartesian hierarchical grid methods, Computers & Fluids 37 (2008) 1103–1125.

[178] C. Liu, W. Henshaw, Multigrid with nonstandard coarse-level operators and coarsening factors, Journal of Scientific Computing 94 (2023) 58.

[179] D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, B. S. Ryujin, T. R. W. Scogland, RAJA: Portable Performance for Large-Scale Scientific Applications (2019) 71–81. doi:{10.1109/P3HPC49588.2019.00014}.

[180] C. Trott, L. Berger-Vergiat, D. Poliakoff, S. Rajamanickam, D. Lebrun-Grandie, J. Madsen, N. Al Awar, M. Gligoric, G. Shipman, G. Womeldorff, The kokkos ecosystem: Comprehensive performance portability for high performance computing, Computing in Science Engineering 23 (2021) 10–18. doi:10.1109/MCSE.2021.3098509.